Windows NT Resource Guide

Microsoft® Window NT™

Preliminary Release

Microsoft Corporation

Printed in the United States of America.

Contents

Appendix

Introduction to This Preliminary Release Edition

Welcome to the *Windows NT™ Resource Guide*.

This preliminary release gives you an early snapshot of a manual that is designed for people who are, or who want to become, expert users of Microsoft® Windows NT. The *Windows NT Resource Guide* presents a detailed, easy-to-read technical view of Windows NT, so that you can better manage how Windows NT is used at your site. The *Windows NT Resource Guide* also contains specific information for system administrators who are responsible for installing, managing, and integrating Windows NT in a network or multi-user environment.

The *Windows NT Resource Guide* is a technical supplement to the documentation that is included in your Windows NT product and does not replace that information as the source for learning how to use Windows NT features and utilities.

This introductory section includes two kinds of information you can use to get started:

- The first section outlines the contents of the *Windows NT Resource Guide*, so you can quickly find technical details about specific elements of Microsoft Windows NT.
- The second section contains an overview of the conventions used to present information in the *Windows NT Resource Guide*.

About the Windows NT Resource Guide

This preliminary release includes key chapters that make up only part of the final version of the *Windows NT Resource Guide*. We wanted to give you an early glimpse of what will be released in a more few months.

This preliminary guide includes the following chapters.

- **Chapter 1, "Windows NT Architecture,"** describes the architecture used by Windows NT and discusses the components that make up its modular design.
- **Chapter 2, "Windows NT Networking Architecture,"** contains information targeted toward the support professional that may not have a local-area network (LAN) background. This chapter provides a technical discussion of networking concepts and discusses the networking components included with Windows NT.
- **Chapter 3, "Windows NT Security Model,"** describes in detail the security architecture for Windows NT. This security architecture is pervasive throughout the entire operating system, from logon security to access control for objects on the system.
- **Chapter 4, "Windows NT Configuration Registry,"** contains information about the Windows NT Configuration Registry that replaces all of the .INI files used in Windows 3.*x*.
- **Chapter 5, "New Technology File System,"** describes the new, fully secure file system for Windows NT.
- **Chapter 6, "Fault Tolerance Features of Windows NT,"** describes the mechanisms you can use with Windows NT to protect from data loss. This chapter includes a discussion of uninterruptable power supply (UPS), tape backup, disk mirroring, and disk duplexing. It also describes a technique available for Windows NT Advanced Servers called striping with parity.
- **Chapter 7, "Windows NT Installation and Configuration,"** contains a technical discussion of the Windows NT Setup program, details about setting up Windows NT on a network, and instructions for creating a custom installation routine for automated setup.
- **Chapter 8, "Windows NT Boot Sequence,"** explains what steps are taken to initialize the operating system. This chapter also identifies several entries in the Registry that affect the boot process.
- **Chapter 9, "Mail,"** includes information about the Mail application provided with Windows NT. This chapter describes the architecture of Mail and tips for customizing Mail.
- **Chapter 10, "Schedule+,"** includes information about the Schedule+ application provided with Windows NT. This chapter describes Schedule+ architecture and key features of interest to system administrators.

- **Chapter 11, "Troubleshooting Windows NT,"** provides specific information for troubleshooting problems with Windows NT, showing the key steps for isolating and solving common problems.

Conventions in This Manual

This document assumes that you have read the Windows NT documentation set and that you are familiar with using menus, dialog boxes, and other features of the Windows family of products. It also assumes that you have installed Windows NT on your system and that you are using a mouse with Windows NT. For keyboard equivalents to the actions described here, see the Microsoft Windows NT online Help.

This document uses several conventions to help you identify information.

Document Conventions

The following table describes the typographical conventions used in the *Windows NT Resource Guide*.

Type style	Used for
bold	MS-DOS®-style command names such as **copy** or **dir** and switches such as **/?** or **/3**, section.
italic	Parameter values for which you can supply specific values. For example, to supply a value for a parameter that calls for a *filename*, you must type a specific filename such as MYFILE.EXE.
ALL CAPITALS	Directory names, filenames, and acronyms. For example, "WINNT" is used to represent the Windows NT main directory.

Other conventions in this document include the following:

- "MS-DOS" refers to Microsoft MS-DOS version 3.3 or later.
- "Windows™-based application" is used as a shorthand term to refer to an application that is designed to run with 16-bit Windows and does not run without Windows. All 16-bit and 32-bit Windows applications follow similar conventions for the arrangement of menus, style of dialog boxes, and keyboard and mouse use.
- "MS-DOS-based application" is used in this document as a shorthand term to refer to an application that is designed to run with MS-DOS, but not specifically with Windows and is not able to take full advantage of Windows features (such as graphics or memory management).

- "Command prompt" refers to the command line where you type MS-DOS-style commands. Typically, you see characters such as "C:\>" to show the location of the command prompt on your screen. When Windows is running, you can double-click the MS-DOS Prompt icon in Program Manager to use the command prompt.
- An instruction to "type" any information means to press a key or a sequence of keys, and then press the ENTER key.
- Mouse instructions in this document, such as "Click the OK button" or "Drag an icon in File Manager," use the same meanings as the descriptions of mouse actions in the *Windows NT User's Guide* and the Windows online tutorial.

CHAPTER 1

Windows NT Architecture

At first glance, users may not recognizing that Windows NT is a completely new operating system. They will notice the familiar Windows 3.*x* interface and will appreciate the fact that they can continue to run their favorite applications for Windows 3.*x*, MS-DOS, and OS/2 1.*x* environments. But as with icebergs, this is just the tip—a host of powerful features lie beneath the surface of Windows NT.

Windows NT is a modern, 32-bit preemptive multitasking operating system with an architecture that is based on modular design principles. It is extensible and provides compatibility with many other operating systems, file systems, and networks. Windows NT also includes security and peer-to-peer networking services as fundamental components of the base operating system.

Additionally, Windows NT is portable across dissimilar processor architectures and runs on both complex instruction set computers (CISC™) and reduced instruction set computers (RISC). Windows NT also supports high performance computing by providing kernel support for computers that have symmetric multiprocessor configurations.

Windows NT only looks familiar. This chapter describes the powerful features under the graphical user interface.

Windows NT Design Goals

Windows NT was not designed as a reworked version of an earlier product. Its architects began with a clean sheet of paper and this list of goals for a new operating system:

- Compatibility
- Portability
- Scalability
- Security
- Distributed Processing

- Reliability and Robustness
- Internationalization
- Extensibility

In making this operating system *compatible*, the designers included the well-received Windows interface and provided support for existing file systems (including FAT and HPFS) and applications (including those written for MS-DOS, OS/2®, Windows 3.*x*, and POSIX). They also provided network connectivity to several existing networking environments.

Portability means that Windows NT runs on both complex instruction set computers (CISC) and reduced instruction set computers (RISC). CISC includes computers running with Intel® 80386 or 80486 processors. RISC includes computers with MIPS® R4000 or DEC Alpha processors.

Scalability means that Windows NT is not bound to single-processor architectures, but takes full advantage of symmetric multiprocessing hardware. Today, Windows NT can run on computers with from one to 16 processors. Windows NT allows you to expand to bigger and faster computers as your business requirements grow. And it gives you the advantage of having the same development environment for both your workstation and your server.

Windows NT includes a uniform *security* architecture that meets the requirements for a U.S. government rating. For the corporate environment, it provides a safe environment to run critical applications.

Distributed processing means that Windows NT is designed with networking built into the base operating system. Windows NT also allows for connectivity to a variety of host environments through its support of multiple transport protocols, and high level client/server facilities like named pipes and remote procedure calls (RPCs).

Reliability and robustness refer to an architecture which protects applications from damaging each other and the operating system. Windows NT employs the robustness of structured exception handling throughout its entire architecture. It includes a recoverable file system and provides protection through its built-in security and advanced memory management techniques.

Internationalization means that Windows NT will be offered in many countries around the world, in their own languages, and that it supports Unicode™.

Extensibility points to the modular design of Windows NT which, as described in the next section, provides for the flexibility of adding future modules at several levels within the operating system.

Windows NT Architectural Modules

As Figure 1.1 illustrates, Windows NT is a modular (rather than monolithic) operating system composed of several relatively simple modules. From the lowest level to the top of the architecture, the Windows NT modules are the Hardware Abstraction Layer (HAL), the Kernel, the Executive, the protected subsystems (included as part of the security model), and the environment subsystems.

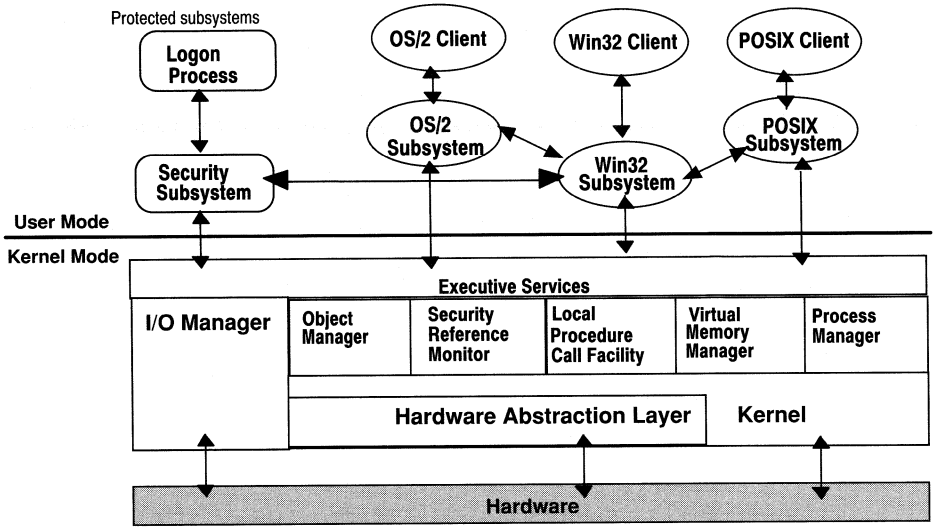


Figure 1.1 The modular architecture of Windows NT

For now, here is a brief explanation of what each layer does:

Hardware Abstraction Layer (HAL) virtualizes hardware interfaces, making the hardware dependencies transparent to the rest of the operating system. This helps in making Windows NT portable from one hardware platform to another.

Kernel is at the core of this layered architecture and manages the most basic operations of Windows NT. This component is designed to be small and efficient. The Kernel is responsible for thread dispatching, multiprocessor synchronization, hardware exception handling, and the implementation of low-level, machine-dependent functions.

Executive is the kernel-mode component that provides basic operating system services to the environment subsystems. It is comprised of several components that each manage a particular set of system services. One component, the Security Reference Monitor, works together with the protected subsystems to provide a pervasive security model for the system.

Environment subsystems are user-mode protected servers that run and support programs from different operating systems environments. Examples of these subsystems are the Win32™ subsystem and the OS/2 subsystem.

The remainder of this chapter describes in detail these architectural components of Windows NT, beginning with the Hardware Abstraction Layer, the layer seated between the computer's hardware and the rest of the operating system.

Hardware Abstraction Layer

The Hardware Abstraction Layer (HAL) is a thin layer of software provided by the hardware manufacturer that hides, or *abstracts*, hardware differences of from higher layers of the operating system. Thus, through the filter provided by HAL, different types of hardware all look alike to the operating system, removing the need to specifically tailor the operating system to the hardware with which it communicates.

The goal for HAL was to provide routines that allow a single device driver to support the same device on all platforms. HAL allows a large number of variations in hardware platforms for a single processor architecture without requiring a separate version of the operating system for each one.

HAL routines are called from both the base operating system and from device drivers. For drivers, HAL provides the ability to support a wide variety of I/O architectures, instead of either being restricted to a single hardware model or performing extensive adaptation, as in the current PC industry.

HAL is also responsible for hiding the details of symmetric multiprocessing hardware from the rest of the operating system.

Working very closely with the HAL is the Kernel. That component is described in the following section.

Kernel

The heart of Windows NT is the Kernel. It schedules and dispatches threads and synchronizes among multiple processors. It also provides hardware exception handling and implements low-level, machine-dependent functions.

Subcomponents at the Executive level, such as the I/O Manager and the Process Manager, use the Kernel to synchronize activities. They also rely on the Kernel for higher levels of abstraction (called kernel objects) which are exported within user-level application programming interface calls (APIs). For more information, see the "Kernel Objects" section later in this chapter.

Generally speaking, the Kernel does not implement any policy since this is the responsibility of the Executive. However, there are one place where policy decisions are made by the Kernel, namely when to remove processes from memory.

The primary functions provided by the Kernel are the following:

- Support of kernel objects
- Trap handling and exception dispatching
- Interrupt handling and dispatching
- Multiprocessor coordination and context switching
- Power failure recovery
- Miscellaneous hardware-specific functions

The Kernel runs entirely in kernel mode and is non-pageable. Software within the Kernel is not preemptable and therefore cannot be context switched, whereas all software outside the Kernel is almost always preemptable and context-switchable.

The Kernel can run simultaneously on all processors in a multiprocessor configuration and synchronizes access to critical regions as appropriate.

Thread Scheduling and Dispatching

The Kernel dispatches threads in a way that ensures that the system's processor is always as busy as possible running the highest priority code. Threads are dispatched for processing according to their software priority level. There are two classes of priority which together contain 32 levels of thread priority. The two priority classes are Real-Time priority and Variable priority.

The Kernel dynamically adjusts the priority of Variable threads, but does not alter the priority of Real-Time threads. However, as quantum-end events occur, the Real-Time threads within a level are scheduled in round-robin fashion. (A *quantum* is the processor time interval given to each thread in the scheduling algorithm.)

The priority of Variable threads is adjusted according to the resource use and demand, and the amount of time the thread has been waiting to access the processor. At each quantum-end event, the priority of the executing thread is decremented if it is above its base priority, and a decision is made as to whether it should be preempted by another thread. If it should be preempted to execute a higher priority thread, a context switch occurs. When a Variable thread makes the transition from Waiting to Ready state, it receives a priority boost based on the type of event that satisfied the Wait. For example, the thread would get a large boost if the event was keyboard input, but a smaller boost if the event was disk I/O.

When a thread is readied for execution, the Kernel tries to dispatch the thread to an idle processor. If there are multiple idle processors, preference is given to the last processor on which the thread executed. Giving such preference maximizes the chances that processor's secondary cache still contains thread data. If an idle processor is not available, the Kernel searches for a preemptable processor, again giving preference to the last processor on which the thread executed. Finally, if no processor can be preempted, the readied thread waits at the end of the queue for its priority level.

Kernel Objects

The Kernel exports a set of abstractions to the Executive which are called *kernel objects*. Kernel objects are used by the Executive to construct more complex objects that are exported in user level APIs. (For more discussion on objects, see the section called "Object Manager" later in this chapter.)

There are two types of kernel objects:

- Dispatcher objects
- Control objects

Dispatcher objects have a signal state (signaled or non-signaled) and control the dispatching and synchronization of system operations. Table 1.1 describes how the Executive uses each type of dispatcher object.

Table 1.1 Dispatcher Objects

Object type	Description
Event	Used to record the occurrence of an event and synchronize it with some action that is to be performed.
Mutant	One of two objects that the Kernel provides for controlling mutually-exclusive access to a resource. This type of object is intended for use in providing a user-mode mutual exclusion mechanism that has ownership semantics. It can also be used in kernel mode.
Mutex	The other of two objects that the Kernel provides for controlling mutually-exclusive access to a resource. This type of object can only be used in kernel mode and is intended to provide a deadlock-free mutual exclusion mechanism with ownership and other special system semantics.

Semaphore	Used to control access to a resource, but not necessarily in a mutually-exclusive fashion. A semaphore object acts as a gate through which a variable number of threads may pass concurrently, up to a specified limit. The gate is open (signaled state) as long as there are resources available. When the number of resources specified by the limit are concurrently in use, the gate is closed (non-signaled state).

Table 1.1 Dispatcher Objects *(continued)*

Object type	Description
Thread	The agent that runs program code and is dispatched to be run by the Kernel. Each thread is associated with a process object which specifies the virtual address space mapping for the thread and accumulates thread run time. Several thread objects can be associated with a single process object which enables the concurrent execution of multiple threads in a single address space (possibly simultaneous execution in a multiprocessor system).
Timer	Used to record the passage of time.

Control objects are used to control the operation of the Kernel but do not affect dispatching or synchronization. Table 1.2 describes how the Executive uses each type of control object.

Table 1.2 Control Objects

Object type	Description
Asynchronous Procedure Call	Used to break into the execution of a specified thread and cause a procedure to be called in a specified processor mode.
Interrupt	Used to connect an interrupt source to an interrupt service routine via an entry in an Interrupt Dispatch Table (IDT). Each processor has an IDT that is used to dispatch interrupts which occur on that processor.
Power notify	Used to automatically have a specified function called when power is restored after a power failure.
Power status	Used to check whether the power has already failed.
Process	Used to represent the virtual address space and control information necessary for the execution of a set of thread objects. A process object contains a pointer to an address map, a list of ready threads containing thread objects while the process is not in the balance set, a list of threads that belong to the process, the total accumulated time for all threads executing within the process, a base priority, and a default thread affinity. A process object must be initialized before any thread objects that specify the process as their parent can be initialized.

Profile	Used to measure the distribution of run time within a block of code. Both user and system code may be profiled.

The next several pages describe the functions of the Executive and its components.

