C H A P T E R 3

Base System Architecture

Ease on the surface requires power and speed at the core, and the modern, 32-bit architecture of Windows 95 meets these requirements. Freed from the limitations of MS-DOS, Windows 95 preemptively multi-tasks for better PC responsiveness —so users will no longer have to wait while the system copies files, for example —and also delivers increased robustness and protection for applications. Windows 95 also provides the foundation for a new generation of easier, more powerful multi-threaded 32-bit applications. And most importantly, Windows 95 delivers this power and robustness on today's average PC platform while scaling well to take advantage of additional memory and CPU cycles.

The mission of Windows 95 is to deliver a complete, integrated, operating system, that offers modern 32-bit operating system technology, and includes built-in connectivity support. In addition to the high-level mission of Windows 95, market requirements must be met to deliver a high performance, robust, and completely backwards-compatible operating system.

This section discusses the base architecture used by Windows 95. The base architecture covers low-level system services for managing memory, accessing disk devices, and providing robust support for running applications. Windows 95 delivers a modern 32-bit operating system that is compatible with existing software and hardware, and delivers a platform for a new generation of applications.

Summary of Improvements over Windows 3.1

Improvements made to the base architecture of Windows 95 result in many benefits to users. A summary of some of the key improvements include:

- Fully integrated 32-bit protected-mode operating system, eliminating the need for a separate copy of MS-DOS
- Preemptive multitasking, and multithreading support—improving system responsiveness and smooth background processing
- 32-bit installable file systems including VFAT, CDFS, and network redirectors providing better performance, use of long filenames, and an open architecture supporting future growth
- 32-bit device drivers available throughout the system, delivering improved performance and intelligent memory use
- Complete 32-bit kernel, including memory management, scheduler, and process management
- Improved system-wide robustness and cleanup after an application ends or crashes, delivering a more stable and reliable operating environment
- More dynamic environment configuration reducing the need for users to tweak their system
- Improved system capacity, including better system resource limits to address issues Windows 3.1 users encountered when running multiple applications

Fully-Integrated Operating System

The first thing that users of Windows 3.1 and MS-DOS will see when they turn their computer on (or perhaps won’t see) is the lack of an MS-DOS command prompt from which they would need to invoke Windows. Windows 95 is a tightly integrated operating system that features a preemptive multitasking kernel that boots directly into the graphical user interface, yet provides full compatibility with the MS-DOS operating system.

Many of components in Windows 95 overcome limitations inherent in MS-DOS and Windows 3.1, moreover, the improvements do not come at the cost of compatibility with existing software, hardware, or computing environment.

A Preemptive Multitasking Operating System

The job of the operating system is to provide services to the applications that are running in the system and, in a multitasking environment, to provide support for allowing more than one application to run concurrently. Windows 3.1 allowed multiple applications to run concurrently in the system in a *cooperative* multitasking manner. The Windows 3.1 operating system required an application to check the message queue every once in a while in order to allow the operating system to relinquish control to other running applications. Applications that did not check the message queue on a frequent basis would effectively hog all of the CPU time and prevent the user from switching to another running task.

Windows 95 uses a *preemptive* multitasking mechanism for running Win32–based applications and the operating system will take control away from or give control to another running task depending on the needs of the system. This means that unlike Win16–based applications, Win32–based applications do not need to *yield* to other running tasks in order to multitask in a friendly manner (Win16–based applications are still cooperatively multitasked for compatibility reasons). Windows 95 provides a mechanism for Win32–based applications to take advantage of the preemptive multitasking nature of the operating system to facilitate concurrent application design, called *multithreading*. A Win32–based application running in the system is called a *process* in terms of the operating system. Each process consists of at least a single *thread* of execution that identifies the code path flow as it is run by the operating system. A *thread* is a unit of code that can get a time slice from the operating system to run concurrently with other units of code, and must be associated with a process. However, a Win32–based application can *spawn* (or initiate) multiple threads for a given process to enhance the application for the user by improving throughput, enhancing responsiveness, and aiding background processing. Due to the preemptive multitasking nature of Windows 95, threads of execution will allow background code processing in a smooth manner.

For example, a word processing application (process) may implement multiple threads to enhance operation and simplify interaction with the user. The application may have a separate thread of code that responds to keys typed on the keyboard by the user to place characters in a document, while another thread is performing background operations such as spell-checking or pagination, while yet another thread is spooling a document to the printer in the background. Some Windows 3.1 applications that are available today may provide functionality similar to that just described, however because Windows 3.1 does not provide a mechanism for supporting multithreaded applications, it is up to the application vendor to implement their own threading scheme. The use of threads in Windows 95 facilitates application vendors to add asynchronous processing of information to their applications. Applications that use multithreading techniques in their applications will also be able to take advantage of improved processing performance available from Windows NT when using a symmetric multiprocessor (SMP) system by allowing different portions of the application code to run on different processors simultaneously (Windows NT uses a thread as the unit of code to schedule symmetrically among multiple processors).

Information about how Windows 95 runs MS-DOS–based applications in a preemptive manner (as Windows 3.1 does today), Win16–based applications in a cooperative manner (as Windows 3.1 does today), and Win32–based applications in a preemptive manner (as Windows NT does today), is provided later in this section.

No Need for CONFIG.SYS or AUTOEXEC.BAT

Windows 95 no longer needs a separate CONFIG.SYS or AUTOEXEC.BAT file as MS-DOS and Windows 3.1 require. Instead, Windows 95 is intelligent about the drivers and settings it needs to use and automatically will load the appropriate driver files or set the appropriate configuration settings during its boot process. If a CONFIG.SYS or AUTOEXEC.BAT file are present, the settings in these files will be used to set the global environment. For example, the default search path or the default appearance of the command prompt can be defined by using the appropriate entries in the AUTOEXEC.BAT file. While Windows 95 itself does not need a CONFIG.SYS or AUTOEXEC.BAT file, compatibility is maintained with existing software or environments that may require one or both of these files.

MS-DOS Not Really There?

Unlike Windows 3.1, Windows 95 is not built on top of real-mode operating system components for its interaction with the file system. This being said, the Windows 95 boot sequence does begin by loading real-mode operating system components that are compatible with MS-DOS. During the boot sequence, support for loading real-mode drivers and TSRs that may be identified in CONFIG.SYS or AUTOEXEC.BAT are processed. Since these drivers have been written to explicitly look for or use MS-DOS application support, the real-mode operating system components of Windows 95 help to maintain compatibility with software that users already have on their system. Once the real-mode drivers have been loaded, Windows 95 begins loading the protect-mode operating system components, and in some cases where a protect-mode Windows–based driver is provided, and will actually remove real-mode drivers from memory. More on this subject is discussed later in this guide.

32-Bit Versus 16-Bit Components

Windows 95 uses a combination of 32-bit and 16-bit code in order to provide a good balance between delivering compatibility with existing applications and drivers, decreasing the size of the operating system working set, and offering improved system performance over Windows 3.1. System reliability is also improved without the cost of compatibility or increased size.

Windows 95 is a 32-bit preemptive multitasking operating system that implements some 16-bit code to provide compatibility with existing applications. In general, 32-bit code is provided in Windows 95 to maximize the performance of the system, while 16-bit code balances the requirements for reducing the size of the system and maintaining compatibility with existing applications and drivers.

The design of Windows 95 deploys 32-bit code wherever it significantly improves performance without sacrificing application compatibility. Existing 16-bit code is retained where it is required to maintain compatibility, or where 32-bit code would increase memory requirements without significantly improving performance. All of the I/O subsystems and device drivers in Windows 95, such as networking and file systems, are fully 32-bit, as are all the memory management and scheduling components (the kernel and virtual memory manager). Figure 1 depicts the relative distribution of 32-bit versus 16-bit code present in Windows 95 for system-level services. As can be seen from the figure, the lowest-level services provided by the operating system kernel are provided as 32-bit code. Most of the remaining 16-bit code consists of hand-tuned assembly language, delivering performance that rivals some 32-bit code used by other operating systems available on the market today.