Windows NT Executive

The Executive manages the interface between the Kernel and the environment subsystems by providing a set of common services that each environment subsystem can use. Each group of services is managed by a separate component of the Executive. The Executive components are the following:

- Object Manager
- Virtual Memory Manager
- Process Manager
- Local Procedure Call Facility
- I/O Manager
- Security Reference Monitor which, with the Logon and Security protected subsystems, makes up the Windows NT security model

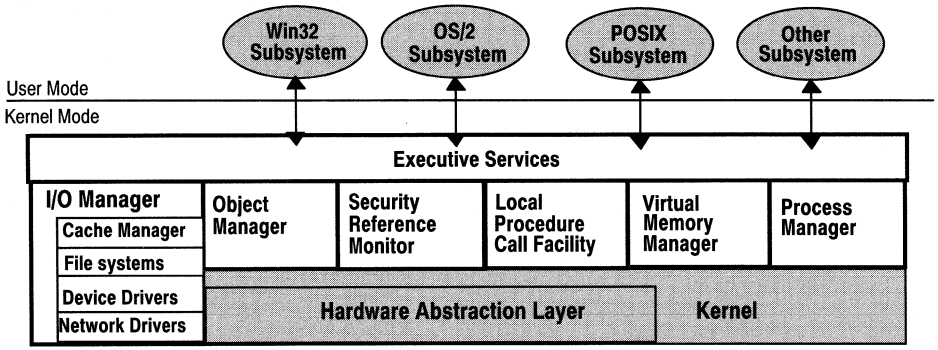


Figure 1.2 Windows NT Executive and its components

The Executive services shown in Figure 1.2 route messages from one Windows NT component to another. The role of each Executive component is described in following sections.

Object Manager

Object Manager is the part of the Windows NT Executive that provides uniform rules for retention, naming, and security of objects. An *object* is a runtime instance of a statically-defined object type and can be manipulated by an operating system process. An *object type* includes a system-defined data type, a list of operations that can be performed upon it (such as wait, create, or cancel), and a set of object attributes.

Before a process can manipulate a Windows NT object, it must first acquire a handle to the object. An *object handle* includes access control information and a pointer to the object itself. As far as the Executive is concerned, there is no differences between a file handle and an object handle. Thus, the same routines that are used to create a file handle can be used to create an object handle. Because all creation of object handles originates from the Object Manager, the Object Manager can satisfy some important Windows NT design requirements by providing the following:

- A uniform, global name space for all objects. The Object Manager can track creation and the use of objects by any process.
- Uniform rules and mechanisms for protecting objects from unauthorized access.
- A uniform model for safe sharing of objects.

Like other Windows NT components, Object Manager is extensible so that new object types can be defined as technology grows and changes.

The Object Manager manages the global name space for Windows NT. This name space is used to access all named objects that are contained in the local machine environment. Some of the objects that can have names include the following:

- Directory objects
- Symbolic link objects
- Process and thread objects
- Port objects
- File system objects
- Object type objects
- Semaphore and event objects
- Section and segment objects
- Device objects
- File objects

The object name space is modeled after a hierarchical file system, where directory names in a path are separated by a backslash (\). For example, after the Object Manager's system initialization, the object name space looks like this:

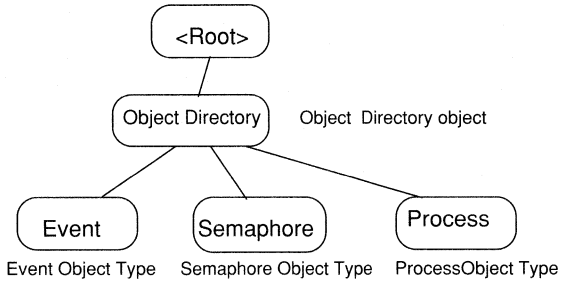


Figure 1.3 Object name hierarchy

A name lookup occurs whenever a new object is being inserted or an existing object is being opened by name. The name lookup is accomplished by searching in the root directory for the first name in the path.

Virtual Memory Manager

The memory architecture for Windows NT is a demand-paged virtual memory system. It is based on a flat, linear address space accessed via 32-bit addresses.

Virtual memory refers to the fact that the operating can actually allocate more memory than the computer physically has. Each process is allocated a unique virtual address space, which is a set of addresses available for the process's threads to use. This virtual address space is divided into equal blocks, or *pages*. Every process is allocated its own virtual address space which appears to be 4 gigabytes (GB) in size—2GB reserved for program storage and 2GB reserved for system storage.

Demand paging refers to a method by which data is moved in pages from physical memory to a temporary paging file on disk. As the data is needed by a process, it is paged back into physical memory.

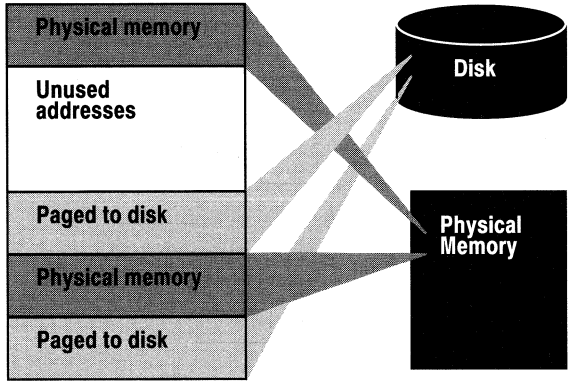


Figure 1.4 Conceptual view of virtual memory

The Virtual Memory Manager maps virtual addresses in the process's address space to physical pages in the computer's memory. In doing so, it hides the physical organization of memory from the process's threads. This ensures that the threads can access its process's memory as needed, but not the memory of other processes. Therefore, as illustrated by Figure 1.5, a thread's view of its process's virtual memory is much simpler than the real arrangement of pages in physical memory.

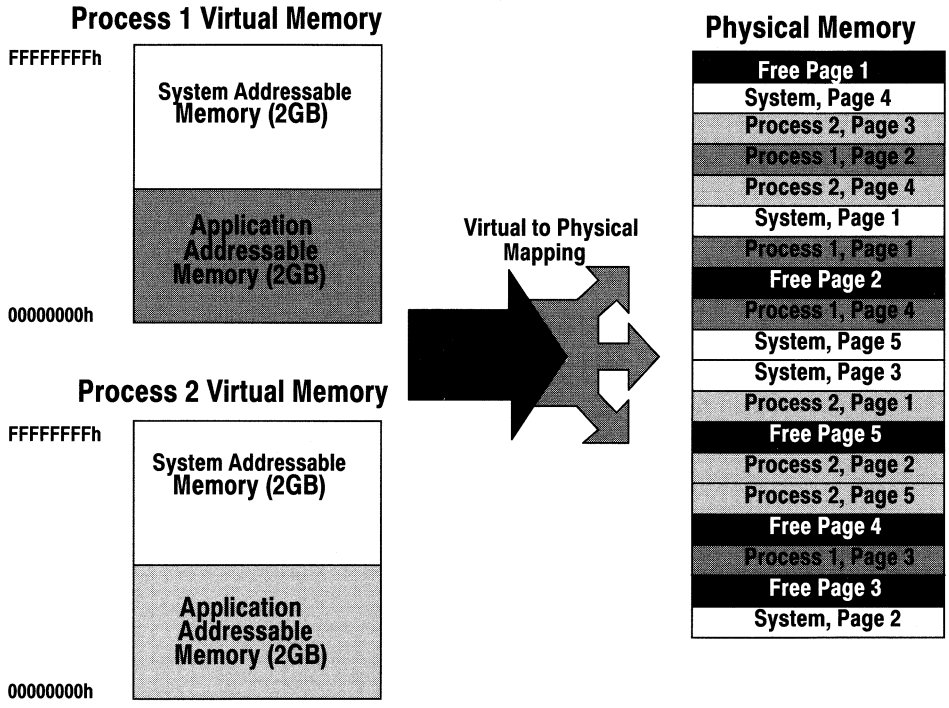


Figure 1.5 Protecting processes' memory

Because each process has a separate address space, a thread in one process cannot view or modify the memory of another process without authorization.

Process Manager

Process Manager is the Windows NT component that manages the creation and deletion of processes. Process Manager does not provide any hierarchical process structure, grouping, or enforce any parent/child relationships.

The Windows NT process structure includes only two types of objects—process objects and thread objects. A *process object* represents an address space, a set of objects (resources) visible to the process, and a set of threads that runs in the context of the process. A *thread object* represents the basic schedulable entity in the system. It contains its own set of machine registers, its own kernel stack, a thread environment block, and user stack in the address space of its process.

Process Manager provides a standard set of services for creating and using threads and processes in the context of a particular operating system environment. Beyond that, the Process Manager does little to dictate rules about threads and processes. Instead, the Windows NT design allows for robust environment subsystems that can define specific rules about threads and processes.

The Windows NT process structure works in conjunction with the security model and the Virtual Memory Manager to provide to provide interprocess protection. Each process is assigned a security access token, called the primary token of the process. The token is used by Windows NT's access validation routines when threads in the process reference protected objects. For more information about how Windows NT uses access tokens, see Chapter 3, "Windows NT Security Model."

Local Procedure Call Facility

Applications and environment subsystem have a client/server relationship. That is, the applications (clients) make calls to the environment subsystems (servers) to satisfy a request for some type of system services. To allow for a client/server relationship between applications and environment subsystems, Windows NT provides a communication mechanism between them. The Executive implements a message-passing facility called a Local Procedure Call (LPC) Facility. It works very much like the Remote Procedure Call (RPC) Facility used for networked processing (described in Chapter 2, "Windows NT Networking Architecture"). However, LPC Facility is optimized for two processes running on the same computer.

Applications communicate with environment subsystems by passing messages via the LPC Facility. The message-passing process is hidden from the client applications by function *stubs* provided in special dynamic link libraries (DLL), as illustrated by Figure 1.6.

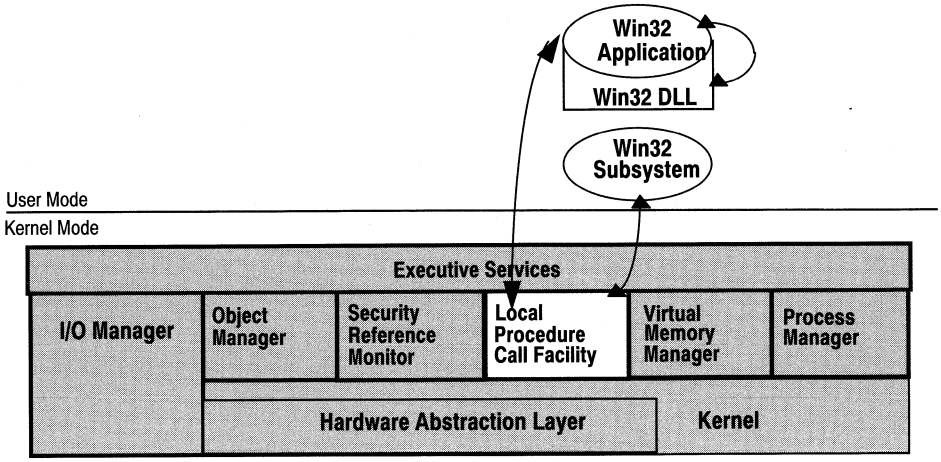


Figure 1.6 Interaction with IPC Manager

When an application makes an API call to an environment subsystem, the stub in the client (application) process packages the parameters for the call and sends them to a server (subsystem) process that implements the call. It is the LPC Facility which allows the stub procedure to pass the data to the server process and wait for a response.

For example, consider how this process works in the Win32 subsystem. When a Win32 application is loaded to run, it is linked to a dynamic link library that contains stubs for all of the functions in Win32 application programming interface. When the application calls a Win32 function, the call is processed as follows (where "CreateWindow()" is a sample Win32 function):

1. The CreateWindow() stub function in the dynamic link library (DLL) is called.
2. The stub function constructs a message that contains all of the data needed to create a window and sends the message to the Win32 server process.
3. The Win32 server receives the message and calls the *real* CreateWindow() function. The window is created.
4. The Win32 server sends a message containing the results of the CreateWindow() function back the stub function in the DLL.
5. The stub function unpacks the server's message and returns the results to the Win32 application.

From the application's perspective, the CreateWindow() function in the DLL created the window. The application does not know that the work was actually performed by the Win32 server process, that a message was sent to make it happen, or even that the Win32 server process exists.

I/O Manager

I/O Manager is the part of the Windows NT Executive that manages all input and output for the operating system. A large part of I/O Manager's role is to manage communications between drivers. I/O Manager supports all file system drivers, hardware device drivers, and network device drivers and provides a heterogeneous environment for them. It provides a formal interface that all drivers can call. This uniform interface allows I/O Manager to communicate with all drivers in the same way, without any knowledge of how the devices they control actually work. I/O Manager also includes device driver helper routines specifically designed for file system drivers, for hardware device drivers, and for network device drivers. These helper routines are available for use by developers to reduce the amount of work that is required to write a device driver.

The Windows NT I/O model uses a layered architecture which allows separate drivers to implement each logically distinct layer of I/O processing. For example, drivers in the lowest layer manipulate the computer's physical devices (these are called device drivers). Other drivers are then layered on top of the device drivers, as in Figure 1.7. These higher-level drivers do not know any details about the physical devices. With the help of the I/O Manager, higher-level drivers simply pass logical I/O requests down to the device drivers, which access the physical devices on their behalf. Windows NT's installable file systems and network redirectors are examples of high-level drivers that work in this way.

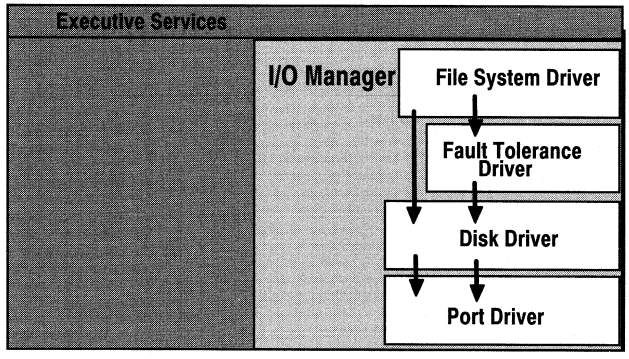


Figure 1.7 Layered device drivers

Layered drivers have several advantages over their monolithic counterparts. In particular, they are easier to develop and maintain. Common code can be implemented once and shared by many drivers. This makes layered drivers generally smaller and simpler to write than their monolithic counterparts. Also, changes in a driver in one layer rarely affect drivers in other layers because each layer represents a different level of I/O abstraction. Layered drivers are also more flexible since a new layered drivers can easily be inserted between two existing drivers to change the I/O system's behavior. (For example, in Figure 1.7, the

fault-tolerance driver is added between the file system driver and the disk driver to add functionality.) This is not possible with monolithic drivers because the entire driver would need to be replaced to change its behavior.

Note that this scheme allows easy replacement of file system drivers and device drivers. It even allows multiple file systems and devices to be active at the same time while being addressed through a formal interface.

Drivers communicate with each other using data structures called *I/O request packets*. The drivers pass I/O request packets to each other via the I/O Manager, which delivers the packets to the appropriate destination drivers using the drivers' standard services. Drivers can exchange I/O request packets synchronously or asynchronously. The simplest way to perform I/O is to synchronize the execution of applications with completion of the I/O operations that they request. (This is known as synchronous I/O.) When an application performs an I/O operation, the application's processing is blocked. When the I/O operation is complete, the application is allowed to continue processing.

Drivers can use one of three kernel objects to synchronous I/O. The first two—mutant (for user-mode operations) and mutex (for kernel-mode operations)—allow for mutually-exclusive access to the resource. Drivers can also use a semaphore which acts as a gate to the resource, limiting when and how many threads may have access at a time. (For more information, see the section called "Kernel Objects" earlier in this chapter.)

One way that applications can optimize their performance is to perform asynchronous I/O, a method employed by many of the processes in Windows NT. When an application initiates an I/O operation, the I/O Manager accepts the request but doesn't block the application's execution while the I/O operation is being performed. Instead, the application is allowed to continue doing work. Most I/O devices are very slow in comparison to a computer's processor, so an application can do a lot of work while waiting for an I/O operation to complete.

When an environment subsystem issues an asynchronous I/O request, the I/O Manager returns to the environment subsystem immediately after putting the request in a queue, without waiting for the device driver to complete its operations. Meanwhile, a separate thread from the I/O Manager runs requests from the queue in the most efficient order (not necessarily the order received).

When each I/O request is finished, the I/O Manager notifies the process that requested the I/O.

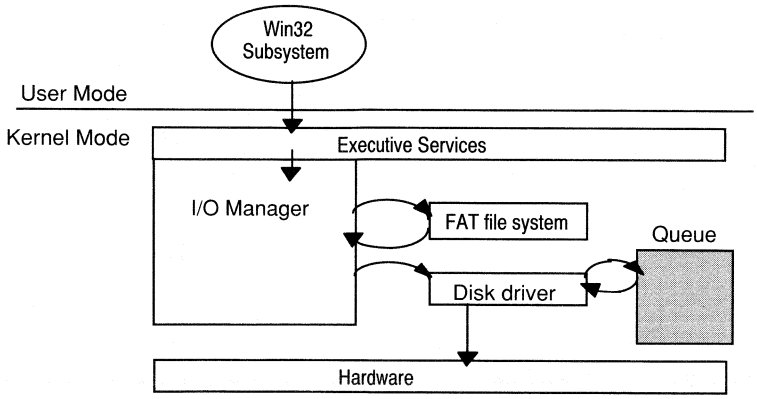


Figure 1.8 Queued I/O requests

While asynchronous I/O permits an application to use the computer's processor during I/O operations, it also makes it harder for the application to determine when I/O operations have completed. Some applications provide a callback function that is called when the asynchronous I/O operation is completed. Other applications use synchronization objects that the I/O system sets to the signaled state when the I/O operation is complete.

Cache Manager

The I/O architecture includes a single Cache Manager that handles caching for the entire I/O system. The Cache Manager uses a file mapping model, which is closely integrated with the Windows NT virtual memory management. Cache Manager provides caching services to all file systems and network components under the control of the I/O Manager. Cache Manager can dynamically grow and shrink the size of the cache as the amount of available RAM varies. When a process opens a file that is already resides in cache, the Cache Manager simply copies the from cache to the process's virtual address space, and vice versa, as reads and writes are performed.

Cache Manager offers services such as lazy write and lazy commit which can improve overall file system performance. *Lazy write* is the ability to record changes in the file structure cache, which is quicker than recording them on disk, and then later, when demand on the computer's CPU is low, the Cache Manager writes the changes to the disk. *Lazy commit* is similar to lazy write. Instead of immediately marking a transaction as successfully completed, the committed information is cached and later written to the file system log as a background process.

For more information about how Cache Manager works with the NTFS file system, see Chapter 5, "New Technology File System."