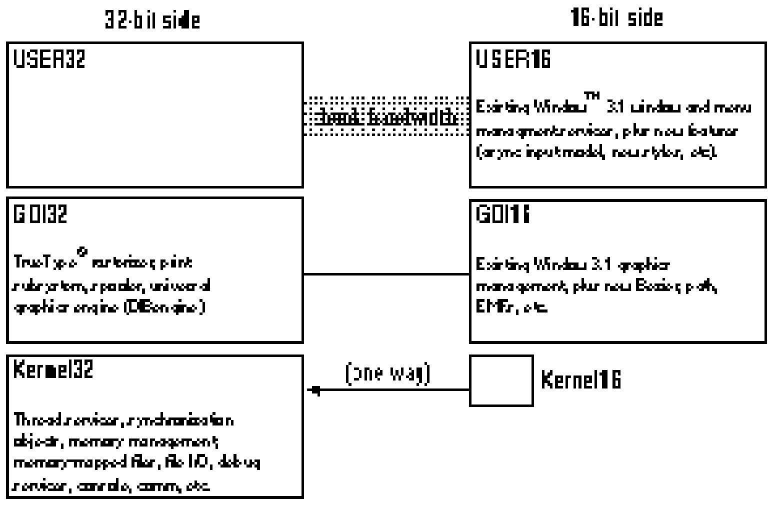

32-bit side
16-bit side
USER32
USER16
Existing Windows™ 3.1 window and menu management services, plus new features (async input model, new styles, etc).
GDI32
TrueType® rasterizing, print subsystem, spooler, universal graphics engine (DIBengine)
GDI16
Existing Windows 3.1 graphics management, plus new Beziers, path, EMFs, etc.
Kernel32
Thread services, synchronization objects, memory management, memory-mapped files, file I/O, debug services, console, comm, etc.
(one way)
Kernel16

Figure 1. Relative Code Distribution in Windows 95

Many functions provided by the Graphics Device Interface (GDI) have been moved to 32-bit code, including the spooler and printing subsystem, the font rasterizer, and the drawing operations performed by the graphics “DIB engine.” Much of the window management code (User) remains 16-bit to retain application compatibility.

In addition, Windows 95 improves upon the MS-DOS and Windows 3.1 environment by implementing many device drivers as 32-bit protected-mode code. Virtual device drivers in Windows 95 assume the functionality provided by many real-mode MS-DOS–based device drivers eliminating the need to load them in MS-DOS. This results in a minimal conventional memory footprint, improved performance, and improved reliability and stability of the system over MS-DOS–based device drivers.

Virtual Device Drivers—What is a VxD?

A virtual device driver (VxD) is a 32-bit, protected-mode driver that manages a system resource, such as a hardware device or installed software, so that more than one application can use the resource at the same time. To understand the improvements available in Windows 95 over the combination of MS-DOS and Windows 3.1, it is good to have a basic understanding of what a VxD is and the role virtual device drivers play in the Windows 95 environment.

The term *VxD* is used to refer to a general virtual device driver—the *x* represents the type of device driver. For example, a virtual device driver for a display device is known as a VDD, a virtual device driver for a timer device is a VTD, a virtual device driver for a printer device is a VPD, and so forth. Windows uses virtual devices to support multitasking for MS-DOS-based applications, virtualizing the different hardware components on the system to make it appear to each MS-DOS VM that it is executing on its own computer. Virtual devices work in conjunction with Windows to process interrupts and carry out I/O operations for a given application without disrupting how other applications run.

Virtual device drivers support all hardware devices for a typical computer, including the programmable interrupt controller (PIC), timer, direct-memory-access (DMA) device, disk controller, serial ports, parallel ports, keyboard device, math coprocessor, and display adapter. A virtual device driver can contain the device-specific code needed to carry out operations on the device. A virtual device driver is required for any hardware device that has settable operating modes or retains data over any period of time. In other words, if the state of the hardware device can be disrupted by switching between multiple applications, the device must have a corresponding virtual device. The virtual device keeps track of the state of the device for each application and ensures that the device is in the correct state whenever an application continues.

Although most virtual devices manage hardware, some manage only installed software, such as an MS-DOS device driver or a terminate-and-stay-resident (TSR) program. Such virtual devices often contain code that either emulates the software or ensures that the software uses data that applies only to the currently running application. ROM BIOS, MS-DOS, MS-DOS device drivers, and TSRs provide device-specific routines and operating system functions that applications use to indirectly access the hardware devices. Virtual device drivers are sometimes used to improve the performance of installed software; the 80386 and compatible microprocessors can run the 32-bit protected-mode code of a virtual device more efficiently than the 16-bit real-mode code of an MS-DOS device driver or TSR. In addition, performance is also enhanced by eliminating ring transitions that result in executing 32-bit applications that access 16-bit real-mode services—with virtual device drivers, the system can stay in protected-mode.

Windows 95 benefits from providing more device driver support implemented as a series of VxDs in the Windows environment, over the use of device drivers previously available as real-mode MS-DOS device drivers. Functionality that was previously supported as MS-DOS device drivers, but are now supported as VxDs in Windows 95 includes components such as:

- MS-DOS FAT file system
- SmartDrive
- CD-ROM file system
- Network card drivers and network transport protocols
- Network client redirector, and network peer server
- Mouse driver
- MS-DOS file sharing and locking support (SHARE.EXE TSR)
- Disk device drivers including support for SCSI devices
- DriveSpace (and DoubleSpace) disk compression

In Windows 95, VxDs provide improved performance due to a 32-bit code path and eliminating or reducing the need to mode switch between real and protected-mode, reduced conventional memory footprint by providing device driver and TSR functionality as protected-mode components that reside in extended memory, and improved system stability and reliability over using the MS-DOS device driver counterparts. Virtual device drivers can be identified by the use of a .VXD extension in Windows 95, or a .386 extension as a virtual device driver from Windows 3.1.

Layout of System Architecture for Windows 95

Figure 2 illustrates the layout of the base system architecture for Windows 95. Components of the system are divided between Ring 0 and Ring 3 code, offering different levels of system protection. The Ring 3 code is protected from other running processes by protection services provided by the Intel processor architecture. The Ring 0 code consists of the low-level operating system services such as the file system, and virtual machine manager.

This figure also depicts the way that MS-DOS–, Win16–, and Win32–based applications run in the system. The following areas of this section discuss the provisions that the system makes for running these applications.

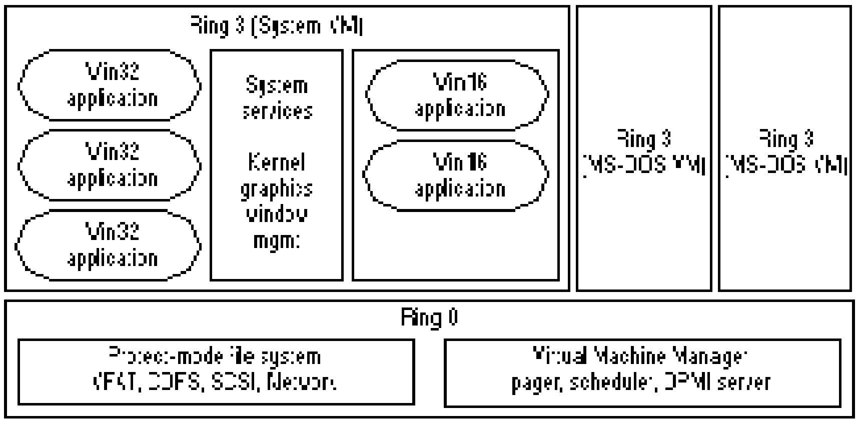


Figure 2. The Integrated Architecture of Windows 95 supports Running MS-DOS–, Win16–, and Win32–based Applications

Support for Win16–based Applications

16-bit Windows–based applications (Win16) run together within a unified address space, and are run in a cooperatively multitasking fashion as they do under Windows 3.1. Win16–based applications benefit from the preemptive multitasking of other system components including the 32-bit print and communications subsystem, and the improvements made in system robustness and protection from the system kernel in Windows 95.

When Win16–based application support was examined by the development team of Windows 95, three goals drove the architectural design based on customer needs, resource needs, and market needs: compatibility, size, and performance. Functionality such as running Win16–based applications together in the Win16 subsystem preemptively or running Win16–based applications in separate VMs was examined, however each option examined failed to meet the design goals set forth. The following discussion will provide some insight as to the architecture design of Windows 95 for running Win16–based applications in a fast, stable, and reliable way.

Compatibility

First and foremost, Windows 95 needs to run existing Win16–based applications without modification. This is extremely important to existing customers that want to take advantage of new functionality offered in Windows 95 such as 32-bit networking, but don’t want to have to wait until new Windows 95-enabled applications are available on the market.

Windows 95 builds upon the Windows 3.1 platform to provide support for running existing Win16–based applications and using existing Windows–based device drivers, while providing support for the next generation of 32-bit applications and components. Windows 95 extends the Windows 3.1 architecture in areas that have little or no impact on compatibility, as well as enhances the architecture to deliver a faster, more powerful 32-bit operating system.

Size

While many newer computer purchases are Intel 80486-based computers with 4MB or 8MB (or more) of memory, there are still a high percentage of 80386DX-based computers with 4MB of memory in use running Windows 3.1 today. To support the needs of the market, Windows 95 needs to run on a base platform of an Intel 80386DX-based computer with 4MB of RAM, to provide access to the new features and functionality provided, without requiring an upgrade of existing hardware or the addition of more RAM.

To meet its design goals, the Windows 95 development team designed Windows 95 to occupy no more working set than Windows 3.1 currently does, thereby insuring that any Win16–based application running at a perceived speed on a 4MB or 8MB computer (or greater) still runs at the same (or higher) speed under Windows 95 and does not suffer any performance degradation. To meet the required size goals of Windows 95, Win16–based applications run within a unified address space, resulting in little overhead beyond that required by Windows 3.1 to support running Windows–based applications. This allows Windows 95 to not only simply fit on a 4MB computer, but also to perform well. The architecture of Windows 95 includes innovative design features such as dynamically loadable VxDs to decrease the working set of components and memory requirements used by the operating system.

Meeting the size design goal (as well as to meet the compatibility goal), precluded the development team from adopting a strategy of running Win16–based applications in a separate VM by running a separate copy of Windows 3.1 on top of the operating system (thereby paying a several megabyte "memory tax" for each application) as OS/2 does, or emulating Windows 3.1 on top of the Win32 subsystem (thereby paying a "memory tax" for running Win16–based applications) as Windows NT does.

Running Win16–based applications in separate VMs is very expensive memory wise. This would require separate GDI, USER, and KERNEL code in each VM that is created, requiring the working set to increase by as much as 2MB for each Win16–based application that is running (as is required by OS/2 for Windows). If you have a computer with 16MB or more, this may not appear to be such a big deal. However, given the existing installed base of computers it would be impossible to run Win16–based applications in their own separate VMs in 4MB at all, and very difficult to run them in 8MB with the same level of performance as customers observe and expect under Windows 3.1 today.

Performance

Users expect their existing Win16 applications to run as fast or faster than they do under Windows 3.1. Win16–based applications will benefit from the 32-bit architecture of Windows 95 including the increased use of 32-bit device driver components and 32-bit subsystems, as will MS-DOS–based applications.

Win16–based applications run within a unified address space and interact with the system much as they do under Windows 3.1 today. Running Win16–based applications in separate VMs requires either a mapping of Win16 system components in each address space, as Windows NT does, or providing a separate copy of each system component in each address space, as OS/2 for Windows does. The additional memory overhead required for Win16 system components in each VM to run a Win16–based application has a negative impact on system performance.

Windows 95 balances the issue of system protection and robustness, with the desire for better system performance and improves on the system robustness over Windows 3.1. The improvements in this area are briefly discussed below, and are described in greater detail in a separate section of this guide.

Protection

The support for running Win16–based applications provides protection of the system from other running MS-DOS–based applications or Win32–based applications. Unlike Windows 3.1, an errant Win16–based application can not easily bring down the system or other running processes on the system. While Win32–based applications benefit the most from system memory protection, the robustness improvements present in Windows 95 result in a more stable and reliable operating environment than Windows 3.1.

Win16–based applications run within a unified address space, and cooperatively multitask as they do under Windows 3.1. The improvements made to overall system-wide robustness greatly enhance the system's ability to recover from an errant application, and lessens the likelihood of application errors due to improved clean up of the system. The occurrence of general protection faults (GPFs) under Windows 3.1 are most commonly caused by an application that writes over its own memory segments, rather than being caused by an application overwriting memory belonging to another application. Windows 3.1 did not recover gracefully when a Windows–based application crashed or hung. When an application was halted by the system due to a GPF, the system commonly left allocated resources in memory, causing the system to degenerate.

Due to improved protection in Windows 95, an errant Win16–based application can not easily bring down either the system as a whole, or other running MS-DOS or Win32–based applications, and can at most impact other running Win16–based applications.

Other protection improvements include the use of separate message queues for each running Win32–based application. The use of a separate message queue for the Win16 address space and for each running Win32–based application provides better recovery of the system and doesn't halt the system should a Win16–based application hang.

Robustness Improvements

System robustness is also greatly improved when running Win16–based applications over Windows 3.1. Windows 95 now tracks resources allocated by Win16–based applications and uses the information to clean up the system after an application exits or ends abnormally, thus freeing up unused resources for use by the rest of the system.

Robustness improvements is discussed later in a separate section of this guide.

Support for MS-DOS–based Applications

There are many improvements in Windows 95 for running MS-DOS–based applications over Windows 3.1. As with Windows 3.1, each MS-DOS–based application runs in its own "virtual machine" (VM). A VM takes advantage of the Intel 80386 (and higher) architecture allowing multiple 8086-compatible sessions to run on the CPU, allowing existing MS-DOS applications to run preemptively with the rest of the system. As with Windows 3.1, the use of virtual device drivers provide common regulated access to hardware resources, thereby making each application running in a virtual machine think it's running on its own individual computer, allowing applications not designed to multitask to run concurrently with other applications.

Windows 95 provides a flexible environment for running MS-DOS–based applications. Unlike Windows 3.1, where users sometimes needed to exit Windows in order to run MS-DOS–based applications that were either ill-behaved or required direct access to system resources. MS-DOS–based application compatibility is improved in Windows 95 so almost all MS-DOS–based applications should run under Windows 95.

Protection

VMs are fully protected from one another, as well as from other applications running on the system. This prevents errant MS-DOS–based applications from being able to overwrite memory occupied or used by system components or other applications. If an MS-DOS–based application attempts to access memory outside of its address space, the system will notify the user and the MS-DOS–based application will be ended.

Robustness Improvements

System robustness is also greatly improved when running MS-DOS–based applications over Windows 3.1. Robustness is discussed later in a separate section of this guide.

Improved Support for Running MS-DOS–based Applications

Windows 95 provides much better support for running MS-DOS–based applications within the Windows environment than Windows 3.1.

A detailed discussion of the improvements made to running MS-DOS–based applications is discussed in the section “Improved Support for Running MS-DOS–based Applications” later in this guide.

Support for Win32–based Applications

Win32–based applications can fully exploit and benefit more from the design of the Windows 95 architecture. In addition, each Win32–based application runs in its own fully-protected, private address space. This prevents other Win32–based applications from crashing each other, crashing other running MS-DOS–based applications, crashing running Win16–based applications, or crashing the Windows 95 system as a whole.

Win32–based applications feature the following benefits over Win16–based applications in Windows 95 or under Windows 3.1:

- Preemptive multitasking
- 32-bit Win32 APIs
- Long filename support
- Separate message queues
- Flat address space
- Memory Protection

Preemptive Multitasking

Unlike the cooperative multitasking used by Win16–based applications under Windows 3.1, 32-bit Win32–based applications are preemptively multitasked in Windows 95. The operating system kernel is responsible for scheduling the time allotted for running applications in the system, and support for preemptive multitasking results in smoother concurrent processing and prevents any one application from utilizing all system resources without permitting other tasks to run.

Win32–based applications can optionally implement threads to improve the granularity at which they multitask within the system. The use of threads by an application improves the interaction with the user and result in smoother multitasking operation.

Separate Message Queues

Under Windows 3.1, the system uses the point when an application checks the system message queue as the mechanism to pass control to another task, allowing that task to run in a cooperative manner. If an application doesn't check the message queue on a regular basis, or the application hangs and thus prevents other applications from checking the message queue, the system will keep the other tasks in the system suspended until the errant application is ended.

Each Win32–based application has its own separate message queue and is thus not affected by the behavior of other running tasks on their own message queues. If a Win16–based application hangs, or if another running Win32–based application crashes, a Win32–based application will continue to run preemptively and will still be able to receive incoming messages or event notifications.