File System Drivers

In the Windows NT I/O architecture, file system drivers are components of the I/O Manager. Windows NT supports multiple active file systems, including existing file systems such as FAT and HPFS. Windows NT supports FAT and HPFS file systems for backward compatibility with MS-DOS, Windows 3.*x*, and OS/2 operating systems.

Windows NT also supports the New Technology File System (NTFS) which is a new file system designed for use with Windows NT. NTFS offers the simple design features of FAT and speed of HPFS. It also many other features including file system security, Unicode support, recoverability, long filename support, and support for POSIX. (For more information about FAT, HPFS, and NTFS, see Chapter 5, "New Technology File System.")

Windows NT's I/O architecture not only supports traditional file systems but has implemented its network redirector and server as file system drives. From the I/O Manager's perspective, there is no difference between accessing files stored on a remote networked computer and accessing those stored locally on a hard disk. This is an efficient model because it allows applications to use a single set of API calls (called Windows NT I/O functions) to access files on local and remote computers. The LAN Manager redirector and server can be loaded and unloaded dynamically, just like any other driver, and can coexist with other redirectors and servers.

Hardware Device Drivers

Hardware device drivers are also components of the I/O architecture. All hardware device drivers (such as printer drivers, mouse drivers, and disk drivers) are written in the C programming language, are 32-bit addressable, are multiprocessor aware. Operating system code that directly accesses the hardware registers of the peripheral devices is isolated in device drivers. Conveniently, device drivers are written for Windows NT so they are portable across different processor types.

The I/O Manager's design includes a "simple elegance" which insists upon clear separation between device drivers. For example, the i8042 is an interface device through which the keyboard and mouse communicate. The I/O Manager requires three separate drivers—i8042, keyboard, and mouse—rather than one large monolithic driver. This allows more flexibility to customize device configurations on the computer and to layer device drivers and other drivers.

Network Device Drivers

A third type of driver implemented as a component to the I/O architecture are network device drivers. As described in Chapter 2, "Windows NT Networking Architecture," Windows NT includes integrated networking capabilities and support for distributed applications. As illustrated by Figure 1.9, networking is supported by a series of network drivers.

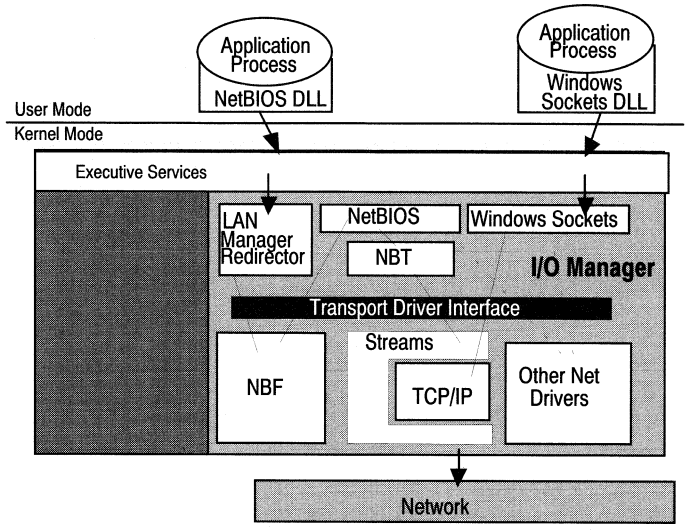


Figure 1.9 Networking components in Windows NT

Redirectors are implemented as file system drivers, as mentioned before, and run at or below a provider interface layer where NetBIOS and Windows Sockets reside.

Below the redirector layer is the Transport Driver Interface (TDI) and below TDI are transport protocols. Windows NT ships with a number of transports, including NBF, TCP/IP, and DLC. NBF (a descendant of NetBEUI) provides compatibility with existing LAN Manager, LAN Server, and MS®-Net installations. DLC provides an interface for access to mainframes and network attached printers. TCP/IP provides a popular routable protocol for wide-area networks.

At the bottom of the networking architecture is the network adapter card device driver. Windows NT currently supports device drivers written to the NDIS (Network Device Interface Specification) version 3.0. NDIS allows for a flexible environment of data exchange between transport protocols and network adapters. NDIS 3.0 allows a single computer to have several network adapter cards installed in it. In turn, each network adapter card can support multiple transport protocols for access to multiple types of network servers.

One final component of the Executive remains to be discussed—the Security Reference Monitor. This component, along with the Logon Process and Security protected subsystems, make up the Windows NT security model. This model is described in the next section.

Windows NT Security Model

In a multitasking operating system such as Windows NT, applications share a variety of system resources including the computer's memory, I/O devices, files, and system processor(s). Windows NT includes a security model (shown in Figure 1.10) which ensures that applications cannot access these resources without authorization.

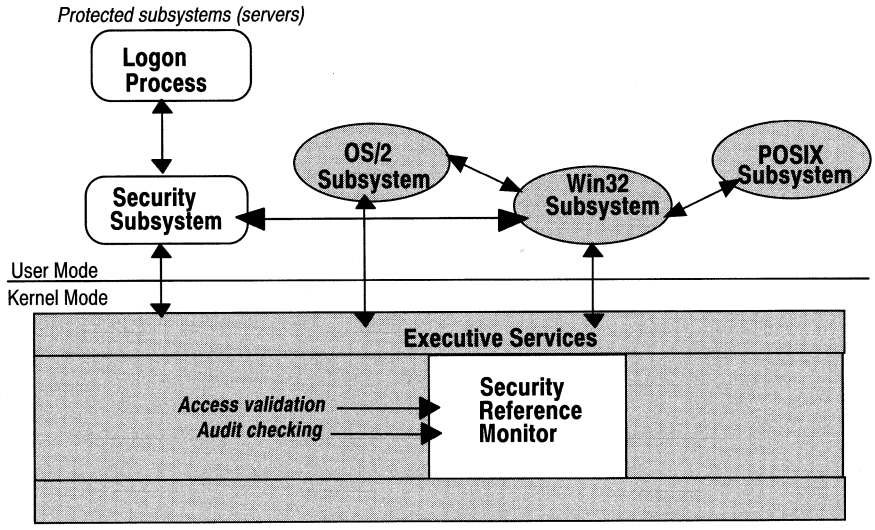


Figure 1.10 Windows NT security components

The Security Reference Monitor is responsible for enforcing the access validation and audit generation policy defined by the local security authority subsystem. The Reference Monitor provides services to both kernel and user mode for validating access to objects, testing subjects for privileges, and generating the necessary audit messages. The Reference Monitor, like other parts of the Executive, runs in kernel mode.

The user-mode Logon Process and Security protected subsystems are the other two components of the Windows NT security model. The Security subsystem is known as a *integral subsystem* rather than an environment subsystem because it affects the entire Windows NT operating system. (Environment subsystems are discussed later in this chapter.)

The Windows NT Kernel and Executive are based on a object-oriented paradigm. One of the reasons why Windows NT included an object model was to provide a consistent and uniform view of security, right down to the fundamental entities that make up the base operating system. This means that all of Windows NT's protected objects use the same routines for access validation and audit checks. That is, whether someone is trying to access a file on the disk or a process in memory, there is one component in the system that is required to perform access checks, regardless of the type of object.

Windows NT is designed to meet very stringent security requirements. For the first release of Windows NT, Microsoft is in the process of having Windows NT evaluated for the U.S. Department of Defense for a C2 rating.

The U.S. government has some very strict guidelines for U.S. government procurement of computer systems. One of those guidelines involves security of computer systems, both hardware and software. The guidelines has several divisions of security classification, ranging from A through D, where A is the most stringent. Divisions B and C are each further subdivided into classes. Class C2, the most stringent class within division C, provides for the following:

- Discretionary access control. This means monitoring access to protected objects based on the identity of a user, or group of users, or both.
- Object reuse protection. For example, if someone has deleted or shrunk a file, then no other user is allowed access any of the data that was in the file.
- Mandatory logon. All users must be identified and authenticated before they can access to the system.
- Auditing. Once a user is authenticated, then any security-related event (for example, access to a protected file) can be audited.

A C2 rating is important for the government sector, and also offers features important to the commercial sector. A great effort has gone into the design of the Windows NT security features to be as unobtrusive as possible and to prevent security from adversely affecting productivity.

The Windows NT Logon Process provides for mandatory logon to identify users. Every user has to have an account and give their password to access their account. Figure 1.11 illustrates the interaction between components of the security model during the processing of an interactive (logon at the attached keyboard and screen) logon request.

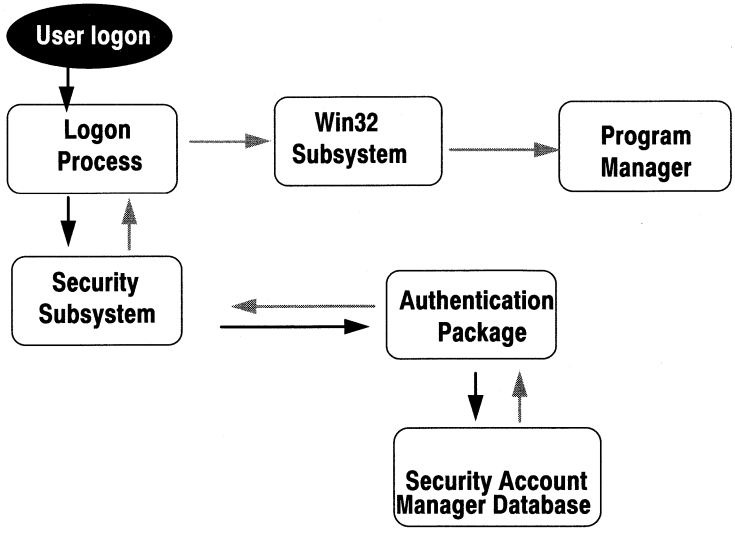


Figure 1.11 Logon security process

Logon authentication and validation works this way: Before a user can access any resource on a Windows NT computer, he or she must logon through a logon process so that the Security Subsystem can authenticate the user name and password for that person. After successful authentication, the user's shell process is tagged with a security access token. Following that, whenever the user tries to access a protected object, the Security Reference Monitor runs an access-validation routine against the user's security access token. Any other process that the user creates is also tagged with the user's access token, and is subject to the same validation routines.

Resource protection is another feature provided by the security model. This means that tasks can't access each others' resources, such as memory, except through specific sharing mechanisms. This feature helps enforce object hiding.

The security model also provides for discretionary access control so that the owner of a resource can specify which users or groups can access resources and what types of access they're allowed (such as read, write, and delete).

Windows NT also provides for auditing so administrators can keep an audit trail of what users perform what actions.

By providing these features, the Windows NT security model prevents applications from gaining unauthorized access to the resources of other applications or the operating system either intentionally or unintentionally.

For a complete description of how the security model works, see Chapter 3, "Windows NT Security Model."

In addition to the protected subsystems—Logon Process and Security—Windows NT includes a number of other user-mode components called environment subsystems. The next section describes each of the Windows NT environment subsystems.

Environment Subsystems

Windows NT was designed to allow many different types of applications to run seamlessly on the same graphical desktop. It runs applications written for existing operating systems such as MS-DOS and OS/2 1.*x*, and Windows 3.*x*. It also runs applications written for newer application programming interfaces such as POSIX and Win32.

Windows NT supports a variety of applications through the use of environment subsystems. *Environment subsystems* are Windows NT processes that emulate different operating system environments.

In this chapter, we've discussed how the Windows NT Executive provides generic services that all environment subsystems can call to perform basic operating system functions. The subsystems build on the Executive's services to produce environments that meet the specific needs of their client applications. Implementing common operating system functions once (in the Windows NT Executive) and separating them from the environment-specific features (in the environment subsystems) reduces the effort required to develop new environment subsystems and makes it easier to maintain them. Figure 1.12 shows a simplified view of the Windows NT environmental subsystem design.

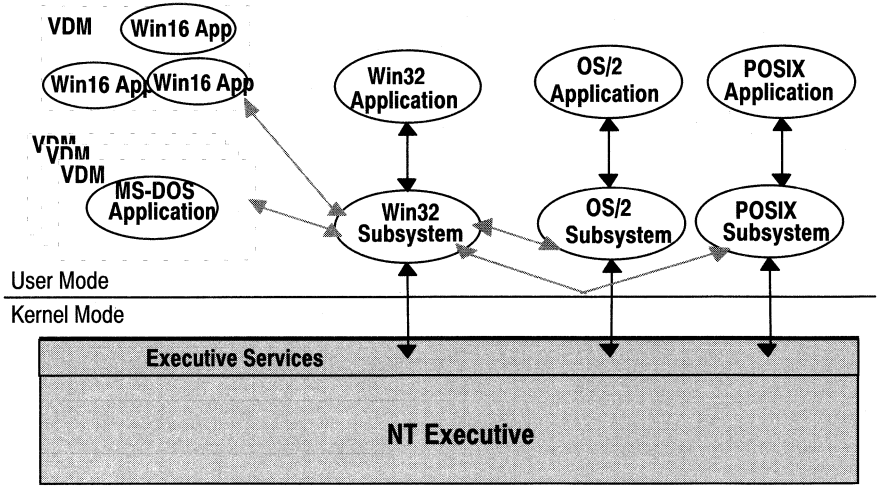


Figure 1.12 Conceptual view of NT environment subsystems

As implied by Figure 1.12, each subsystem runs as a separate user-mode process. Failure in one won't cause another subsystem or the Executive to be disabled. Each subsystem is protected from errors in the others. (An exception to this is if the Win32 subsystem crashes, since it handles keyboard and mouse input and screen output for all subsystems.) Applications are also user-mode processes, so they can't hinder the subsystems or Executive either.

Windows NT provides these protected environments subsystems and Multiple Virtual DOS Machines (MVDMs):

- MS-DOS VDM
- Win16 VDMs
- OS/2 subsystem
- POSIX subsystem
- Win32 subsystem

With the exception of the Win32 subsystem, each of these environments is optional and loaded only when their services are needed by a client application. The next few sections describe each of these environments.

MS-DOS Environment

On Windows NT, MS-DOS applications run within the context of a process called a Virtual DOS Machine (VDM). A *VDM* is a Win32 application that establishes a complete virtual *x*86 (that is, 80386 or 80486) computer running MS-DOS. There is no limit on the number of VDMs that can be run. Each VDM runs in its own address space which protects the applications from each other and the rest of the operating system from the VDMs.

When Windows NT is running on an *x*86 processor, a processor mode called Virtual-86 mode is available. This mode allows direct execution of most instructions in a MS-DOS-based application. A few instructions, such as I/O instructions, must be emulated in order to virtualize the hardware. When Windows NT is running on a RISC processor, hardware support for executing *x*86 instructions is not available. In such an environment it is necessary to emulate all of the *x*86 instructions in addition to providing a virtual hardware environment

To run MS-DOS-based applications, a VDM must provide a virtual PC. This involves providing virtual hardware for devices such as the screen and keyboard, support for ROM BIOS interrupt services, support for MS-DOS Interrupt 21 services, and support for processing *x*86 instructions. Hardware virtualization is provided by Virtual Device Drivers (VDDs). MS-DOS and ROM BIOS support is provided by the MS-DOS emulation module, and instruction execution support is provided by the Instruction Execution Unit (IEU). Figure 1.13 illustrates the structure of a VDM.

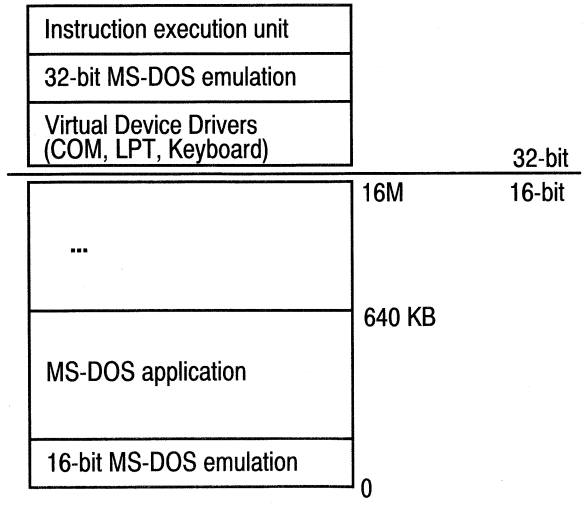


Figure 1.13 Structure of an MS-DOS VDM

On *x*86-based computers, character-based applications can run either in a window or in full-screen. Graphical applications can run only in full screen. If an application is in a window and then changes the video mode, it is automatically switched to full screen. On RISC-based computer, character-based and graphical applications run only in a window.

In short, this environment provides MS-DOS 5.0 application support, virtual device drivers that run in user-mode, full-screen and windowed text, and over 625K free memory. It also offers XMS (MS-DOS 5.0 HIMEM) support and EMM LIM 4.0 support. The MS-DOS VDM is completely pageable and provides transparent access to Windows NT file systems and network redirected drives. It supports DPMI 0.9 (just as Windows 3.1 does), multiple MS-DOS sessions, and Copy and Paste operations to the Clipboard.

Windows 16-bit Environment

Windows NT uses a single multi-threaded VDM support to run 16-bit Windows-based (Win16) applications. One of the mains goals for Win16 support is to provide a seamless interface for running Win16 applications in the Windows NT environment.

The Win16 VDM (sometimes called "WOW" for Win16 on Win32) is preemptively multitasked with respect to other processes running on the system. However, each Win16 application is non-preemptively multitasked with respect each other. That is, only one Win16 application at a time can run while the others are blocked. If the Win16 VDM is preempted when the system returns, it always unblocks the Win16 application that was running before the Win16 VDM was preempted.

Additionally, the Win16 VDM provides stubs for a Windows 3.1 Kernel, User, GDI dynamic link libraries. It also includes code to handle the *thunking* (translation to and from 16-bit) of Windows APIs and messages.

Briefly, the process of thunking works this way: When a Win16 application issues an API call, the appropriate stub pushes the parameters onto the stack and converts the call to the equivalent 32-bit call. This 32-bit-equivalent call is issued to the Win32 subsystem, and any handles, parameters, or messages that are returned are converted back into 16-bits, then passed back to the application.