Message queues are discussed in more detail in the "Robustness Improvements" section of this guide.

Flat Address Space

Win32–based applications benefit from improved performance and simpler construct due to being able to access memory in a linear fashion, rather being limited to the segmented memory architecture used by MS-DOS and Windows 3.1. In order to provide a means of accessing high amounts of memory using a 16-bit addressing model, the Intel CPU architecture provides support for accessing 64K chunks of memory at a time, called segments. Applications and the operating system suffer a performance penalty under this architecture due to the necessary manipulations required by the processor for mapping memory references from the segment/offset combination to the physical memory structure.

The use of a flat address space by the 32-bit components in Windows 95 and for Win32–based applications will allow application and device driver developers to write software without the limitations or design issues inherent with the segmented memory architecture used by MS-DOS and Windows 3.1.

Compatibility with Windows NT

Win32–based applications that exploit Win32 APIs common between Windows 95 and Windows NT can run without modification on either platform on Intel-based computers. The commonality of the Win32 API provides a consistent programmatic interface allowing application vendors to use a single development effort to leverage delivery of software that runs on multiple platforms. This provides scalability of applications and broadens the base of platforms available for running ISV or custom applications with minimal additional effort.

Application vendors are encouraged to develop applications either under Windows 95 or Windows NT, and test compatibility on both platforms.

Long Filename Support

Win32–based applications that call the file I/O functions supported by the Win32 API will benefit from the ability to support and manipulate filenames up to 255 characters, with no additional development effort. The Win32 APIs and common dialog support handles the work for manipulating long filenames, and the file system provides compatibility with MS-DOS and other systems by also maintaining the traditional 8.3 filename automatically. This eases the burden from the application developer.

Memory Protection

Each Win32–based application runs in its own private address, and is protected by the system from other applications or processes that are running in the system. Unlike running Win16–based applications under Windows 3.1, errant Win32–based applications under Windows 95 will only end themselves, rather than bring down the entire system if they attempt to access memory belonging to another application.

The use of separate message queues for Win32–based applications also protects to ensure that the system will continue to run if an application hangs or stops responding to messages or events.

Robustness Improvements

Win32–based applications benefit from the highest level of system robustness supported under Windows 95. Resources allocated for each Win32–based application is tracked on a per-thread basis and are automatically freed when the application ends. If an application hangs, users are able to perform a *local reboot* operation to end the hung application without affecting other running tasks, and the system will clean up properly.

Detailed information about robustness enhancements is discussed later in a separate section of this guide.

32-Bit File System Architecture

The file system in Windows 95 has been re-architected from Windows 3.1 to support the characteristics and needs of the multitasking nature of the kernel in Windows 95. The changes present in Windows 95 provide many benefits to the user and results in:

- Improved ease of use

  Ease of use is improved by support long filenames so users no longer need to reference files by the MS-DOS 8.3 filename structure—users can use up to 255 characters to identify their documents. Ease of use is also improved by hiding the filename extensions from users.

- Improved performance

  As in Windows for Workgroups 3.11, file I/O performance is improved dramatically over Windows 3.1 by featuring 32-bit protected-mode code for reading information from and writing information to the file system, reading and writing information from/to the disk device, and intelligent 32-bit caching mechanisms—a full 32-bit code path is available from the file system to the disk device.

- Improved system stability and reliability

  File system components implemented as 32-bit protected mode device drivers offer improved system stability and reliability over MS-DOS device driver counterparts due to being able to remain in protected-mode for code execution and leveraging existing driver technology first implemented in Windows NT and also available in Windows for Workgroups 3.11.

Architecture Overview

Windows 95 features a layered file system architecture that supports multiple file systems, and provides a protected-mode path from the application to the media device, resulting in improved file and disk I/O performance over Windows 3.1. Features of the new file system architecture include:

u Win32 API support

u Long filename support

u 32-bit FAT file system

u 32-bit CD-ROM file System

u Dynamic system cache for file and network I/O

u Open architecture for future system support

u Disk device driver compatibility with Windows NT

Figure 3 depicts the file system architecture used by Windows 95.

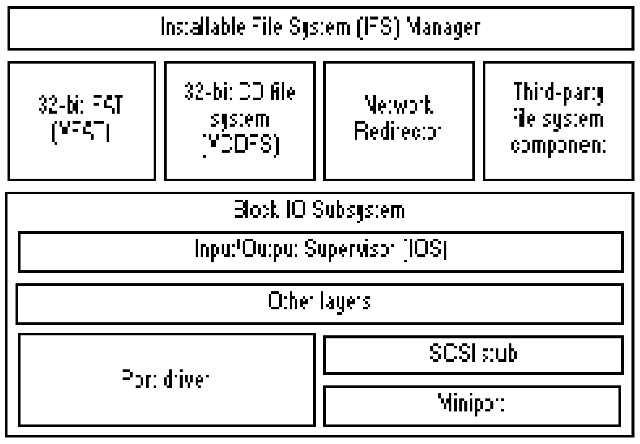

Installable File System (IFS) Manager
32-bit FAT (VFAT)
32-bit CD file system (VCDFS)
Network Redirector
Third-party file system component
Block IO Subsystem
Input/Output Supervisor (IOS)
Other layers
Port driver
SCSI stub
Miniport

Figure 3. File System Architecture in Windows 95

The file system architecture in Windows 95 is made up of the following components:

- Installable File System (IFS) Manager

  The IFS Manager is responsible for arbitrating access to different file system components.

- File system drivers

  The file system drivers layer includes access to file allocation table (FAT)-based disk devices, CD-ROM file systems, and redirected network device support.

- Block I/O subsystem

  The block I/O subsystem is responsible for interacting with the physical disk device.

We'll examine components of each of these layers in this section.

Installable File System Manager

Under MS-DOS and Windows 3.1, the MS-DOS Int 21h interrupt is responsible for providing access to the file system to manipulate file information on a disk device. In order to support redirected disk devices (for example, a network drive, or a CD-ROM drive), other system components such as the network redirector would hook the Int 21h function so that it could examine the file system request to determine whether it should handle the file I/O request, or let the base file system handle it. While this mechanism provided the ability to add on additional device drivers, some add-on components would be ill-behaved and would interfere with other installed drivers.

Another problem that was encountered with the MS-DOS–based file system was the difficulty in supporting the loading of multiple network redirectors to provide concurrent access to different network types. Windows for Workgroups provided support for running the Microsoft Windows Network redirector at the same time as an additional network redirector including Novell NetWare, Banyan VINES, and SUN PC-NFS, however support for running more than two network redirectors at the same time was not supported.

The key to friendly access to disk and redirected devices in Windows 95 is the Installable File System (IFS) Manager. The IFS manager is responsible for arbitrating access to file system devices, as well as other file system device components.

Windows 95 includes support for the following file systems:

- 32-bit FAT driver (VFAT)
- 32-bit CD-ROM file system driver (CDFS), and
- 32-bit network redirector for connectivity to Microsoft Network servers (e.g., Windows NT Server), along with a 32-bit network redirector to connect to Novell NetWare servers

Third-parties will use the IFS Manager APIs to provide a clean way of concurrently supporting multiple device types, adding additional disk device support and network redirector support.

32-bit File Access—Protected-mode FAT (VFAT) File System

The 32-bit VFAT driver provides a 32-bit protected-mode code path for manipulating the file system stored on a disk. It is also re-entrant and multi-threaded, providing smoother multi-tasking performance. The 32-bit file access driver is improved over that provided originally with Windows for Workgroups 3.11, and is compatible with more MS-DOS-device drivers and hard disk controllers.

Benefits of the 32-bit file access driver over MS-DOS–based driver solutions include:

- Dramatically improved performance and real-mode disk caching software
- No conventional memory used—replacement for real-mode SmartDrive
- Better multitasking when accessing information on disk—no blocking
- Dynamic cache support

Under MS-DOS and Windows 3.1, manipulation of the file allocation table (FAT) and writing or reading information to/from the disk is handled by the Int 21h MS-DOS function and is 16-bit real-mode code. Being able to manipulate the disk file system from protected-mode removes or reduces the need to transition to real-mode in order to write information to the disk through MS-DOS, which will result in a performance gain for file I/O access.

The 32-bit VFAT driver interacts with the block I/O subsystem to provide 32-bit disk access to more device types than is supported by Windows 3.1. Support is also provided for mapping to existing real-mode disk drivers that may be in use on a user's system. The combination of the 32-bit file access and 32-bit disk access drivers result in significantly improved disk and file I/O performance.

32-Bit Cache—VCACHE

The 32-bit VFAT works in conjunction with a 32-bit protected-mode cache driver (VCACHE), and replaces and improves on the 16-bit real-mode SmartDrive disk cache software provided with MS-DOS and Windows 3.1. The VCACHE driver features more intelligent caching algorithm than SmartDrive to cache information read from or written to a disk drive, and results in improved performance for reading information from cache. Also, the VCACHE driver is responsible for managing the cache pool for the CD-ROM File System (CDFS), and the provided 32-bit network redirectors.

Another big improvement in VCACHE over SmartDrive is that the memory pool used for the cache is *dynamic* and is based on the amount of available free system memory. Users no longer need to statically allocate a block of memory to set aside as a disk cache, the system automatically allocates or de-allocates memory used for the cache based on system use. The performance of the system will also scale better than Windows 3.1 or Windows for Workgroups 3.11, due to the intelligent cache use.

32-Bit CDFS—Protected-mode CD-ROM File System

The 32-bit protected-mode CD-ROM file system (CDFS) implemented in Windows 95 provides improved CD-ROM access performance over the real-mode MSCDEX driver in Windows 3.1 and is a full 32-bit ISO 9660 CD file system. The CDFS driver replaces the 16-bit real-mode MSCDEX driver, and features 32-bit protected-mode caching of CD-ROM data. The CDFS driver cache is dynamic and shares the cache memory pool with the 32-bit VFAT driver, requiring no configuration or static allocation on the part of the user.

Benefits of the new 32-bit CDFS driver include:

- No conventional memory used—replacement for real-mode MSCDEX
- Improved performance over MS-DOS–based MSCDEX and real-mode cache
- Better multitasking when accessing CD-ROM information—no blocking
- Dynamic cache support to provide a better balance between providing memory to run applications versus memory to serve as a disk cache

If MSCDEX is specified in the user's AUTOEXEC.BAT, the 32-bit CDFS driver will take over role played by the MSCDEX driver and communicate with the CD-ROM device. The use of MSCDEX is no longer necessary under Windows 95.

Users of CD-ROM multimedia applications will benefit greatly from the new 32-bit CDFS. Their multimedia applications will run smoother and information will be read from the CD-ROM quicker providing improved performance.

Disk Device Architecture—Block I/O Subsystem

The Block I/O Subsystem in Windows 95 improves upon the 32-bit disk access "FastDisk" device architecture used in Windows 3.1 to improved performance for the entire file system and a broader array of device support.

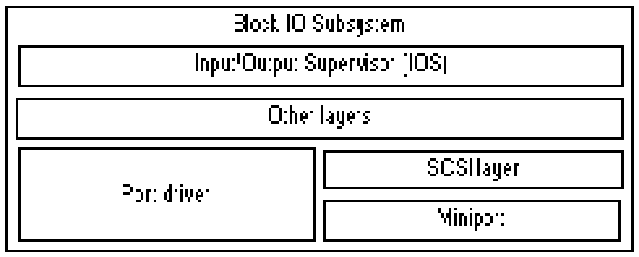


Figure 4. Architecture of Block I/O Subsystem in Windows 95

Components of the block I/O subsystem include the high-level I/O Supervisor (IOS) layer, which provides the interface to the block I/O subsystem to the higher layer components; the port driver, which represents a monolithic disk device driver; the SCSI layer, which provides a standard interface and driver layer to provide device-independent control code for SCSI devices; and the SCSI mini-port driver, which contains the device-dependent control code responsible for interacting with individual SCSI controllers.

The block I/O subsystem provides the following support in Windows 95:

- Fully Plug and Play-enabled architecture
- Support for mini-port drivers that are binary compatible with Windows NT
- Support for Windows 3.1 fast disk drivers for backwards compatibility
- Protected-mode drivers that take over real-mode MS-DOS device drivers if it is thought to be safe to do so
- The ability to support existing MS-DOS real-mode disk device drivers for compatibility

Let’s examine the different areas that make up the block I/O subsystem. Keep in mind that the configuration of the disk device driver layers is isolated from the user, so the explanation here is provided to facilitate an understanding of the components.

I/O Supervisor

The I/O Supervisor (IOS) provides services to file systems and drivers. The IOS is responsible for the queuing of file service requests and for routing the requests to the appropriate file system driver. The IOS also provides asynchronous notification of file system events to drivers that are installed.

Port Driver

The port driver is a monolithic 32-bit protected-mode driver that communicates with a specific disk device such as a hard disk controller. This driver is specifically for use with Windows 95 and resembles the 32-bit disk access (fast disk) driver used in Windows 3.1 (for example, WDCTRL for Western Digital compatible hard disk controllers). In Windows 95, the driver to communicate with IDE/ESDI hard disk controllers and floppy disk controllers is implemented as a port driver. A port driver provides the same functionality as the combination of the SCSI manager and the mini-port driver.

SCSI Layer

The SCSI layer applies a 32-bit protected-mode universal driver model architecture to communicating with SCSI devices. The SCSI layer provides all the high level functionality that is common to SCSI-like devices, and then uses a mini-port driver to handle device-specific I/O calls. The SCSI Manager is also part of this system and provides the compatibility support for using Windows NT mini-port drivers.

Mini-Port Driver

The mini-port driver model in Windows 95 simplifies the task for a hardware disk device vendor to write a device driver. Because the SCSI Stub provides the high level functionality for communicating with SCSI devices, the hardware disk device vendor only needs to create a mini-port driver that is tailored to his own disk device. The mini-port driver for Windows 95 is 32-bit protected-mode code, and is binary compatible with Windows NT mini-port drivers, minimizing the task required by a hardware vendor to write device drivers. Binary compatibility with NT also results in a more stable and reliable device driver as the hardware vendor needs to only maintain one code base for device support, and users of Windows 95 benefit from the preexistence of many mini-port drivers already available for Windows NT.

Support for IDE, SCSI, ESDI controllers

Through the use of either a port driver, or a mini-port driver, support for a broad array of disk devices will be available when Windows 95 ships including popular IDE, ESDI, and SCSI disk controllers. Keep in mind that users don't have to decide whether to use a port driver or a mini-port driver, the driver is provided by the hardware vendor and configuration of the driver is handled by the Windows 95 system.

Real-Mode Mapper (RMM)

To provide compatibility with real-mode MS-DOS device drivers for which a protected-mode counterpart does not exist, the block I/O subsystem provides a mapping layer to allow the protected-mode file system to communicate with a real-mode driver as if it was a protected-mode component. The layers above and including the real-mode mapper are protected-mode code, and the real-mode mapper translates file I/O requests from protected-mode to real-mode such that the MS-DOS device driver can perform the desired operation to write or read information to or from the disk device. An example scenario where the real-mode mapper would come into play is when real-mode disk compression software is running and a protected-mode disk compression driver is not available. The net effect of this component is to ensure binary compatibility with existing MS-DOS–based disk device drivers in Windows 95.

Long Filename Support

The use of long filenames in Windows 95 overcomes the sometimes cryptic 8.3 MS-DOS filename conventions, to allow more user friendly filenames. MS-DOS 8.3 filenames are still maintained and tracked by the system to support compatibility with existing Win16 and MS-DOS–based applications that only manipulate 8.3 filenames, but as users migrate to Win32–based applications the use of 8.3 filename conventions is hidden from the user. Long filenames can be up to 255 characters in length.

Long filenames are supported by extending the MS-DOS FAT file system and using bits and fields that were previously reserved by the operating system to add special directory entries that maintain long filename information. Extending the MS-DOS FAT layout, rather than creating a new format, allows users to install and use Windows 95 on existing disk formats without having to change their disk structure, or reformat their drives. This implementation provides future growth and ease of use, while still maintaining backward compatibility with existing applications.

Because Windows 95 simply extend the FAT structure, support for long filenames is support on disks as well as hard disk drives. If a long filename is used for a file on a disk and is viewed by a user on a computer not running Windows 95, the user would only see the 8.3 filename representations.