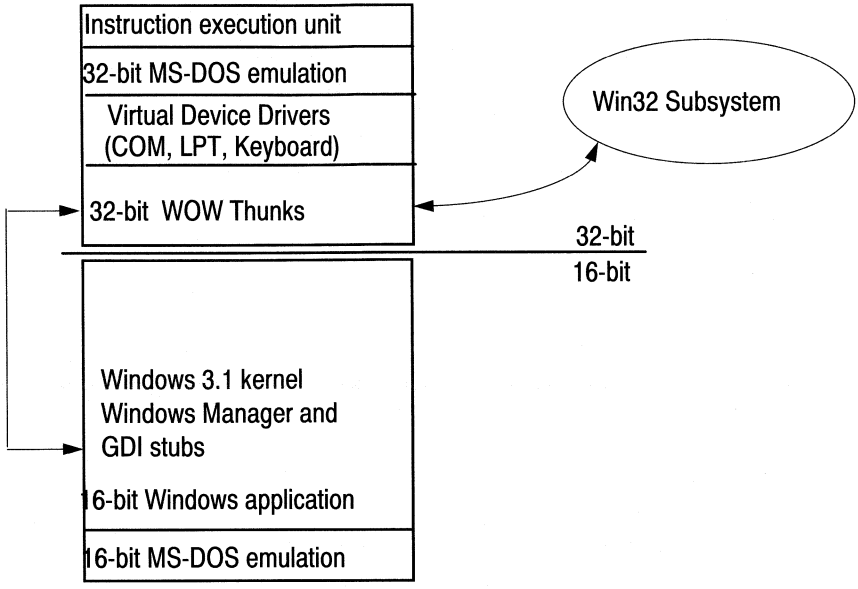


Figure 1.14 Structure of the Win16 VDM

OS/2 Subsystem

The OS/2 subsystem supports OS/2 1.*x* character-based applications on *x*86-based computers. This subsystem isn't supported on MIPS computers; however, OS/2 real-mode applications can run on a MIPS computer in the MS-DOS environment.

Bound applications, which are applications designed to run under either OS/2 or MS-DOS, will always run in the OS/2 subsystem if it is available.

The OS/2 subsystem is automatically loaded at logon and always remains active. Like the POSIX subsystem, the OS/2 subsystem is immediately paged out of memory after logon, and is only paged back in when needed.

POSIX Subsystem

The Windows NT POSIX subsystem is designed to meet the requirements of POSIX.1.

POSIX (Portable Operating System Interface for Computing Environments) is a family of over 20 standards being drafted by the Institute of Electrical and Electronic Engineers (IEEE) that define various aspects of an operating system. So far, only one of these standards—IEEE Standard 1003.1-1990 (usually referred to as POSIX.1)—has made the transition from draft to final form and gained a base of customer acceptance.

POSIX.1 defines a C language programming interface between applications and the operating system. It is an API based on ideas drawn from the UNIX® file system and process model. Because POSIX.1 does not address many areas needed to develop useful applications, such as graphical user interfaces, it is limited in its usefulness. Most applications written to the POSIX.1 API must therefore rely on non-POSIX operating system extensions to provide these needed services. Nevertheless, the U.S. government, in Federal Information Processing Standard (FIPS) 151-1, has adopted POSIX.1 compliance as a requirement for a broad range of operating system purchases.

To provide a fully compliant POSIX environment, certain file system requirements needed to be met. Hence, Windows NT also includes the new NTFS file system (described in Chapter 2, "New Technology File System") to meet these requirements.

The POSIX subsystem is automatically loaded at logon and then immediately paged out until it is required by a POSIX application. The POSIX subsystem remains active until Windows NT ends. It is paged back into memory only when it is needed to run a POSIX application.

Win32 Subsystem

The key environment subsystem is the Win32 subsystem. In addition to being able to run Win32 applications, this subsystem manages keyboard and mouse input and screen output for all subsystems.

The Win32 subsystem is responsible for collecting all user input (or messages, in this message-driven environment) and delivering it to the appropriate applications. The Win32 input model is optimized to take advantage of Windows NT's preemptive multitasking capabilities. Figure 1.15 shows how the Win32 subsystem handles input for Win32 and 16-bit Windows-based applications.

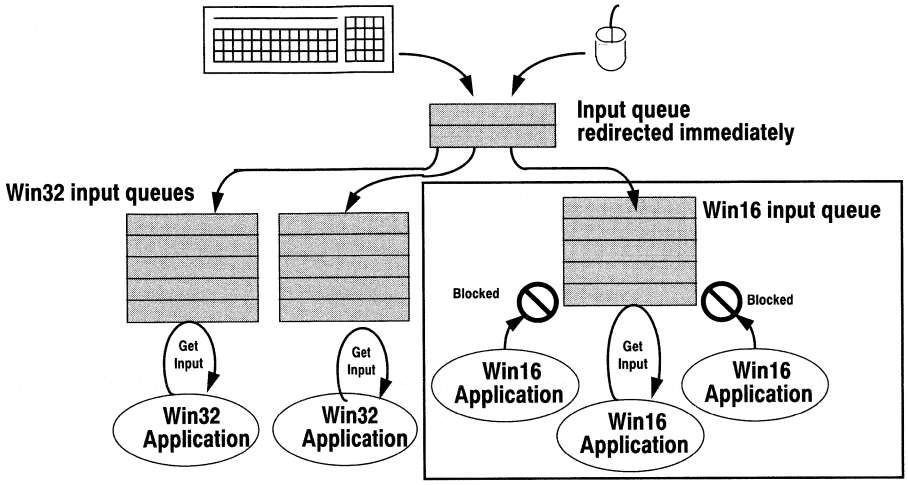


Figure 1.15 Synchronized (Win16) and desynchronized (Win32) input

Win32 uses a desynchronized input model for Win32 applications, and the synchronized input for 16-bit Windows-based applications.

For example, when the Win32 subsystem receives a message for a Win32 application, it stores the message in a single raw input queue. As soon as it can, the Win32 subsystem transfers the message to the input queue thread for the appropriate Win32 application. If the input queue thread stops retrieving its messages, no other Win32 applications are affected.

By contrast, all input messages for 16-bit Windows-based applications sit in a common queue. At any point in time all applications except the one retrieving messages from the input queue are blocked. However, as with Windows 3.1, if the executing application has some problem with retrieving messages off of the queue, or it does it very slowly, the remaining applications stay blocked.

Now we have described how Windows NT functions from the Hardware Abstraction Layer up to the application level. To conclude, the final section shows how various application types run on various hardware using Windows NT.

Application Types On Various Hardware Platforms

Windows NT runs on computers with Intel 80386 or 80486 processors, MIPS, or DEC Alpha processor. The following table shows how Windows NT supports applications of various types on these different hardware platforms.

Table 1.3 Application Compatibility

	Win32	MS-DOS and Windows 3.*x*	POSIX	OS/2 1.*x*
Intel x86	Source compatible	Runs application in a VDM	Source compatible	16-bit character-based only
DEC Alpha	Source compatible	Runs application in 286 emulation	Source compatible	Not available; can run real-mode applications in MS-DOS subsystem
MIPS R4000	Source compatible	Runs application in 286 emulation	Source compatible	Not available; can run real-mode applications in MS-DOS subsystem

CHAPTER 2

Windows NT Networking Architecture

Windows NT is a network operating system in the fullest sense. It is a complete operating system with fully integrated networking capabilities. These capabilities differentiate Windows NT from other operating systems such as MS-DOS, OS/2, and UNIX for which network capabilities are sold, installed, and managed separately from the core operating system itself.

The integrated networking support provided in Windows NT was designed to do these things:

- Provide peer networking capabilities. All Windows NT workstations can act as both network clients and servers. They can share their files and printers with other computers, and exchange messages over the network.
- Allow easy addition of networking software and hardware. The networking software integrated into Windows NT lets you easily add of protocol drivers, network card drivers, and even sophisticated domain management software.
- Interoperate with computers on existing networks. Windows NT systems can communicate using a variety of transport protocols and network adapters. It can even communicate over a variety of different vendors' networks.
- Support industry standard APIs to establish and end connections, transfer data, and browse lists of resources. Among the APIs applications running on Windows NT can use to access networks are NetBIOS, Sockets, and the Windows Network (WNet) interface.
- Support development of distributed applications. Windows NT provides a transparent Remote Procedure Call (RPC) facility. This allows an application on a local workstation to call functions provided by remote workstations as though the functions were provided by the local workstation.

This chapter describes the Windows NT networking architecture and how it achieves each of these goals. For perspective, the next section provides a brief explanation of two industry-standard models for networking—the Open System Interconnection (OSI) reference model and the Institute of Electrical and Electronic Engineers (IEEE) 802 project model. The remainder of the chapter describes Windows NT's networking components as they relate to the OSI and IEEE models.

Overview of Networking

In the early years of networking, several large companies, including IBM®, Honeywell, and Digital Equipment Corporation (DEC®), each had their own standard for how computers could be connected together. These standards described the mechanisms necessary to move data from one computer to another. These early standards, however, were not entirely compatible. Networks adhering to IBM's Systems Network Architecture (SNA) could not directly communicate with networks using DEC's Digital Network Architecture (DNA), for example.

In later years, standards organizations, including the International Standards Organization (ISO) and the Institute of Electrical and Electronic Engineers (IEEE), developed models that became globally recognized and accepted as the standards for designing any computer network. Both models describe networking in terms of functional layers.

OSI Reference Model

ISO developed a model called the Open Systems Interconnection (OSI) reference model. It is used to describe the flow of data between the physical connection to the network and the end-user application. This model is the best known and most widely used model to describe networking environments.

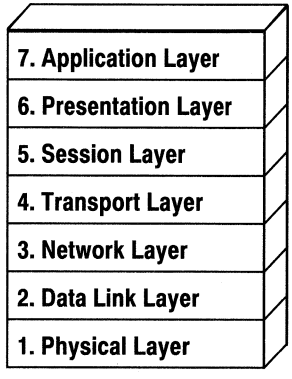


Figure 2.1 The OSI model

As shown in Figure 2.1, the OSI layers are numbered from bottom to top. The most basic functions, such as putting data bits onto the network cable, are on the bottom, while functions attending to the details of applications are at the top.

In the OSI model, the purpose of each layer is to provide services to the next higher layer, shielding the layer from the details of how the services are actually implemented. The layers are abstracted in such a way that each layer believes it is communicating with the associated layer on the other computer. In reality, each layer communicates only with adjacent layers on one computer. That is, for information to pass from Layer 5 on Computer A to Layer 5 on Computer B, it actually follows this the route illustrated by Figure 2.2:

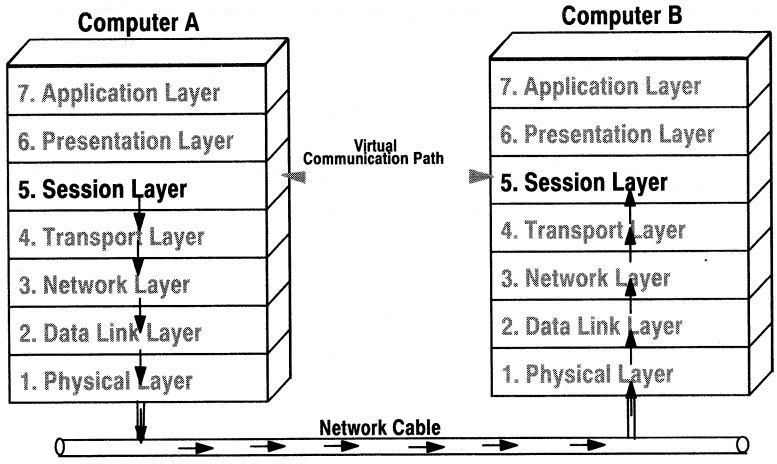


Figure 2.2 Communication between OSI layers

The following list describes the purpose of each of the seven layers of the OSI model and identify services that they provide to adjacent layers.

1. The Physical layer addresses the transmission of the unstructured raw bit stream over a physical medium (that is, the networking cable). The Physical layer relates the electrical/optical, mechanical, and functional interfaces to the cable. The Physical layer also carries the signals that transmit data generated by all the higher layers.

   This layer defines how the cable is attached to the network adapter card. For example, it defines how many pins the connector has and what each pin is used for. It also defines which transmission technique will be used to send data over the network cable.

2. The Data Link layer packages raw bits from the Physical layer into *data frames*, which are logical, structured packets in which data can be placed. The Data Link layer is responsible for providing the error-free transfer of these frames from one computer to another through the Physical layer. This allows the Network layer to assume virtually error-free transmission over the network connection.
3. The Network layer is responsible for addressing messages and translating logical addresses and names into physical addresses. This layer also determines the route from the source to the destination computer. It determines which path the data should take based on network conditions, priority of service, and other factors. It also manages traffic problems, such as switching, routing, and controlling the congestion of data packets, on the network.

   The Network layer bundles small data frames together for transmission across the network. It also restructures large frames into smaller packets. On the receiving end, the Network layer reassembles the data packets into their original frame structure.
4. The Transport layer takes care of error recognition and recovery. It also ensures reliable delivery of host messages originating at the Application layer. Similar to how the Network layer handles data frames, this layer repackages messages—dividing long messages into several packets and collecting small messages together in one packet—to provide for their efficient transmission over the network. At the receiving end, the Transport layer unpacks the messages, reassembles the original messages, and sends an acknowledgment of receipt.
5. The Session layer allows two applications on different computers to establish, use, and end a connection called a *session*. This layer performs name recognition and the functions needed to allow two applications to communicate over the network, such as security functions.

   The Session layer provides synchronization between user tasks by placing checkpoints in the data stream. This way, if the network fails, only the data after the last checkpoint has to be retransmitted. This layer also implements dialog control between communicating processes, regulating which side transmits, when, for how long, and so on.
6. The Presentation layer determines the form used to exchange data between networked computers. It can be called the network's translator. At the sending computer, this layer translates data from a format received from the Application layer into a commonly recognized, intermediary format. At the receiving end, this layer translates the intermediary format into a format useful to that computer's Application layer.

   The Presentation layer also manages network security issues by providing services such as data encryption. It also provides rules for data transfer and provides data compression to reduce the number of bits that need to be transmitted.

7. The Application layer serves as the window for application processes to access network services. This layer represents the services that directly support the user applications such as software for file transfers, for database access, and for electronic mail.

IEEE 802 Model

Another networking model developed by the IEEE further defines sublayers of the Data Link layer. The IEEE 802 project (named for the year and month it began—February 1980) defines the *Media Access Control* (MAC) and the *Logical Link Control* (LLC) sublayers.

As Figure 2.3 indicates, the Media Access Control sublayer is the lower of the two sublayers, providing shared access for the computers' network adapter cards to the Physical layer. The MAC layer communicates directly with the network adapter card and is responsible for delivering error-free data between two computers on the network.

The Logical Link Control sublayer, the upper sublayer, manages data link communication and defines the use of logical interface points (called Service Access Points—SAPs) that other computers can reference and use to transfer information from the LLC sublayer to the upper OSI layers.

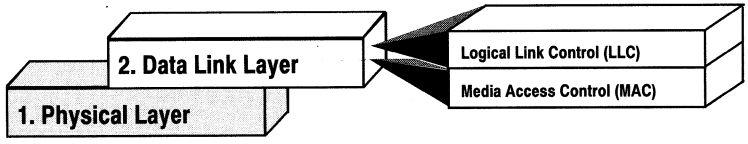


Figure 2.3 Logical Link Control and Media Access Control sublayers

In this chapter, we will describe the layered components of the Windows NT networking architecture, beginning in the following section with an overall description of that architecture.

Windows NT Networking Model

As with other architecture components of Windows NT, the networking architecture is built of layers. This helps provide expandability by allowing other functions and services to be added. Figure 2.4 shows all of the components that make up the Windows NT networking model.

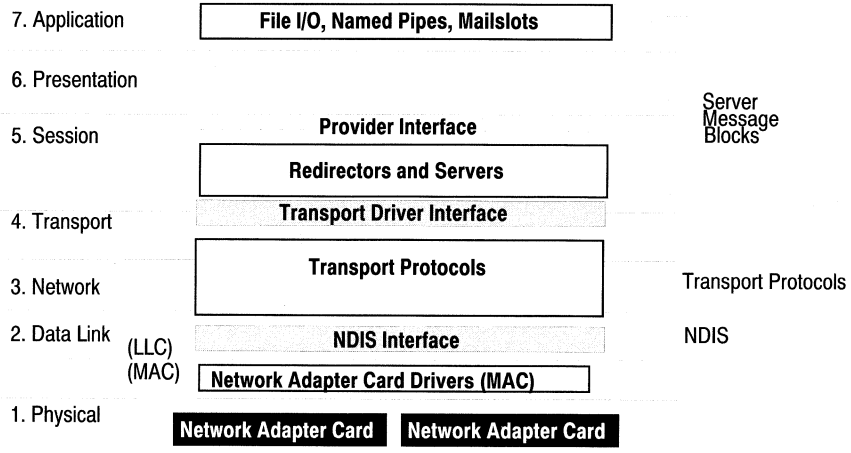


Figure 2.4 Windows NT networking model

Briefly, here's what each of the networking layers does:

The Windows NT networking model begins at the MAC sublayer where *network adapter card drivers* reside. These link Windows NT to the network via corresponding network adapter cards. Figure 2.4 includes network adapter cards to illustrate that you can use multiple network adapter cards running Windows NT.

The network model includes two important interfaces—the *NDIS 3.0 Interface* and the *Transport Driver Interface*. These interfaces isolate one layer from the next by allowing an adjacent component to be written to a single standard rather than many. For example, a network adapter card driver (below the NDIS Interface) does not need to include blocks of code specifically written for each transport protocol it uses. Instead, the driver is written to the NDIS interface, which solicits services from the appropriate NDIS-conformant transport protocol(s). These interfaces are included in the Windows NT networking model to allow for portable, interchangable modules.

Between the two interfaces are *transport protocols*, which act as data organizers for the network. A transport protocol defines how data should be presented to the next receiving layer and packages the data accordingly. It passes data to the

network adapter card driver through the NDIS Interface, and to the redirector through the Transport Driver Interface.

Above the Transport Driver Interface are *redirectors* which "redirect" local requests for network resources to the network.