Figure 5 shows a disk directory on a computer running Windows 95, showing long filenames and the corresponding 8.3 filename mappings.

```
Volume in drive C is MY HARDDISK
 Volume Serial Number is 1B47-7161
 Directory of C:\LONGFILE

.            <DIR>        05-11-94 10:34a .
..           <DIR>        05-11-94 10:34a ..
4THQUART XLS          147  05-11-94 12:25a 4th Quarter Analysis.xls
TEXTFILE TXT          147  05-11-94 12:25a TEXTFILE.TXT
THISISMY DOC          147  05-11-94 12:25a this is my long filename.doc
1994FINA DOC          147  05-11-94 10:35a 1994 Financial Projections.doc
        4 file(s)          588 bytes
        2 dir(s)      48,009,216 bytes free
```

Figure 5. Directory with Long Filenames Visible from Command Prompt

Support for Existing Disk Management Utilities

In order for existing disk management utilities to recognize and preserve long filenames, utility vendors will need to revise their software offerings. Microsoft is working closely with utilities vendors and is documenting long filename support and its implementation as an extension to the FAT format as part of the Windows 95 Software Development Kit (SDK).

Existing MS-DOS–based disk management utilities that manipulate the FAT, including disk defragmenters, disk bit editors, and some tape backup software, may not recognize long filenames as used by Windows 95 and may destroy the long filename entries in the FAT. However, the corresponding system-defined 8.3 filename will be preserved so there is no loss of data if the long filename entry is destroyed.

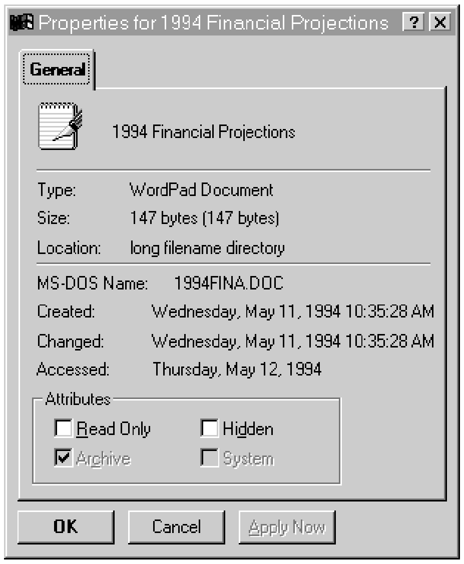
File Extensions Hidden From User

File extensions are used by Windows 95 to associate a given file type with an application as is handled under Windows 3.1. However, file extensions are hidden from users in the Shell and Windows Explorer to make it easier to manipulate files, and icons are used in the UI in Windows 95 to differentiate documents associate with applications. For compatibility reasons, it is still necessary for Windows 95 to track filename extensions for use with existing MS-DOS and Win16–based applications. Information on the file type associations is stored in the Registry, and the associations are used to map a given file with the appropriate icon representing the document type.

In addition to hiding filename extensions in the Windows 95 shell and Windows Explorer, mechanisms are available for application developers to hide filenames from users in their applications, and this is documented in the *Windows 95 SDK*. A good Windows 95 application will make use of these mechanisms for handling files to be consistent with the rest of the Windows 95 environment.

Additional File Date/Time Attributes

To further enhance the file system, Windows 95 maintains additional date/time attributes for files that MS-DOS does not track. Windows 95 will now maintain the date/time when a new file is created, the date/time when a file has been modified or changed, and the date when a file was last opened. These file attributes will be displayed when a user requests to display file properties as shown in Figure 6.


Properties for 1994 Financial Projections

General

1994 Financial Projections

Type: WordPad Document
Size: 147 bytes (147 bytes)
Location: long filename directory

MS-DOS Name: 1994FINA.DOC
Created: Wednesday, May 11, 1994 10:35:28 AM
Changed: Wednesday, May 11, 1994 10:35:28 AM
Accessed: Thursday, May 12, 1994

Attributes
☐ Read Only ☐ Hidden
☑ Archive ☐ System

Figure 6. Properties for a File, Showing New File Attributes

Utilities can take advantage of this additional time/date information to provide enhanced backup utilities, for example, to use a better mechanism when determining whether a given file has been changed or modified by the system.

Coordinated Universal Time (UTC) Format

MS-DOS has traditionally used the local time of the computer as the time stamp for the directory entry of a file. Windows 95 will continue to do this for files stored on the local system, however support for using the UTC time format for accessing or creating information on network file servers. This will provide better, more universal tracking of time information as required by networks that operate across time zones.

Exclusive Volume Access For File Recover Tools

Today, disk management utilities such as disk defragmenters, sector editors, and disk compression utilities, don't get along well with Windows 3.1. File system programs, such as CHKDSK and DEFRAG, require special (exclusive) access to the file system to minimize the disk access complexities that are present in a multi-tasking environment where disk I/O occurs. For example, if a user requests to do a disk operation that moves files or information around on the disk, if another task was accessing the information or writing information to disk at the same time, without exclusive access to the disk it would be possible that data corruption could occur. Windows 3.1 and MS-DOS do not provide a means of controlling access to the disk when other tasks may need to write information out at the same time, and it is for this reason that it is necessary today for users to exit Windows and enter MS-DOS to run disk management utilities.

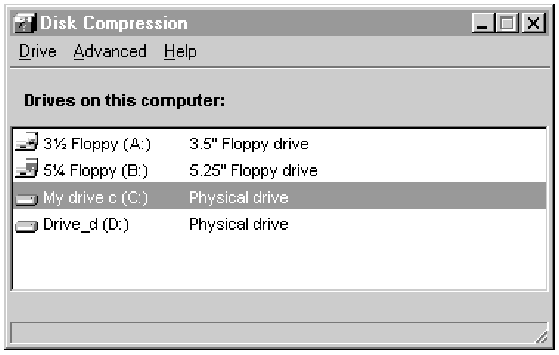
The file system in Windows 95 has been enhanced to permit exclusive access to a disk device to support the use of Windows–based disk management utilities. This is not an end-user feature, but rather is an end-user benefit. Exclusive disk access is handled through a new API mechanism as part of the file system and can be used by utilities' vendors to write Windows–based disk management utilities. Microsoft is evangelizing this API mechanism to third-party utility vendors to facilitate moving existing MS-DOS–based utilities to Windows, as well as is using it to deliver disk management utilities as part of Windows 95.

For example, this mechanism is being used by the Disk Defragmenter (Optimizer) utility delivered as part of Windows 95. Unlike the combination of MS-DOS and Windows 3.1, the disk defragment utility in Windows 95 can be run from the Windows 95 shell, *and* can even be run in the background while you continue to work on your system.

DriveSpace Disk Compression

Windows 95 provides built-in support for DriveSpace disk compression. Compatible with DoubleSpace and DriveSpace disk compression provided with MS-DOS, Windows 95 provides base compression in the form of a 32-bit virtual device driver that delivers improved performance over previously available real-mode compression drivers, and frees conventional memory for use by MS-DOS–based applications. Existing users of DoubleSpace and DriveSpace will not need to change the compressed volume file (CVF) that they are presently using, and thus will not need to take any special actions when they install Windows 95.

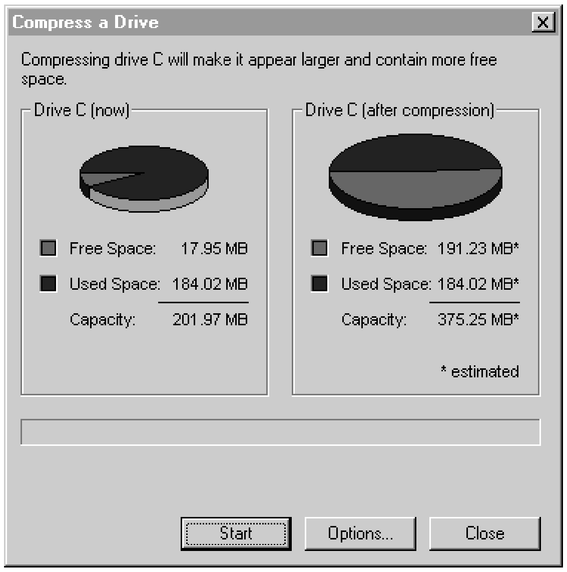
The DriveSpace disk compression tool provided with Windows 95 is GUI-based and provides the ability to compress a physical hard drive or a removable floppy drive.