When multiple redirectors are included on a system, the Provider Interface routes requests from applications to the appropriate redirector.

The following sections describe each of the Windows NT networking layers in detail, beginning with the network adapter card driver layer and the NDIS Interface.

NDIS-Compatible Network Adapter Card Drivers

Until the late 1980s, many of the implementations of transport protocols were tied to a proprietary implementation of a MAC-layer interface defining how the protocol would converse with the network adapter card. This made it difficult for network adapter card vendors to support the different network operating systems available on the market. Each network adapter card vendor had to create proprietary interface drivers to support a variety of protocol implementations for use with several network operating system environments.

In 1989, Microsoft and 3Com® jointly developed a standard defining an interface for communication between the MAC layer and the transport protocol drivers that reside on layers 3 and 4 of the OSI model. This standard is known as the *Network Device Interface Specification (NDIS)*. NDIS allows for a flexible environment of data exchange. It defines the software interface—called the NDIS interface—used by transport protocols to communicate with the network adapter card driver.

Windows NT currently supports device drivers and transport protocols written to NDIS version 3.0. (NDIS 3.0 conforms to the device driver standards established for Windows NT.)

NDIS allows multiple network adapter cards on a single computer. Each network adapter card can support multiple transport protocols. The advantage of supporting multiple protocol drivers on a single network card is that Windows NT computers can have simultaneous access to different types of network servers, each using a different transport protocol. For example, a computer can have access to both a Windows NT Advanced Server and a Novell® NetWare® server simultaneously.

The flexibility of NDIS comes from the standardized implementation used by the network industry. Any NDIS-conformant protocol can pass data to any NDIS-conformant network adapter card driver, and vice versa. A process called *binding* is used to establish the initial communication channel between the protocol driver and the network adapter card driver.

Unlike previous NDIS implementations, Windows NT does not need a Protocol Manager module to link the various components at each layer together. Instead, Windows NT uses the information in the Registry (described in Chapter 4, "Windows NT Configuration Registry") and a small piece of code called the *NDIS wrapper* that surrounds the network adapter card driver.

Transport Protocols

Sandwiched between the NDIS 3.0 interface and the Transport Driver Interface are transport protocol device drivers. These drivers communicate with a network adapter card via a NDIS-compliant device driver.

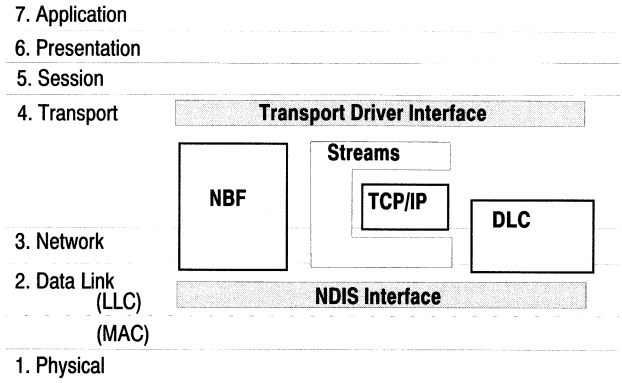


Figure 2.5 Transport Protocols

Windows NT includes multiple transports, including NBF, TCP/IP, and DLC. NBF is a transport protocol derived from NetBEUI and provides compatibility with existing LAN Manager, LAN Server, and MS-Net installations. TCP/IP is a popular routable protocol for wide-area networks. DLC is provides an interface for access to mainframes and network attached printers.

DLC

The DCL device driver allows access directly to the data link at the Network Layer of the OSI model. There is very little protocol information added to the transmitted frame when communicating via the DLC protocol. Thus, the DLC protocol is very simple and flexible in nature, placing higher order functions within other modules as they are needed. This makes DLC perfect for devices, such as network-attached printers, with "need networking" coding on ROM and for hardware with remote IPL (initial program load) PROMs, such as on some network adapter cards.

Windows NT supports the DCL protocol chiefly to use with network-attached printers and for communication to mainframes via SNA services.

Network-attached printers such as the HP® III si use the DLC protocol because the frames received are very easy to take apart and because functionality can easily be coded onto ROM.

SNA Server uses the DLC protocol as a method of communicating to a mainframe. The downstream PC capability of SNA services uses DLC to mimic a mainframe, thereby allowing devices with full PU2-compliant software to communicate to the mainframe via the SNA services unit.

NBF

The NBF transport protocol is a descendent of the NetBEUI (NetBIOS Extended User Interface) protocol, first introduced by IBM® in 1985. NetBEUI was designed to be a small and efficient protocol for use on a departmental LAN of 20 to 200 workstations. This protocol has powerful flow control and tuning parameters. It also has robust error detection.

NetBEUI is not routable. The original design assumed broader connectivity services could be added as the network grew by including gateways. A *gateway* is a device that allows one type of network to communicate with a different type of network.

Microsoft has supported the NetBEUI protocol in all of its networking products since Microsoft's first networking product, MS-Net, was introduced in the mid-1980s.

When we talk about NBF, it is important to understand we are talking about the Transport layer protocol, not the programming interface NetBIOS. Earlier implementations on MS-DOS and OS/2 provided the NetBIOS programming interface as part of the transport's device driver. In contrast, the Windows NT implementation separates the programming interface from the transport protocol to increase flexibility in the layered architecture.

TCP/IP and Streams

TCP/IP is implemented slightly differently than NBF or DLC. Instead of being a single device driver bound directly to the NDIS device driver, TCP/IP is wrapped in the Streams driver. Calls to the TCP/IP transport protocol driver must first go through the upper layer of the Streams device driver to TCP/IP, then back through the lower layer of Streams to the NDIS device driver.

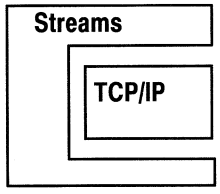


Figure 2.6 TCP/IP and Streams

Using Streams makes it easier for developers to port other protocol stacks to Windows NT. It also encourages protocol stacks to be organized in a modular, stackable style, which is in keeping with the original OSI model.

Transport Driver Interface

Just as NDIS provides a common interface to which transport protocols and network adapter cards can be written, the Transport Driver Interface provides a common interface for networking components that communicate at the Session layer of the OSI model.

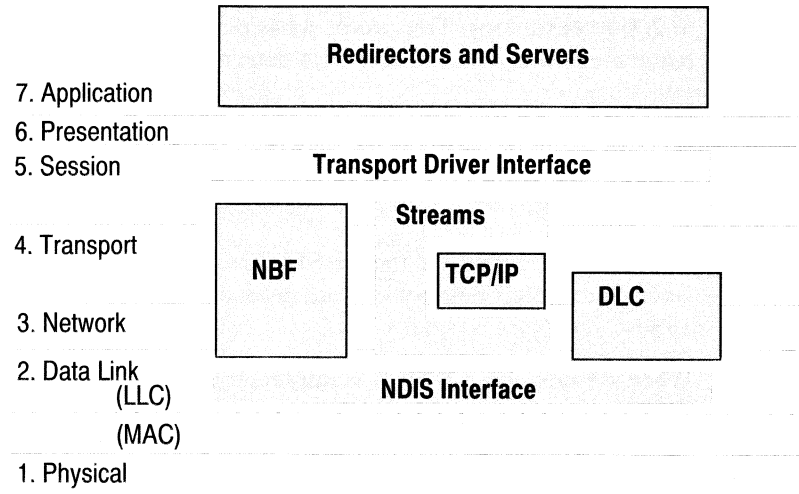


Figure 2.7 The Transport Driver Interface

One of the chief goals of the Windows NT networking model is to provide a platform on which other vendors can develop distributed applications. The NDIS and TDI boundaries help to do this by providing a unified interface at each significant break point in the model. These boundaries allow software components above and below a level to be mixed and matched without reprogramming.

The TDI not a single program, but a protocol specification to which the upper bounds of transport protocol device drivers are written. Windows NT also includes a TDI driver that handles interrupt request packet traffic from multiple TDI providers. (You can find this driver is in the device driver tree, but not in the Registry.)

Above the TDI layer reside redirectors and servers, described in the next section.

Redirectors and Servers

The first design goal of the networking system is to allow users to share and use files and printers. This is accomplished chiefly by two components—the server and the workstation.

On a network, workstations and servers can be organized in one of two ways. Some networks have dedicated servers which share resources and dedicated workstations which access those shared resources. Other networks, including those running Windows NT, allow each computer to act as both a client and a server. This means a computer that is routinely used as a client also shares one or more of its resources with other computers. This second approach is called *peer-to-peer networking* or *workgroup computing*.

For a Windows NT workstation, a component called the redirector resides at this layer above the TDI. The redirector is the component through which one computer gains access to another computer. The Windows NT redirector allows connection to LAN Manager, LAN Server, and MS-Net servers. This redirector communicates to the protocol stacks to which it is bound to via the TDI layer.

The redirector is implemented as a Windows NT file system driver. Implementing a redirector as a file system has several benefits:

- Allows applications to call a single API (namely, Windows NT I/O functions) to access files on local and remote computers. From the I/O Manager's perspective, there is no difference between accessing files stored on a remote networked computer and accessing those stored locally on a hard disk.
- Runs in kernel mode so it can directly call other drivers and other Kernel components such as the Cache Manager. This access can be used to optimize the redirector's performance.
- It can be loaded and unloaded dynamically, just like any other driver.
- It can coexist easily with other redirectors.

When a process on a Windows NT workstation tries to open a file that resides on a remote computer connected by a LAN Manager network, these steps occur:

1. The process calls the I/O Manager to request that the file be opened.
2. The I/O Manager recognizes that the request is for a file on a remote computer, so it passes it to the redirector file system driver.
3. The redirector passes the request to lower-level network drivers that transmit it to the remote Server for processing (see Figure 2.8).

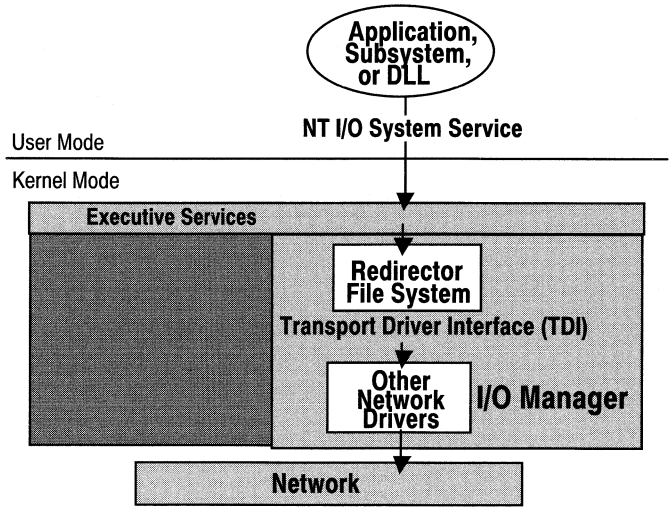


Figure 2.8 Client-side processing using the redirector

Windows NT also includes a component called the server at the layer above the TDI. Like the redirector, the server is implemented as a file system driver and directly interacts with various other file system devices to satisfy command requests such as reading or writing to a file.

The server entertains the connections requested by client-side redirectors, and provides them with access to the resources they request.

When a Windows NT server receives a server message block from a remote workstation asking it to read a file that resides on the local hard drive, these steps occur:

1. The low-level network drivers receive the request and pass it to the server driver.
2. The server passes a file read request to the appropriate local file system driver.
3. The local file system driver calls lower-level disk device drivers to access the file.
4. The data is passed back to the local file system driver.
5. The local file system driver passes the data back to the server.
6. The server passes the data to the lower-level network drivers for transmission back to the client computer.

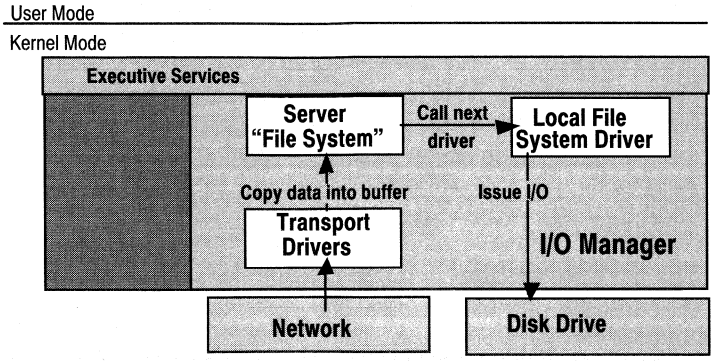


Figure 2.9 Server-side processing using the server

Provider Interface Layer

The Provider layer really includes three kinds of components:

- "Provider" DLLs, such as LanmanWorkstation, meant to work with a corresponding redirector, and other DLLs, such as LanmanServer which coordinates with the server
- A Multiple UNC Provider (MUP) which coordinates among redirectors
- A Multiple Provider Router (MPR), which coordinates among providers

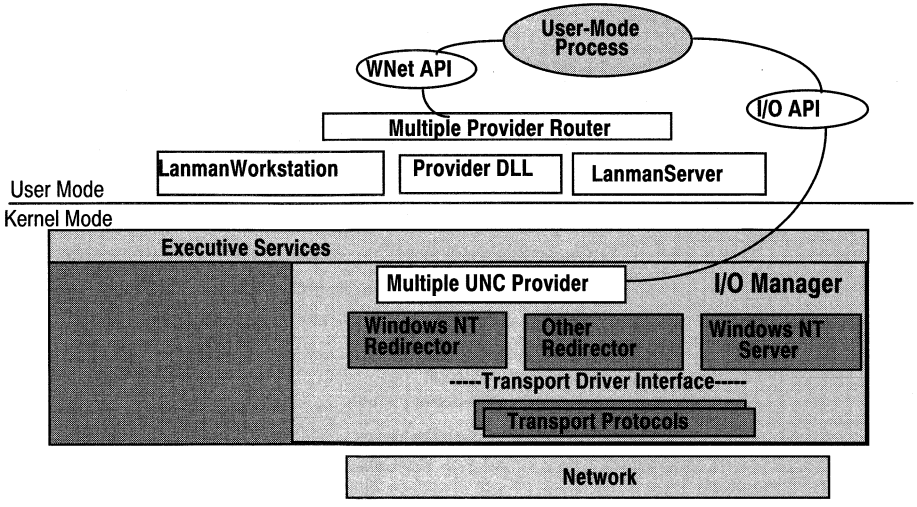


Figure 2.10 Provider Interface components

This Provider layer spans the line between kernel and user modes to manage commands that may cause network traffic. From the application view point there are two sets of commands which can cause network traffic—UNC commands and WNet commands. UNC commands are sent to the MUP which locates the redirector that can make a connection to the UNC name. WNet commands are passed to the MPR which passes the request to each redirector in turn until one is found which can satisfy the request. Any I/O call, such as Open, can contain a UNC name and WNet calls.

The next two sections describe the functions of the Multiple UNC Provider and the Multiple Provider Router.

Multiple UNC Provider

The Multiple UNC Provider (MUP) is a resource locator that runs in kernel-mode memory. The type of resources it locates are UNC names.

UNC stands for "Universal Naming Convention," a method of identifying a share point on a network. UNC names start with two backslashes followed by the server name. All other fields in the name are separated by a single backslash. A typical UNC name would appear as follows:

```
\\server\share\subdirectory\filename
```

Not all of the components of the UNC name need to be present with each command; for example, the following is sufficient to find a server to get a list of a server's sharepoints:

```
\\server
```

Unlike the NDIS and TDI boundary layers, MUP is a program. NDIS and TDI simply define ways for a component on one layer to communication with another over specifically defined paths, or bindings. MUP too has defined paths to redirectors, or as the name implies, *UNC providers*.

MUP receives commands containing UNC names from applications. MUP sends each UNC name to each of the registered UNC providers—LanmanWorkstation and any others that may be installed. When a provider identifies a UNC name as its own, MUP automatically redirects future instances of that name to that provider.

When applications make I/O calls which contain UNC names, the MUP directs them to the appropriate redirector file system driver. The call is routed to its redirector based on the handle on the I/O call.

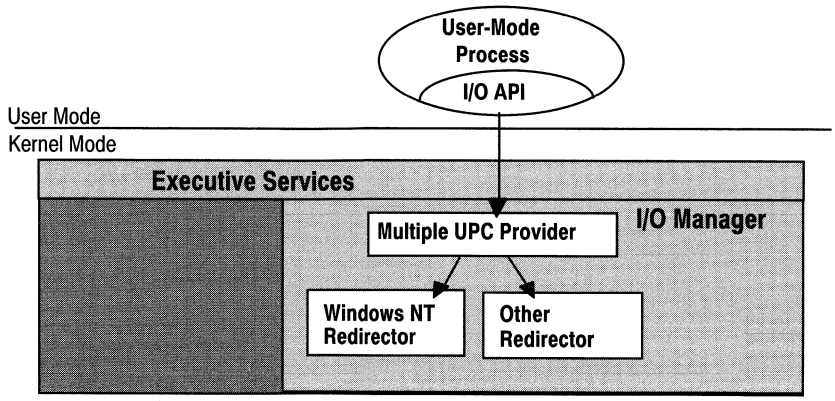


Figure 2.11 Multiple UNC coordinates among redirectors

Multiple Provider Router

The Multiple Provider Router is a software component that sits above the provider DLLs and allows applications using the WNet API to access resources on multiple networks.

WNet APIs are a subset of the Win32 APIs specifically designed to allow applications on Windows NT workstations to connect to multiple networks, browse the resources of computers on those networks, and transfer data between computers of various networks. File Manager, for example, uses the WNet interface to provide its network browsing and connection facilities.

Each network vendor supplies a provider DLL that acts as an intermediary between the Multiple Provider Router and the vendor's redirector. The provider DLL provides a standard interface that the MPR can call to perform resource browsing and connection management. When an application calls a WNet function, MPR simply routes the command to the appropriate provider DLL, which in turn communicates with its redirector. The redirector then carries out the requested action.

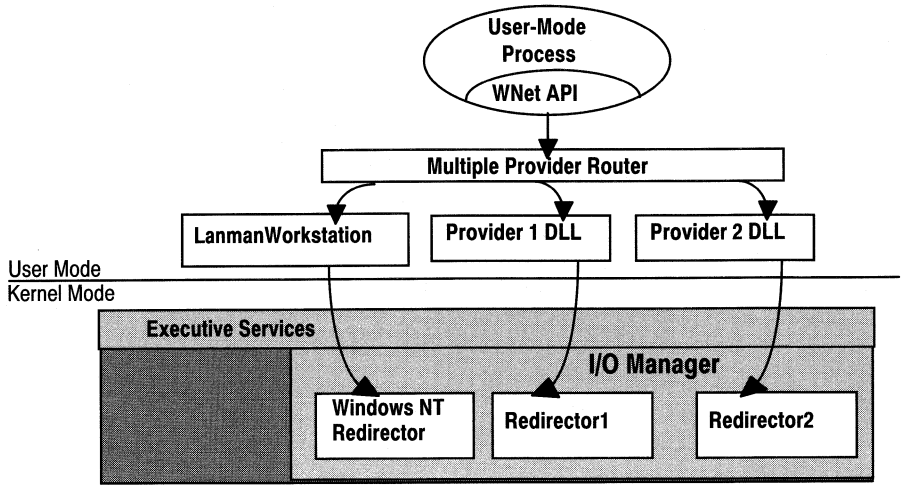


Figure 2.12 Multiple Provider Router

Once a connection is established with a remote computer, the Win32 file I/O APIs provide the ability to transfer data to and from the remote computer.

Application Layer Interface

In addition to providing file and print sharing capabilities, Windows NT provides several mechanisms for building distributed applications.

In distributed computing, the goal is to divide the computing task into two sections. The first section requiring minimal resources runs on the client's workstation. The other section requiring large amounts of data, number crunching, or specialized hardware runs on the server. A connection between the client and the server at a process-to-process level allows data to flow in both directions between the client and server.

The Windows 32-it application programming interface (Win32 API) includes an extensive set of network APIs that can replace those that network providers have previously needed to supply. Win32 exposes driver-style interfaces similar to the WinNet API provided by Windows version 3.0 so that third-party vendors can plug their network services into the Windows Open Architecture.

Some of the new 32-bit network APIs include file, print, named pipes, mailslots, server browsing, and machine configuration. This means applications can rely on a consistent programming interface regardless of the underlying network. Even if a network is not present, the APIs are still available and will return appropriate error codes.

The Win32 API includes peer-to-peer named pipes, mailslots, and APIs to enable remote procedure call (RPC) compilers. With Win32 for example, a mail-server vendor can build a messaging service on named pipes and asynchronous communication that will run on top of any network operating system, protocol stack, or network card, each of which could come from a different network vendor.

Of named pipes, mailslots, and RPC, RPC is the most portable mechanism. RPCs use other IPC (interprocess communication) mechanisms—including named pipes, and the NetBIOS and Windows Sockets interfaces—to transfer functions and data between client and server computers.

Named pipes and mailslots are implemented to provide backward compatibility with existing LAN Manager installations and applications.

Windows NT also includes NetBIOS and Windows Sockets interfaces for building distributed applications. The NetBIOS interface is also provided for backward compatibility with existing LAN Manager installations and applications. Windows Sockets is a Windows implementation of the widely-used UC Berkeley Sockets programming interface.

This following three sections describes these five mechanisms used in distributed applications.

Remote Procedure Calls

Much of the original design work for a Remote Procedure Call (RPC) facility was started by Sun Microsystems®. This work has been carried forward by the Open Software Foundation (OSF) as part of their overall Data Communications Exchange (DCE) standard. The Microsoft RPC facility is compatible with the OSF/DCE-standard RPC. It is important to note that it is *compatible* and not compliant. Compliance in this situation implies one started with the OSF source code and worked forward. For a number of reasons Microsoft developed its RPC facility from the ground up. The RPC facility is completely interoperable with other DCE-based RPC systems such as the ones for HP® and IBM AIX® systems.

The RPC mechanism is unique in that it uses the other IPC mechanisms to establish communications between the client and the server. RPC can use named pipes, NetBIOS, or Windows Sockets to communicate with remote systems. If the client and server are on the same computer it can use the Local Procedure Call (LPC) facility to transfer information between processes and subsystems. This makes RPC the most flexible and portable of the IPC choices available.

To understand RPC, we need to understand structured programs, which can be viewed as having a "backbone" to which a series of "ribs" can be attached. The backbone is the mainstream logic of the program which should rarely change. The ribs are the procedures the backbone calls on to do work or functions.

In traditional programs, these ribs are statically linked. Beginning with OS/2 and Windows, structured programs could dynamically link ribs through the use of dynamic link libraries (DLLs). With DLLs, the procedure code and the backbone code are in different modules. The DLL can thus be modified or updated without changes to the backbone.

RPC means that the backbone and the ribs can exist on different computers, as shown in Figure 2.13.

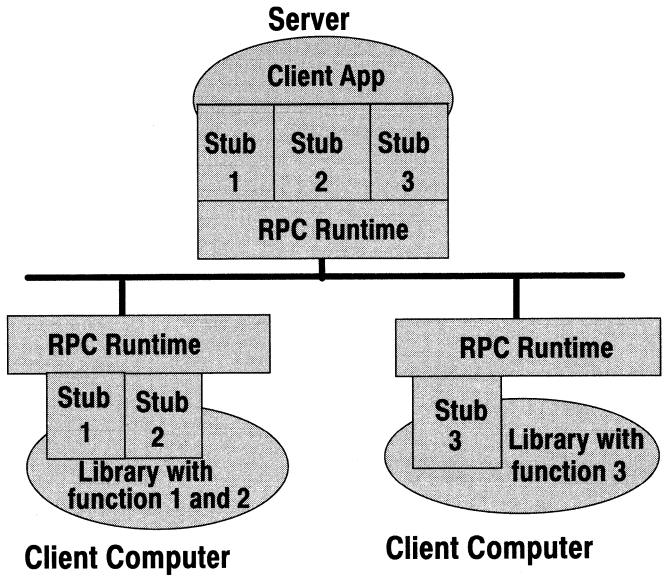


Figure 2.13 Remote Procedure Call facility

In this figure, the client application was developed with a specially-compiled "stub" library. The client application thinks it is calling its own subroutines. In reality, these stubs are transferring the data and the function down to a module called the RPC Runtime. This module is responsible for finding the server which can satisfy the RPC command. Once found, the function and data are sent to the server where it is picked up by the RPC Runtime module on the server. The server piece then loads the needed library for the function, builds the appropriate data structure, and calls the function. The function thinks it is being called by the client application. When the function is completed, any return values are collected, formatted, and sent back to the client via the RPC Runtime modules. When the function returns to the client application it has the appropriate returned data, or it has an indication that the function failed in stream.

Named Pipes and Mailslots

Named pipes and mailslots are implemented slightly differently that the other IPC mechanisms. They are actually written as file systems. Thus, in the Registry you find entries for NPFS (Named Pipes File System) and MSFS (Mailslot File System). (See Chapter 4, "Windows NT Configuration Registry" for more information about the Registry). As file systems they share common functionality, such as security, with the other file systems. In addition, local processes can use named pipes and mailslots with other processes on the local computer without going through the networking components. Remote access to named pipes and mailslots, as with all of the file systems, is accomplished via the redirector.

Named pipes are based on OS/2 API calls. Most of the calls however have been ported into the Win32 base API set. Additional asynchronous support has been added to the named pipes to make support of client/server applications easier. In addition to the APIs ported from OS/2, Windows NT provides special API which increases the security of using named pipes.

A new feature added to named pipes is *impersonation*. In impersonation, the server can change its security identify to that of the client on the other end. For example, suppose a database server system uses named pipes to receive read and write requests from clients. When a request comes in the database server program can impersonate the client before attempting to perform the request. So even if the server program does have authority to perform the function, the client may not, and the request would be denied. (For more information on impersonation, see Chapter 3, "Windows NT Security Model.")

The mailslot implementation in Windows NT is a subset of the Microsoft OS/2 LAN Manager implementation. Windows NT implements only second class mailslots, not first class mailslots. Second class mailslots provide *connectionless* messaging for broadcast messages and the like. Delivery of the message is not guaranteed, though the delivery rate on most networks is quite high. It is most useful for identifying other computers or services on a network and wide-scale notification of a service.

NetBIOS and Windows Sockets

During the 1980s, the NetBIOS and Sockets Transport layer interfaces became standards within the IBM PC-compatible and UNIX worlds, respectively. To provide compatibility for applications that rely on these interfaces, both are supported by Windows NT.

The NetBIOS and Windows Sockets APIs are supplied by separate DLLs. These DLLs communicate with corresponding drivers in the Windows NT Executive. As shown by Figure 2.14, the NetBIOS and Windows Sockets drivers then bypass the Windows NT redirector and communicate with protocol drivers directly using the TDI.

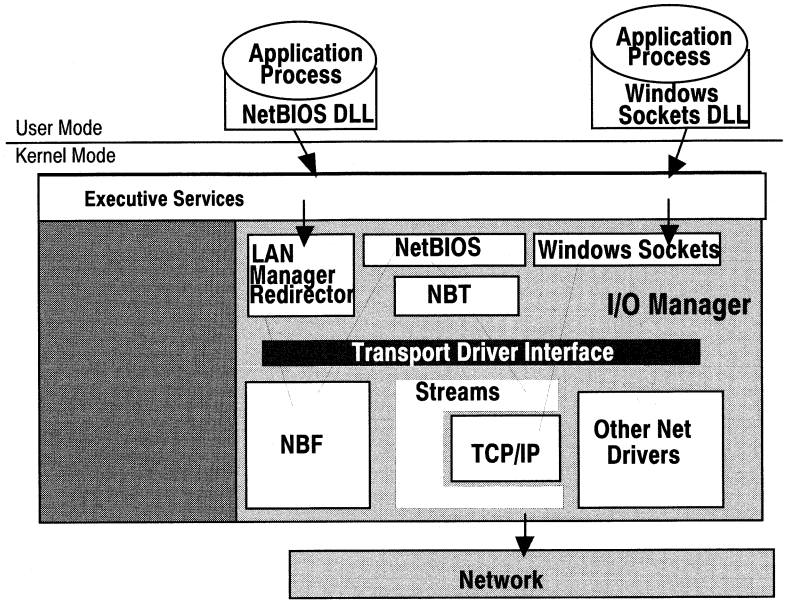


Figure 2.14 NetBIOS and Windows Sockets support

An interesting side effect of this relationship is that applications using the NetBIOS and Windows Sockets interfaces can only communicate using protocol drivers that support the TDI. Protocol drivers such as Novell's IPX driver, which supports a private interface known only by the Novell redirector, can't be accessed using the NetBIOS or Windows Sockets APIs.

NetBIOS can be used by the NBF (NetBEUI) transport protocol and by the NBT (NetBIOS over TCP/IP) protocol. NBT is a protocol that communicates with Streams for access to the TCP/IP transport protocol driver.

As shown by Figure 2.14, TCP/IP also uses the Windows Sockets interface.

CHAPTER 3

Windows NT Security Model

Windows NT brings a whole new meaning to the word security, when applied to a PC operating system. This operating system does not have just a single security component, but has a security structure that is pervasive throughout the entire operating system. Windows NT provides security in the areas of user authentication (when users log on), remote access validation, and file access, to name but a few.

This chapter begins with a global picture of the security architecture, showing how security fits into the Microsoft Windows NT operating system. It covers design goals, class C2-level security requirements, and the Windows NT security architecture.

Next, the chapter describes identification and authentication. This will reinforce the break in tradition from what we have come to expect from security in the desktop personal computer environment. In a class C2-level security environment, a user accessing secure objects must be identified by a unique identifier, hence the requirement for identification and authentication.

Then the chapter describes how account management is implemented in Windows NT, looking briefly at how accounts are managed within and across security domains.

Finally, the chapter explores the access control design within the Windows NT security model. The chapter also looks at how Windows NT uses this access control design to grant or deny access to a particular object, and how it determines auditing.

The Need for Security

Regardless of how secure you believe your systems to be, it is the implementation of an organization's security policy by the system administrator that determines how secure your systems really are.

Consider this scenario:

In an effort to protect user's passwords, an organization insisted that users change their passwords every 30 days. The system administrator made the appropriate changes by setting the system up to require the periodic password changes. However, this does not stop users from toggling between their two favorite passwords, or from changing a password twice in succession to end up with the same password, thus leading to a potential security breach.

What the organization really needed was forced password change and a password history which would prevent users from reusing their favorite passwords. This set of features is included in the Windows NT password policy.

The bottom line is that secure computing depends on good system management practice, a clear understanding of the security policies, and facilities offered by the operating system.

Windows NT Security Design Goals

The Windows NT design team had several key goals for the Windows NT security architecture. Specifically, the team decided that this architecture must do the following:

- Provide a consistent, robust local security model
- Support a variety of environment subsystem security models
- Meet the security requirements that commercial users demand to run line of business applications
- Meet the requirements for the United States Department of Defense C2 rating

Department of Defense's Trustworthiness Criteria

Widespread availability of "trusted computer systems" is one of the major goals of the United States National Computer Security Center. (NCSC is formally known as the Department of Defense Computer Security Evaluation Center.)

Trusted computer systems communicate with one another based on trust relationships, a method of authentication in which a user has one and only one user account in one domain, yet can access the entire network. (For more information about trusted computers, see the *Windows NT Advanced Server Concepts and Planning Guide*.)

To reach this goal of widely-available trusted computer systems, the Center created a formal measurement criteria called the *Department of Defense Trusted Computer System Evaluation Criteria*, also known as the "Orange Book." The Orange Book is the standard against which computer systems can be evaluated for security, or "trustworthiness."

The Orange Book's criteria for trustworthiness results from the culmination of various research efforts that were initiated by the Department of Defense in the late 1960s. This criteria is divided into four divisions:

Table 3.1 Department of Defense security divisions

Division	Corresponding classes
D: Minimal Protection	(none)
C: Discretionary (Need-To-Know) Protection	C1: Discretionary Security Protection C2: Controlled Access Protection
B: Mandatory Protection	B1: Labeled Security Protection B2: Structured Protection B3: Security Domains
A: Verified Protection	A1: Verified Design

Divisions A though D are ordered hierarchically where Division A is reserved for systems providing the most comprehensive security. Each division represents a major improvement in the overall confidence one can place in the system for the protection of sensitive information. As shown by Table 3.1, within divisions C and B there are a number of subdivisions know as *classes*, also ordered hierarchically.

The criteria defines discretionary access control as "a means of restricting access to named objects based on the identity of subjects and/or groups to which they belong." The controls are discretionary in the sense that a user with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject.

Windows NT is designed to meet the security requirements for the class C2-level security. A C2 rating requires that the computer system meets a certain standard for security policy, accountability, assurance, and documentation. Class C2-level security, as applied to Windows NT, provides for the following security features:

- The owner of a resource, at his or her discretion, is able to control access to that resource and what can be done with it. This access control must provide the ability to include or exclude individual users or named groups.
- Memory is protected so that its contents cannot be read after it is freed by a process.
- Users are required to identify themselves, each with a unique identifier and a password, when they logon. All auditable actions by the user must be associated with the user's identifier.
- System administrators are able to audit security-related events. Access to this audit data is limited to authorized administrators.

As the Orange Book puts it, the criteria "constitutes a uniform set of basic requirements and evaluation classes for assessing the effectiveness of security controls built into... systems."

The Security Model

Looking at it from a security standpoint, Windows NT includes three groups of components—the Executive, environment subsystems (such as Win32), and integral protected subsystems (such as the Security subsystem).

As shown in Figure 3.1, the Windows NT security model is made up of three components—Logon Process, Security Subsystem, and Security Reference Monitor.

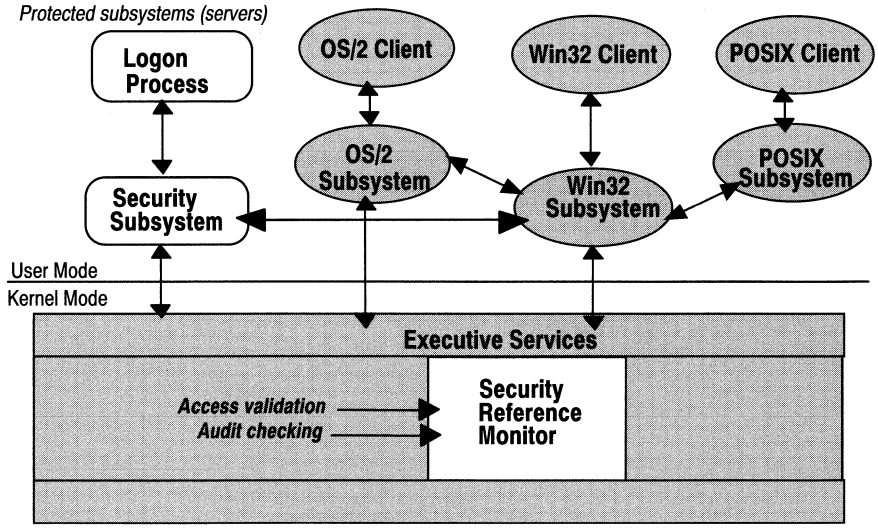


Figure 3.1 Windows NT security architecture

Before users can gain access to anything on a Windows NT computer, they must be authenticated. Authentication of users is handled by the Logon Process and the Security Subsystem. After successful authentication, the user's shell process is tagged with a security access token. From this point forward, whenever the user attempts to access a protected object, the Security Reference Monitor will run an access-validation routine against the user's security access token.

Any other process that the user creates will be tagged with the user's access token, and therefore any process acting on the user's behalf will have access validation routines run against the user's security access token. Thus, we can track any security-related events that the user may initiate, which is one of the requirements for a C2 rating. (Authentication and access determination are discussed in detail later in this chapter.)

In general, the Windows NT security model provides two types of security—discretionary access control and system security.

Discretionary access control refers to security information associated with an object which is controlled by someone with authority for that object. In most secure operating systems, for example, the owner of a file has control over who may access that file. The owner is said to have discretionary access control over the file.

The Windows NT security architecture defines discretionary access control mechanisms for explicitly allowing or denying access to users or to groups of users.