3½ Floppy (A:)	3.5" Floppy drive
5¼ Floppy (B:)	5.25" Floppy drive
My drive c (C:)	Physical drive
Drive_d (D:)	Physical drive

Figure 7. DriveSpace Disk Compression Tool

The Compress a Drive dialog box (see Figure 8) graphically shows the result of the amount of free space available as the drive is now (before compression), and the estimated size and space available after compressing the drive.



Compressing drive C will make it appear larger and contain more free space.

Drive C (now)

▨	Free Space:	17.95 MB
■	Used Space:	184.02 MB
	Capacity:	201.97 MB

Drive C (after compression)

▨	Free Space:	191.23 MB*
■	Used Space:	184.02 MB*
	Capacity:	375.25 MB*

* estimated

Figure 8. Compress a Drive Graphical Free Space Display

Improved System Capacity

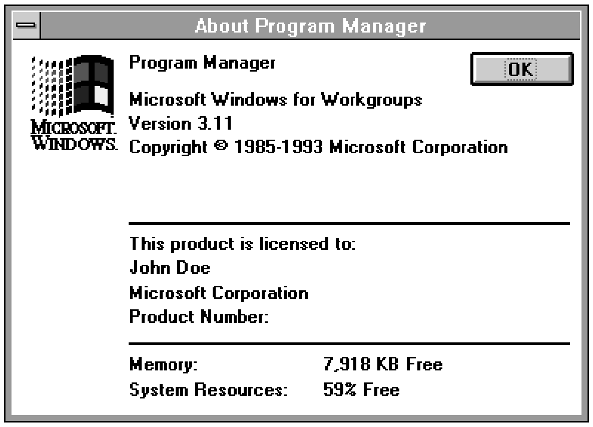
Windows 95 provides better system capacity for running MS-DOS-based and Win16-based applications than Windows 3.1. A number of internal enhancements have been made to the base system, allowing for internal system resources to not be exhausted as quickly as was possible under Windows 3.1 when running multiple Windows–based applications.

Many of the artificial limitations present in Windows 3.1 due to its architecture or internal data structures and largely due to the fact that Windows 3.1 had to run on an Intel 80286-based computer, have been greatly improved and overcome in Windows 95. This will please end-users, as well as ISVs and other developers.

System Resource Limitation Improved

Many users have probably seen “Out of Memory” error messages when running multiple Windows–based applications under Windows 3.1, even though the system still reports several megabytes of available free memory. What users typically encountered was a condition where the system was not able to allocate an internal memory resource in a Windows API function call due to not enough space available in a region of memory called a *heap*.

Windows 3.1 maintains heaps for system components called GDI and USER. Each of the heaps is 64K in size and is used for storing GDI or memory *object* information allocated when an application calls a Windows API function. The amount of space available in the combination of these two heaps is identified as a percentage of system resources that are free and is shown in the Help About box in Program Manager and other Windows applications as shown in Figure 9.



Program Manager

Microsoft Windows for Workgroups
Version 3.11
Copyright © 1985-1993 Microsoft Corporation

This product is licensed to:
John Doe
Microsoft Corporation
Product Number:

Memory: 7,918 KB Free
System Resources: 59% Free

Figure 9. About Box in Program Manager In Windows 3.1 Showing Free System Resources

The percentage of free system resources displayed in the About box is calculated using an internal algorithm to represent the aggregate percentage of free memory in the GDI and USER heaps. When the free system resources percentage drops to a low number, it is quite common that the user will see an “out of memory” error message, even though the amount of free memory shown in the About box is still quite high. This error can be due to low memory in either the GDI or the USER heap (or both).

To help reduce the system resource limitation, a number of the data structures stored in the 16-bit GDI and USER heaps in Windows 3.1 have been moved out of these heaps and stored in 32-bit heaps, providing more room for the remaining data elements to be created. Users will see improvements by not encountering a decrease in system resources as rapidly as they may have seen with Windows 3.1.

All objects were not simply removed from the 16-bit GDI or USER heaps, and placed in 32-bit heaps for compatibility reasons. For example, there are some Windows–based applications that manipulate the contents of the GDI heap directly, bypassing the published API mechanisms for doing so. These application vendors to do this for perceived performance reasons. However, because they bypass the Windows API mechanisms, moving the data from the existing heap structures and placing them in 32-bit heaps would cause the existing applications to fail due to memory access violations.

Both Win16 and Win32–based applications use the same GDI and USER heaps. The impact of removing selected items from the heaps was closely examined and objects were selected based on the biggest improvement that could be achieved, while affecting the fewest number of applications. For example, the GDI heap can quickly become full due to the creation of memory-intensive region objects that are used by applications for creating complex images and by the printing subsystem for generating complex output. Regions have been removed from the 64K 16-bit GDI heap and placed into a 32-bit heap, benefiting graphic-intensive applications and providing for the creation of more smaller objects by the system. Windows 95 improves the system capacity for the USER heap, by moving menu and window handles to the 32-bit USER heap, raising the total limit of these data structures from 200 in Windows 3.1, to a total limit now of 32,767 menu handles and an additional 32,767 window handles *per* process rather than system wide.

In addition to examining information present in the GDI and USER heaps, the robustness improvements present in Windows 95 that facilitate cleaning up the system of unfreed resources will also help the system resource limitation problem. Windows 95 will clean up and de-allocate left over data structures once Windows 95 determines that the owner and other ended processes no longer need the resources in memory. The robustness improvements available in Windows 95 are discussed in the next section.

Better Memory Management

Windows 95 improves addressibility for accessing physical memory in the computer, as well as improves upon the swapfile implementation provided in Windows 3.1 to support virtual memory to supplement physical system memory.

Linear Memory Addressing for Win32–based Applications

To support a 16-bit operating environment, the Intel processor architecture uses a mechanism called *segments* to reference memory by using a 16-bit segment address, and a 16-bit offset address within the segment. A segment is 64K in size, and applications and the operating system endure a performance penalty for accessing information across segments. Windows 95 addresses this issue by using the 32-bit capabilities of the Intel 80386 (and above) processor architecture to support a flat, linear memory model for 32-bit operating system functionality and Win32–based applications. A linear addressing model simplifies the development process for application vendors, removes the performance penalties imposed by the segmented memory architecture, and provides access to a virtual address space that enables addressing up to 4 gigabytes (GB) of memory. Windows 95 uses the flat memory model internally for 32-bit components and virtual device drivers.

Compatible with the Memory Model used by Windows NT

Windows 95 uses the same memory model architecture used by Windows NT, providing high-end operating system functionality on the mainstream desktop. Windows 95 will allow full use of the 4 gigabytes (4 billion bytes of memory) of addressable memory space to support even the largest desktop application. The operating system provides a 2 gigabyte memory range for applications, and reserves a 2 gigabyte range for itself.

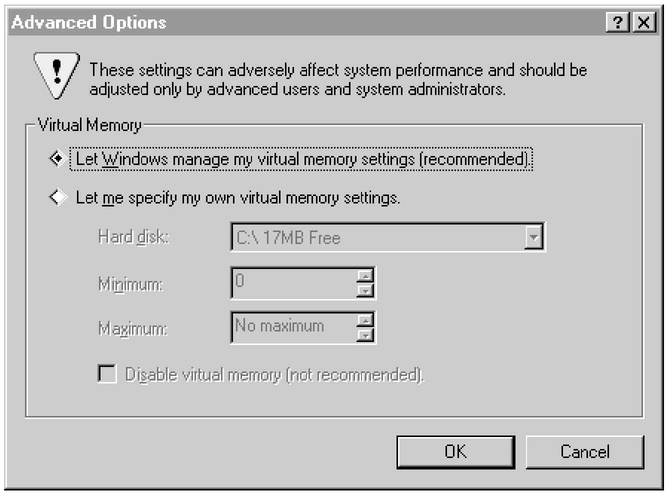
Improved Virtual Memory Support—Swapfile Improvements

Windows 95 improves on the virtual memory swapfile implementation provided in Windows 3.1 to address the problems and limitations imposed in Windows 3.1.

Under Windows 3.1, users were faced with a myriad of choices and configuration options when it came to setting up a swapfile to support virtual memory. They had to decide whether to use a temporary swapfile or a permanent swapfile, how much memory to allocate to the swapfile, and whether to use 32-bit disk access to access the swapfile or not. Users benefited from a temporary swapfile in that the swapfile did not need to be contiguous, and Windows would allocate space on the hard disk when Windows was started and free up the space when the user exited Windows. A permanent swapfile provided the best performance, however it required a contiguous block of space, had to be set up on a physical hard disk, and was statically specified by the user and not freed up when the user exited Windows.