System security refers to a more pervasive view of security. In most secure systems, it is not only necessary to provide mechanisms to protect information, but also to monitor activity, particularly activity that might be part of an attempt to breach security. One important aspect of this monitoring is the generation of audit messages as a result of successful or unsuccessful attempts to access objects —for example, files and directories.

Currently, Windows NT supports the generation of audit messages for a number of security-related events, including successful or unsuccessful logon attempts, security account database changes, and file access attempts.

The next several sections describe elements that make up system security for Windows NT, including object-level security, user identification, and user authentication. Later, a section called "Discretionary Access Control" describes that aspect of the Windows NT security model.

Objects and Object-Level Security

To fully appreciate the pervasiveness of security within Windows NT, you must first understand the granularity to which security is distributed throughout the system. To accomplish this you need to understand the two object categories that interact with the security subsystem—executive objects and user-mode objects.

As illustrated in Figure 3.2, an object is made up of an object header and a body. Object headers contain data that is common to all objects—object name, object directory, and so on. Each object has an object body whose format and contents

are unique to its object type. All objects of the same type share the same object format.

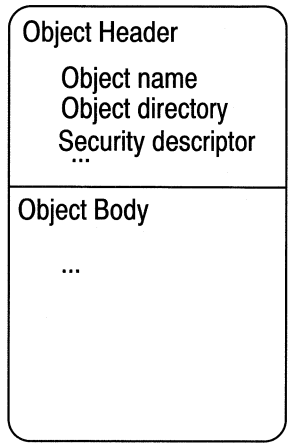


Figure 3.2 Windows NT object

In the Windows NT Executive, an object is defined as a single run-time instance of a statistically defined object type. An *object type* (sometimes called an object class) is a system defined data type, which consist of services that operate on instances of the data type, and a set of object attributes. The following table list some of the objects that are defined in the Executive.

Table 3.2 Executive objects

Executive object type	Represents
File	An instance of an open file or I/O device
Process	A program invocation, including the resource required to run the program
Access token	A tamper-proof ID containing security information about a logged on user

One of the reasons for having Environment subsystems in Windows NT is to facilitate the projection of several different types of operating system environments such as, Win32 and POSIX. From a user's viewpoint, there is no single concept of what constitutes an object. Environment susbsystems define their own objects, known as user-mode objects, for the use of application developers.

Container and Non-Container Objects

There are two categories of objects that are recognized by this architecture—container and non-container objects. A *container object* is one which logically contains other objects. For example, a file system directory logically contains files and other directories. In this example, the directory is a container object and files are *non-container objects.*

This distinction between container and non-container objects is important to establishing protection-inheritance rules. (Inheritance for container and non-container objects is discussed further in a section called "Access Control Inheritance" later in this chapter.)

User and Group Accounts

Users are granted access to objects by way of their user and group accounts.

Users typically log on to a user account. A *user account* is an account specifically set up for the user. It includes information, such as a password, which is used to authenticate the user.

From the standpoint of the Windows NT security architecture, a user account is identified by a unique security identifier (SID). A user account may also have a character-string name, but that is considered a convenience for users. The account's SID is always considered the authoritative value identifying a user account. (SIDs are described in the next section.)

Users may belong to groups. (A group is also represented by a SID corresponding to a group account.) Windows NT defines two types of groups—global groups and local groups. Global groups may have zero or more users as members. Local groups may include users and global groups as members. (For a complete discussion on local and global groups, see the *Windows NT Advanced Server Concept and Planning Guide.*)

Groups are convenient for purposes of resource administration. For example, rather than list all users that may access a particular file, you can create a group for all those users, then grant access to the group. In this way, if the group membership changes, only the group definition needs to be updated, not all the access validation information on every object to which the group has access.

Account Identifiers

The Windows NT security architecture makes extensive use of special identifiers called *security identifiers* (SIDs), which are used to uniquely identify a user, a group, some type of security authority. By using SIDs for user and group identification, the architecture provides the following properties:

- Unambiguous user and group identification.
- Portable protection information where protection information assigned in one security domain may be properly enforced within another security domain.

Unambiguous user and group identification means that a SID is unique across time and space. That is, even if two users have the same character string name associated with them, such as "Jack Smith," their SID values will be different. Similarly, if you deleted a user account called SallyMgr, then recreated it, the new account would have a different SID.

To illustrate portable protection information, suppose protection information is assigned to files on a removable hard drive. Further, suppose this protection allows the users with SIDs S-1-5-20-725-364-42-1003 (AshaB at XYZ corporation) and S-1-5-20-168-47-315-1007 (MarwanK of ACME Computers) the right to read and write to these files. The protection was assigned by AshaB at XYZ corporation. AshaB then mailed the removable drive to MarwanK at ACME Computers. When MarwanK connects the drive to his system, all the files are still protected so that only AshaB and MarwanK can modify them.

For the purpose of this document SIDs are represented using the following format:

```
S-1-X-Y1-Y2-.......
```

In this syntax,

- The prefix "S-1" indicates that this a revision-1 SID
- "X" is the value representing the identifier-authority
- $Y_1, Y_2, .. Y_n$ are values representing the sub-authority values

For example, the string S-1-5-2 represents a revision-1 SID, with an identifier-authority value of 5, and a single sub-authority value of 2.

Figure 3.3 illustrates the SID data structure.

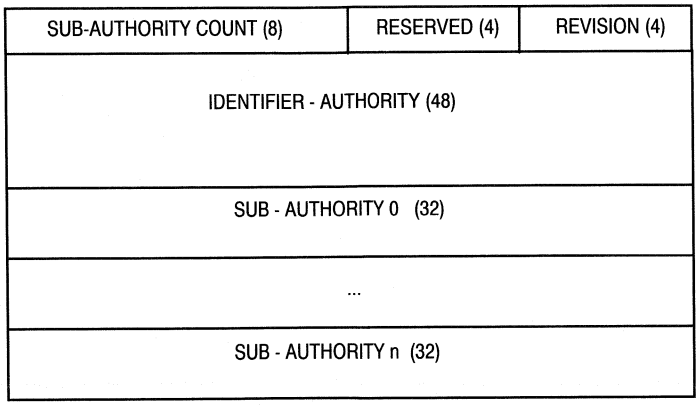


Figure 3.3 Security identifier data structure

SIDs have two levels of hierarchy—the domain name and the user or group name. Rather than deal with the unwieldy SID representation, SIDs are translated to the human-readable form DomainName\UserName or DomainName\GroupName. One of the benefits of using SIDs to represent user IDs is that users can change their name (for example, when they get married), but their security profile will remain constant without any administrator intervention apart from changing the user's name in the account database.

Another way of looking at sub-authorities is to say that they are a *relative identifier* (RID) to the authority that generated it. For example, S-1-5-4 is a revision-1 SID. "S-1-5" is the Windows NT identifier authority, and "4" is a RID of "S-1-5," which represents a well-known SID (defined in a later section) for an interactive logon session.

Under the covers, all security checks are carried out against the SID, not the username. There is one point to note if you should delete an account. Consider this example:

Suppose that the name Fred Smith appears in several access control lists on protected objects. You delete his account, and then you decide to add Fred Smith to the account database again. The new Fred Smith account does not have the same access to objects that the old Fred Smith account had, because this is a completely new SID. The only way for Fred Smith to get access to the objects is for the administrator to do it manually.

Well-Known SIDs

On a Windows NT system there are some predefined or "well known" SIDs. A *well-known SID* is one whose value is constant across all Windows NT systems. Table 3.3 lists the well-known SIDs for Windows NT.

Table 3.3 Windows NT well-known SIDs

SID value	SID name	Represents
S-1-5	Windows NT Authority	This identifier authority produces SIDs that are not universal; they are meaningful only on Windows NT installations.
S-1-5-1	Dialup Logon	A group identifier that represents users who logon using a modem.
S-1-5-2	Network Logon	A group identifier that represents users who logon across the network. The Windows NT user interface represents this as the group called NETWORK.
S-1-5-4	Interactive Logon	A group identifier that represents users who logon for interactive operation. The Windows NT user interface represents this as the group called INTERACTIVE.
S-1-5-6	Security Service	Represents an account that is authorized to perform security services.
S-1-5-12	Local System	Represents a special account that may be used by Windows NT services.

In addition to well-known SIDs for Windows NT, there are some well-known SIDs which are universal across all types of system. Table 3.4 lists these.

Table 3.4 Universal well-known SIDs

SID value	SID name	Represents
S-1-0-0	Null SID	Identifies a group that includes no members. This is often used when a SID value is not known.
S-1-1-0	World	Identifies a group that includes all users on all systems. The Windows NT user interface represents this as the group called Everyone.
S-1-2-0	Local	Identifies users who logon to terminals that are locally (that is, physically) connected to the system.
S-1-3-0	Creator Owner ID	Identifies a security identifier that should be replaced by the security identifier of the creator of a new object. This SID is used in inheritable ACLs.
S-1-3-1	Creator Group ID	Identifies a security identifier that should be replaced by the group SID of the creator of a new object. This is used in inheritable ACLs.

Account Management

Windows NT servers support the idea of managing accounts by domains, whereby you can group a set of servers together in a logical group. A *domain* is an

administrative realm with a specific domain name and including one or more servers and workstations. Within the domain, you have one server that is nominated to be the Primary Domain Controller (PDC). The PDC holds the master user account database.

All other servers within the domain are Backup Domain Controllers (BDCs) and maintain a replicant of the PDC user account database. All servers within the domain function as logon servers. However, if you need to add a new user account to the domain, the account must be added to the PDC. This model is good because you can have dozens of servers, and you only need one account in the entire domain. It does have some limitations, however. If you have multiple domains, you would need multiple accounts, one for each domain.

Windows NT takes the single domain model a step further by allowing you to have trusted domains. This new model allows you to have a single account across any number of domains by permitting domains to trust each other.

Trust relationships are not always bidirectional. For example, you could have domain 1 trusting domain 2, but domain 2 not trusting domain 1. It is up to the administrator to set up the trust relationships.

What does the trust relationship give you? Let's say that in each domain there are three users—user1, user2, and user3. Once the trust relationship is set up, in domain2, the administrator can take the domain1\user1 account and give it permission to use objects within his or her own domain. (This is exactly how the accounts are represented within the Permissions Editor. That is, if an account belongs in another domain, you see the account name preceded by the domain name to which the account belongs.)

Like user accounts, a domain is represented not only by a human-readable name, but it is represented to the system by a SID. The SIDs representing user and group accounts are the same as the SID of the domain they are in, except that they have one extra RID. For example, S-1-5-21-234-345-539-1004 represents the SID of a user where S-1-5-21-234-345-539 represents the Domain SID. 1004 is the RID of the user within that security domain.

Privileges (Rights)

Privileges are used to gain access to objects or services that normal discretionary access control does not provide. For example, in order to set the system time, a subject must be granted the privilege to do so. The Windows NT user interface refers to privileges as "user rights."

A privilege on Windows NT is represented by something called a *locally unique identifier* (LUID). A LUID is guaranteed to be unique on only the system on which it was generated. Furthermore, this uniqueness is guaranteed only until the

next time the system is rebooted. That is, LUIDs are not unique across system boots.

Windows NT represents privileges in the following ways:

Table 3.5 Privilege representation

Privilege representation	Value
Privilege Program Name	SeSystemtimePrivilege
User Interface Display Name	Change the system time
Internal Representation	12

Well-Known Privilege Values

On a Windows NT system there are some predefined or well-known privilege value. A *well-known privilege value* is a privilege whose value is constant. Table 3.6 provides some examples:

Table 3.6 Examples of well-known privileges

Privilege program name	User interface name
SeTcbPrivilege	Act as part of the operating system.
SeBackupPrivilege	Backup files and directories.
SeSystemtimePrivilege	Change the system time.
SeChangeNotifyPrivilege	Bypass traverse checking. Note that this privilege is enabled by default for all users, it is required to avoid notifications of changes to files or directories. Disabling this privilege (and therefore causing system notification) impairs system performance.
SeCreatePagefilePrivilege	Create a page file.
SeCreateTokenPrivilege	Create a token object.
SeCreatePermanentPrivilege	Create permanent shared objects.
SeDebugPrivilege	Debug programs.
SeRemoteShutdownPrivilege	Force shutdown from a remote system.
SeIncreaseQuotaPrivilege	Increase quotas.
SeIncreaseBasePriorityPrivilege	Increase scheduling priority.
SeLoadDriverPrivilege	Load and unload device drivers.
N/A (System access)	Local logon.
SeLockMemoryPrivilege	Lock pages in memory.

Table 3.6 Examples of well-known privileges *(continued)*

Privilege program name	User interface name
SeSystemEnvironment	Modify firmware environment.
SeSecurityPrivilege	Perform security operator functions.
N/A (System access)	Remote access.
SeRestorePrivilege	Restore files and directories from backup.
N/A (System access)	Service logon.
SeShutdownPrivilege	Shutdown the system.
SeTakeOwnershipPrivilege	Take ownership of files or other objects.
SeAuditPrivilege	Use security audit generation services.

Note In the first release of Windows NT, only well-known privileges are supported, future releases of Windows NT will allow user-defined privileges.

Identification and Authentication

Before a user is allowed to do anything on a Windows NT system, he or she must first log on to the system and be authenticated. To log on to Windows NT, a user provides a username and password. The username is used for identification and password is used for authentication.

Figure 3.6 illustrates the Windows NT logon and authentication process.

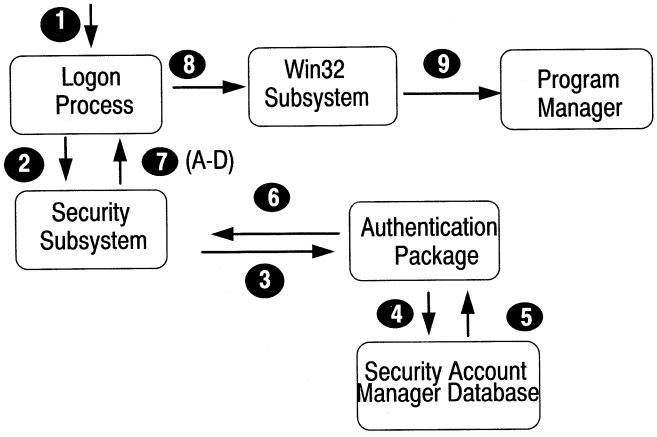


Figure 3.6 Windows NT authentication process

For Windows NT, the interactive logon process proceeds as follow:

1. The operating system monitors the keyboard for a secure attention sequence, Ctrl+Alt+Del. This secure-attention sequence is, by definition, a device-specific series of events that the user causes to indicate that he or she wants to logon to the system.
2. The Logon Process calls the Security Subsystem.
3. The Security Subsystem receives the call from the Logon Process and dispatches the call to the appropriate authentication package. For a Windows NT logon, this is the MSV1_0 authentication package.
4. The security subsystem receives the call from the logon process and dispatches the call to the appropriate authentication package.

Note Windows NT has the ability to support multiple authentication packages which are implemented as DLLs. This flexibility allows developers the opportunity to have their own custom authentication routines that meet their specific requirements. For example, a bank might augment the standard Windows NT authentication package by adding one that uses an ATM card and a personal identification number (PIN).

---

5. The authentication package calls the Security Account Manager (SAM) database to see if the account is local. If not, the requested logon is forwarded to a remote authentication package. If the account is local, the username and password are verified against those held in the SAM.
6. Upon authentication, the SAM returns the user's SID and the SIDs of any global groups to which the user belongs.
7. The authentication package creates a logon session, then passes the logon session and the SIDs associated with the user to the Security Subsystem.
8. If the logon is rejected, an error message is passed back to the Logon Process, Otherwise, the access token is created by following these steps:
   a. Add the SID representing "Everyone" and other SIDs that represent appropriate well-known global groups.
   b. Look up any local groups assigned to any of the returned user or global group SIDs.
   c. Look up any rights assigned to any of the collected SIDs. If the appropriate logon right (for example, Interactive) is not granted, delete the logon session and return an error to the logon process.
   d. Create an access token containing all these SIDs and user rights, and return it to the logon process with a success status.
9. The logon session calls the Win32 subsystem to create a process and attach the access token to the process to form a subject.

10. For an interactive Windows session, the Win32 subsystem calls the Program Manager to start the user's shell.

Access Tokens

After the authentication process, a user's shell process is tagged with an access token. An *access token* is a data structure that contains security-related information about a logged-on user. Among other things, an access token contains a set of SIDs representing who the logged-on user is, which groups the user is a member of, and which privileges are assigned to those user and group accounts.

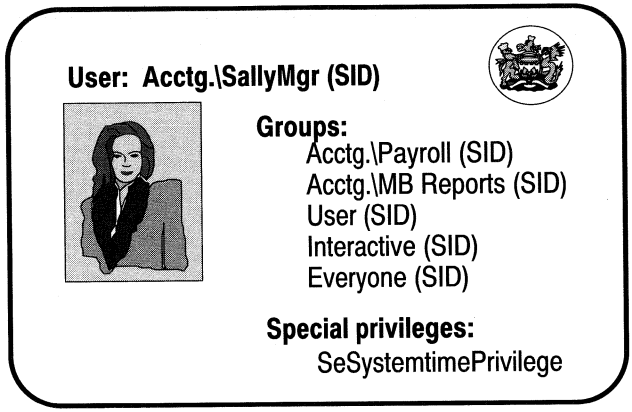


Figure 3.4 A security access token

An access token contains information about the following aspects of a process:

Table 3.7 Token attributes

Component	Function
User SID	The SID of the user account the user logged onto. All security contexts must include this value. This value is used in access validations and provides a default object ownership value. This SID is a required part of security context.
Group SIDs and attributes	The group SIDs and corresponding attributes of groups to which the user is assigned membership. Security contexts can contain one or more groups. If present, any enabled groups are used for access validations.
Privileges and attributes	The names of the privileges assigned to the user and their corresponding attributes. A security context can have zero or more privileges.
An owner SID	Points to the SID designated as the owner of any objects created on behalf of the user represented by this token. The SID must be one of the user or group SIDs already in the token.
Default discretionary access control	Optionally, the user can establish a default discretionary ACL to be part of his or her security context. (The use of this information is outside the scope of this document. Note that in the first release of Windows NT, this feature is not used by any exposed functionality.)
Default primary group SID	Optionally, a default primary group SID can be part of the security context. The user does not have to be a member of this group, but that is typically the case. If not present, a null SID is used. This attribute is used by the POSIX environmental subsystem.
Authentication ID	Assigned when the user logs on. This ID is intended to uniquely relate any actions of that user back to the logon session.
Source of access token	Identifies which protected server generated the token. This is used to distinguish between sources such as Session Manager, LAN Manager, and RPC Server.
Token type	Identifies whether this is the primary token or an impersonation token.
Token statistics	Contains various information about a token.
Token impersonation level	Specifies token impersonation level. Security impersonation levels govern the degree to which a server process can act on behalf of the a client process.