The swapfile implementation in Windows 95 simplifies the configuration task for the user and combines the best of a temporary swapfile and a permanent swapfile due to improved virtual memory algorithms and access methods. The swapfile in Windows 95 is now dynamic, and can shrink or grow based on the operations that are performed on the system. The swapfile can also occupy a fragmented region of the hard disk, as well as can be located on a compressed disk volume.

Windows 95 uses intelligent system defaults for the configuration of virtual memory, thus preventing the user from needing to change virtual memory settings. Figure 10 shows the new simplified virtual memory configuration settings.


Advanced Options

These settings can adversely affect system performance and should be adjusted only by advanced users and system administrators.

Virtual Memory

[✓ Let Windows manage my virtual memory settings (recommended).]

Let me specify my own virtual memory settings.

Hard disk: C:\ 17MB Free

Minimum: 0

Maximum: No maximum

☐ Disable virtual memory (not recommended).

Figure 10. Virtual Memory Settings in Windows 95 are Simplified Over Windows 3.1

The Registry—Centralized Configuration Store

Windows 95 uses a mechanism called the *Registry* that serves as the central configuration store for user, application, and computer-specific information. The Registry solves problems associated with .INI files as used in Windows 3.1, and is a hierarchical database that stores system-wide information in a single location, making it easy to manage and support.

Problems with Windows 3.1 .INI Files

Windows 3.1 uses initialization (.INI) files to store system-specific or application-specific information on the state or configuration of the system. For example, the WIN.INI file is used to store state information about the appearance or customization of the Windows environment, the SYSTEM.INI file is used to store system-specific information on the hardware and device driver configuration of the system, and various .INI files are used to store application-specific information about the default state of an application (for example, WINFILE.INI, MSMAIL.INI, CLOCK.INI, CONTROL.INI, PROGMAN.INI, and so on).

Problems with .INI files under Windows 3.1 for configuration management include:

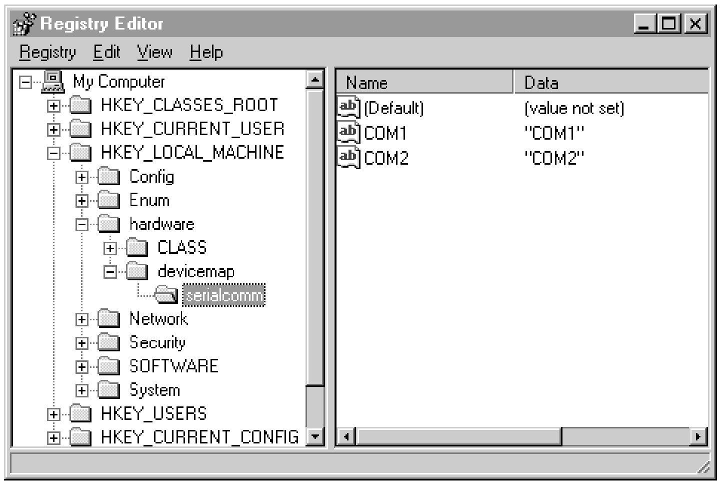
- Information is stored in several different locations including CONFIG.SYS, AUTOEXEC.BAT, WIN.INI, SYSTEM.INI, PROTOCOL.INI, private .INI files, and private .GRP files
- .INI files are text-based, are limited to 64K in total size, and APIs only allow for get/write operations
- information stored in .INI files is non-hierarchical and supports only two-levels of information (i.e., key names broken up by section heading)
- Many .INI files contain a myriad of switches and entries that are complicated to configure or are used only by operating system components
- .INI files provide no mechanism for storing user-specific information, thus making it difficult for multiple users to share a single computer
- Configuration information in .INI files is local to each system, and no API mechanisms are available for remotely managing configuration, thus making it difficult to manage multiple systems

Solution to Windows 3.1 .INI File Problems

To solve problems associated with .INI files under Windows 3.1, the Registry was designed with the following goals in mind:

- Simplify the support burden
- Centralize configuration information
- Provide a means to store user, application, and computer-specific information
- Provide local and remote access to configuration information

The Registry is structured as a hierarchical database of keys, where each key can contain a value, or can even contain other keys (subkeys). While similar in some ways to the Registration Database used in Windows 3.1, which served as a central repository for file associations and OLE registration information, the Registry in Windows 95 extends the previous structure to support keys that can have more than one value and can also support data of different types. The Registry uses a hierarchical structure to store text or binary value information to maintain all of the configuration parameters normally stored in the Windows system .INI files such as WIN.INI, SYSTEM.INI, and PROTOCOL.INI.



```
My Computer
  HKEY_CLASSES_ROOT
  HKEY_CURRENT_USER
  HKEY_LOCAL_MACHINE
    Config
    Enum
    hardware
      CLASS
      devicemap
        serialcomm
    Network
    Security
    SOFTWARE
    System
  HKEY_USERS
  HKEY_CURRENT_CONFIG
```

Name	Data
(Default)	(value not set)
COM1	"COM1"
COM2	"COM2"

Figure 11. Hierarchy of Registry as Displayed by the Registry Editor

The Registry is made up of several .DAT files that contain system-specific information (SYSTEM.DAT) or user-specific information (USER.DAT). System-specific information such as the static reference to loading virtual device drivers will be moved as appropriate from the SYSTEM.INI file to the Registry.

System Switch Simplification

Another improvement made over Windows 3.1 and its use of ..INI files is related to system switch simplification. Windows 3.1 supports over several hundred different configuration switches that can be specified in system .INI files including the WIN.INI or SYSTEM.INI files. With intelligent enhancements made to the system, and better dynamic configuration properties, Windows 95 has reduced the number of entries that are normally associated with .INI files. These reductions didn't come just by moving .INI entries to the Registry, but by examining and justifying the presence of each and every one.

.INI Files Still Exist for Compatibility Reasons

For compatibility reasons, WIN.INI and SYSTEM.INI and application-specific .INI files (as well as CONFIG.SYS and AUTOEXEC.BAT) do not go away. The Win16 APIs for manipulating .INI files will still manipulate .INI files, however Win32–based applications will be encouraged to use the Registry APIs to consolidate application-specific information.

Many existing Win16–based applications expect to find and manipulate the WIN.INI and SYSTEM.INI files to add entries or load unique device drivers, therefore SYSTEM.INI, for example, will still be examined during the boot process of Windows 95 to check for virtual device drivers in the **[386Enh]** section.

Role in Plug and Play

One of the primary roles of the Registry in Windows 95 is to serve as a central repository for hardware-specific information for use by the Plug and Play system components. Windows 95 maintains information about hardware components and devices that have been identified through an enumeration process in the hierarchical structure of the Registry. When new devices are installed, the system checks the existing configuration in the Registry to determine the hardware resources (for example, IRQs, I/O addresses, DMA channels, and so on) that are not being used, so the new device can be properly configured without conflicting with a device already installed in the system.

Remote Access to Registry Information

Another advantage of the Registry for Win32–based applications is that many of the Win32 Registry APIs are remoted using the remote procedure call (RPC) mechanism in Windows 95 to provide access to Registry information across a network. This allows desktop management applications to be written to aid in the management and support of Windows–based computers, and allows the contents of the Registry on a given PC to be queried and over a network. With this mechanism, industry management mechanisms such as SNMP or DMI can easily be integrated into Windows 95, simplifying the management and support burden of an MIS organization. See the Networking section later in this guide for more information on manageability and remote administration.

Better Font Support

Font support in Windows 95 has been enhanced to provide better integration with the user interface shell, optimized for the 32-bit environment, and provides capabilities such as font smoothing for fonts that has not been offered previously as part of a mainstream desktop operating system.

32-bit TrueType Rasterizer

The rasterizer component for rendering and generating TrueType fonts is enhanced in Windows 95. The rasterizer is written as a 32-bit component, and delivers better fidelity from the mathematical representation to the generated bitmap, as well as better performance for rendering TrueType fonts.

In addition to performance enhancements, the new 32-bit rasterizer also provides support for generating complicated glyphs (for example, Han), and results in a faster initial boot time when lots of fonts are installed in the system than Windows 3.1.