Subjects

When an access token is assigned to a program acting on the user's behalf, the combination is known as a *subject.*

To accommodate Windows NT's client/server model, there are two classes of subject within the Windows NT security architecture—simple subjects and server subjects.

A *simple subject* is a process which was assigned a security context when the corresponding user logged on. It is not acting in the capacity of a protected server, which may have other subjects as clients.

A *server subject* is one implemented as a protected server, and does have other subjects as clients. In this role, a server subject typically has the security context of those clients available for use when acting on their behalf.

In general, when a subject makes a call to an object service through a protected subsystem, the subject's token is used within the service to determine who made the call and to decide whether the caller has sufficient rights to perform the requested operation.

Impersonation

Impersonation is the ability of a process to take on the security attributes of another process. Typically, a server process impersonates a client process to complete a task involving objects that the client does not normally have access to, this done to ensure that access to secure objects is carried out in the correct security context.

All of the components of a user's security context are inherited throughout the user's process tree as subprocesses are created and destroyed. It is an important part of the audit log information, relating actions back to a particular user and logon session. This is a required part of security context.

Consider the scenario depicted in Figure 3.5.

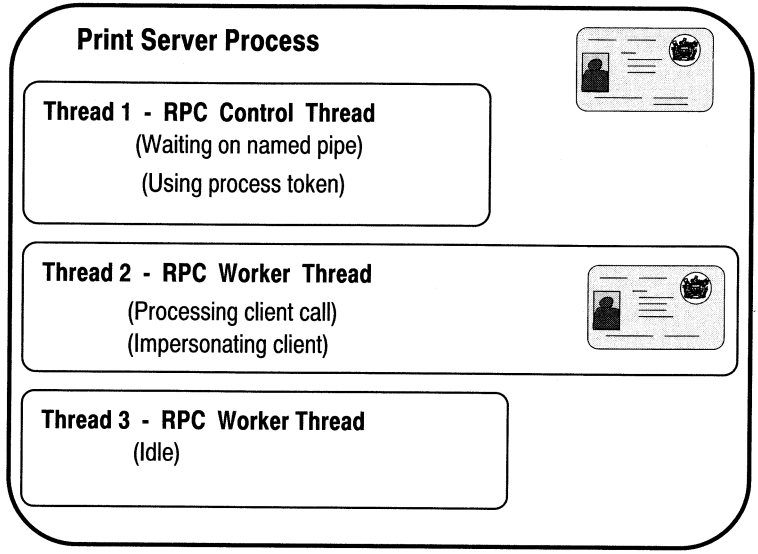


Figure 3.5 Sever subject security context

In this case, some client has called a print server process that is running on a Windows NT computer (locally or remote) to access one of the print server's objects (such as a print queue).

The first thread in the process is a control thread that is waiting on a named pipe for RPC calls to come in. It is not impersonating anyone, so any access validation to which Thread 1 is subjected will be carried out against the process's primary token.

The second thread in the process is currently handling a call from a client. This thread handles the client's call by temporarily using the client's access token to run in the security context of that client. While impersonating the client, any access validation to which Thread 2 is subjected is carried out in security context of the client.

The third thread in this scenario is just an idle worker thread that and is not impersonating anyone.

Security Context

A critical part of a subject is its *security context*. This is the information that indicates which user the subject represents and controls what access the subject has to objects or system services. For example, subject security context includes a user ID and group IDs that are used within an access control list to control access to objects. (For more information, see the "Access Control List" section later in this chapter.)

Just as there are two types of subjects, there are two corresponding forms of subject security context—simple subject security context and server subject security context. A *simple subject security context* is a thread with no token of its own, but which uses the token of its process. A *server subject security context* is a thread that has its own token (representing its client) in addition to the token of its process.

Thus far, this chapter has described security elements—such as objects, accounts, and access tokens—that provide for overall system security. As we mentioned earlier, there is a second type of security provided by the Windows NT security model called discretionary access control. The remainder of this chapter describes this type of security.

Discretionary Access Control

There are several discretionary access control architectures that could be used to implement discretionary access control. Windows NT uses an access control list model for two reasons:

- To provide discretionary access control for objects
- To represent mandatory audit control for objects

The next section describes how the access control list architecture is implemented within Windows NT.

Access Control List

Access control lists (ACLs) are lists of users and/or groups, with each of their specific associated permissions. ACLs allow any particular user or group to be granted or denied access to a particular protected object. Figure 3.7 illustrates an ACL data structure.

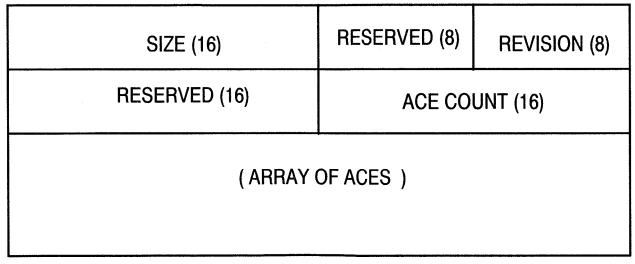


Figure 3.7 ACL data structure

Access control lists are used for two purposes. *Discretionary ACLs* (DACLs) are used to specify discretionary protection information. *System security ACLs* are used to specify system-level security information and represent mandatory audit control for objects.

Each entry in an ACL is known as an access control entry (ACE). The ACE data structure is illustrated below.

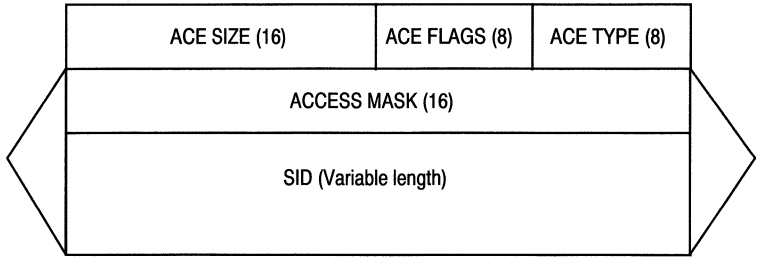


Figure 3.8 ACE data structure

There are currently three defined ACE types; two for discretionary access control and one for system security.

Table 3.8 ACE types

Ace type	Used for
AccessAllowed	Discretionary access control ACE used to grant access to a user or group of users. The user or group of users is identified in the ACE by a SID. The accesses to be granted by the ACE are also included in the ACE.
AccessDenied	Discretionary access control ACE used to explicitly deny access to a user or group of users. The user or group of users is identified in the ACE by a SID. The accesses to be denied by the ACE are also included in the ACE.
SystemAudit	System security ACE used to keep a log of security significant events, such as who accessed which files. The SystemAudit ACE is used to generate security audit messages, and cause them to be entered into a system audit log for later processing.

Note There is an important distinction between a DACL that is empty (one which has no ACEs in it) and an object without any DACL. In the case of an empty DACL, no accesses are explicitly granted, so access is implicitly denied. In the case of not having an ACL at all, on the other hand, there is no protection assigned to the object, so any access request is granted.

Access Control Inheritance

When an object is created, one or more ACLs may need to be assigned to it.

Windows NT's NTFS file system (described in Chapter 5, "New Technology File System") supports the inheritance of ACLs from directory objects to file objects which are created within the directory. The inheritance design is intended to allow access control information on a container object to be thought of, and presented to the user, as three separate ACLs:

- The ACL pertaining to the container object itself (called the Effective ACL); for example, a directory
- An ACL to be inherited by non-container objects (called the Object Inherit ACL); for example, files which inherit from a directory
- An ACL to be inherited by container objects (called the Container Inherit ACL); for example, subdirectories which inherit from a directory

From this perspective, when a new non-container object is created, the parent container's Object Inherit ACL is applied to the new object. This notion is illustrated in Figure 3.9 with a file inheriting from its parent directory.

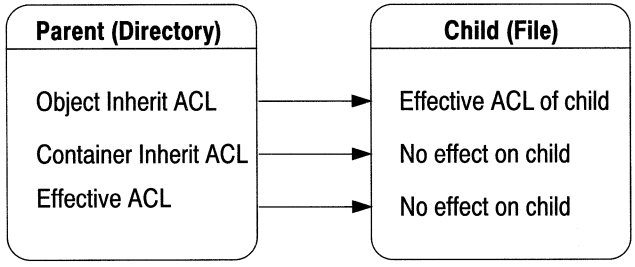


Figure 3.9 ACL inheritance to a non-container object

When a new sub-container object is created, the parent container's Container Inherit ACL becomes both the Effective and Container Inherit ACL for the sub-container. Also, the Object Inherit ACL of the parent container propagates to the new sub-container as its Object Inherit ACL. This notion is illustrated in Figure 3.10 with a subdirectory inheriting from its parent directory.

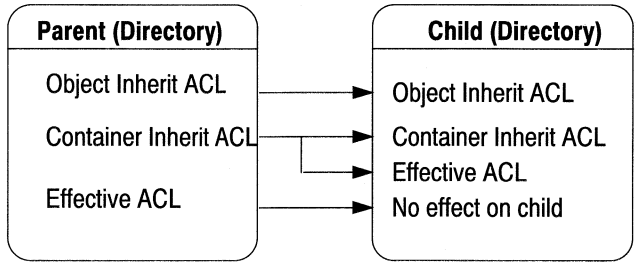


Figure 3.10 ACL inheritance to a container object

While we can think of ACL information in terms of the three separate ACLs listed above, this information is actually stored as a single ACL. The ACL design supports inheritance specification defined for each ACE within an ACL. Each ACE can be marked for no inheritance, for inheritance by sub-containers, for inheritance by non-container objects, or for inheritance by both.

Consider the scenario depicted in Figure 3.11, where the permissions for the directory \NTResKit are as follows:

```
Administrators      Full Access (All)(All)
Mgrs                Read (RX)(RX)
```

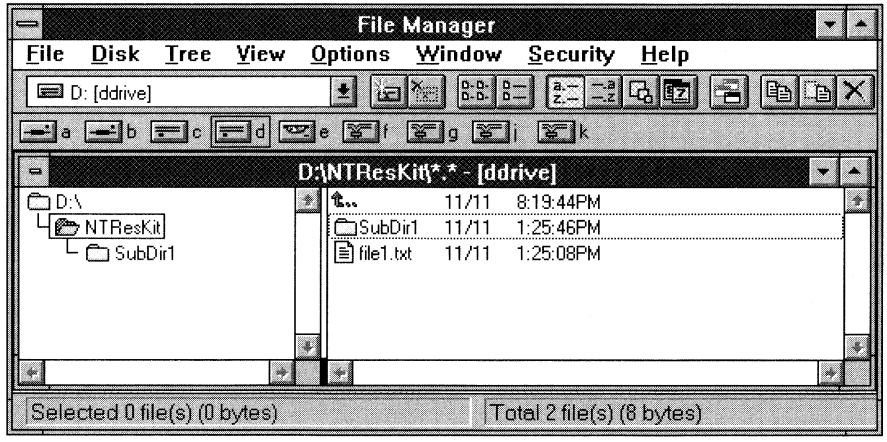


Figure 3.11 An NTFS partition

In this case, when the directory \SubDir1 is created, it inherits the permissions from its parent directory \NTResKit. When the file FILE1.TXT is created, it inherits the effective ACL from its parent directory \NTResKit, and thus has the following permissions:

```
Administrators      Full Access (All)
Mgrs                Read (RX)
```

Access Types and Access Masks

Within each ACE data structure are two key components—an access type and an access mask.

An *access type* defines a particular set of abilities that can be granted or denied to a process when it attempts to use an object. For example, if a process attempts to write data to a secured file but does not have FILE_WRITE_DATA access to the file, the process would be denied that operation.

An *access mask* defines what access that a user or a group has to an object. Access masks are also used in open calls to specify the desired access to an object.

As illustrated in Figure 3.12, an access mask data structure can contain the following categories of information:

- Specific types
- Standard types
- Generic types (including some special types)

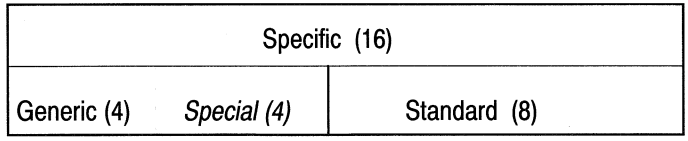


Figure 3.12 Access mask data structure

Access Mask Specific Types

Specific types contain the access mask that is specific to the object type associated with the mask. Specific access types, or more properly, object type-specific access types, are applicable to only a single object type. These provide a detailed level of protection when needed.

The definition of an object must include the definition of the object type-specific access types. Each object type can have up to 16 specific access types.

The specific access types are collectively referred to as the *specific access mask.*

To illustrate what specific access types are, consider the Windows NT file object, which has the following specific access types defined:

```
FILE_READ_DATA                  FILE_WRITE_DATA
FILE_APPEND_DATA                FILE_READ_EA
FILE_WRITE_EA                   FILE_EXECUTE
FILE_READ_ATTRIBUTES            FILE_WRITE_ATTRIBUTES
```

The definition and meaning of object-type specific access types is entirely up to the designer of the object type and is outside the scope of this document.

Access Mask Standard Types

Standard types apply to all objects. Figure 3.14 illustrates the access mask standard types.

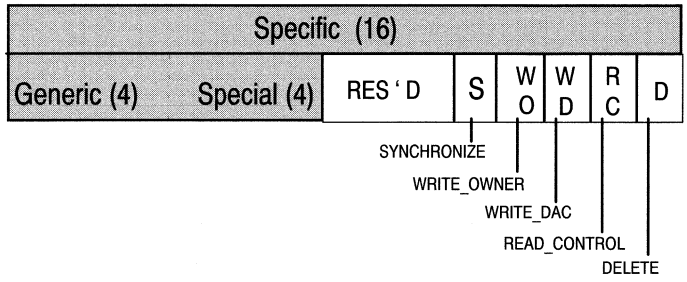


Figure 3.14 Access mask standard types data structure

This structure includes these constants:

- RESERVED.
- SYNCHRONIZE. Used to synchronize access (allows a process to wait for an object)
- WRITE_OWNER. Used to assign write access owner.
- WRITE_DAC. Used to assign write access to the discretionary ACL.
- READ_CONTROL. Used to assign read access to the security descriptor.
- DELETE. Used to assign delete access.

Access Mask Generic Types

Generic types are mapped to specific and standard types when access to an object is requested. (Included with generic types are some additional *special types*, which are described in the next section.) Figure 3.15 illustrates the data structure for access mask generic types.

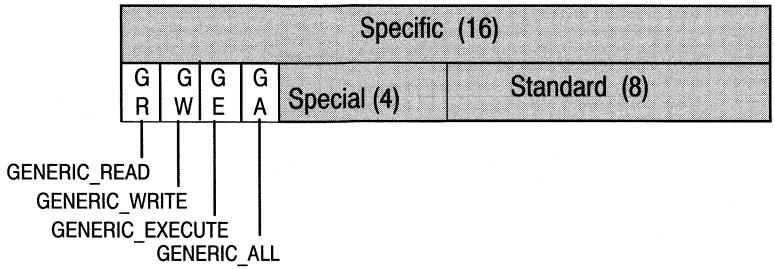


Figure 3.13 Access mask generic types data structure

Access mask generic types data structure includes these constants:

- GENERIC_READ. Used to assign read access.
- GENERIC_WRITE. Used to assign write access.
- GENERIC_EXECUTE. Used to assign execute access.
- GENERIC_ALL. Used to assign read, write, and execute access.

Consider once again the example of the file object type. Suppose that a file object type includes the generic accesses FILE_GENERIC_READ, FILE_GENERIC_WRITE, and FILE_GENERIC_EXECUTE. These accesses imply certain standard and specific accesses, as shown in Table 3.14:

Table 3.14 Implied accesses for a file object

Generic access	Implied standard and specific accesses
FILE_GENERIC_READ	(STANDARD_RIGHTS_READ|FILE_READ_DATA| |FILE_READ_ATTRIBUTES |FILE_READ_EA| |SYNCHRONIZE)
FILE_GENERIC_WRITE	(STANDARD_RIGHTS_WRITE|FILE_WRITE_DATA| |FILE_WRITE_ATTRIBUTES|FILE_WRITE_EA| |FILE_APPEND_DATA|SYNCHRONIZE)
FILE_GENERIC_ EXECUTE	(STANDARD_RIGHTS_EXECUTE| |FILE_READ_ATTRIBUTES|FILE_EXECUTE| |SYNCHRONIZE)

Note STANDARD_RIGHTS_READ, STANDARD_RIGHTS_WRITE, and STANDARD_RIGHTS_EXECUTE are all manifest constants for READ_CONTROL.

To illustrate how generic access types can be used, suppose a user wants to establish a default ACL to be assigned to new objects created on the user's behalf. It would not be desirable to use specific access types in such a default ACL, since that would require a default ACL for each type of object that might be created. Instead, a single ACL that uses generic access types could be established. Each time a new object is created, a copy of the default ACL is applied to the new object. However, the generic access types are mapped to the specific access types corresponding to the type of object being created, then the generics access types are cleared.

Notice that the sum of all accesses granted by GENERIC_READ, GENERIC_WRITE, and GENERIC_EXECUTE do not need to cover all accesses supported by an object. In the example of a file inheriting access from its parent directory, the file object access types WRITE_DAC and WRITE_OWNER access types are not granted by GENERIC_READ, GENERIC_WRITE, or GENERIC_EXECUTE. GENERIC_ALL is used for this purpose; that is, it maps to all defined access types for the object type.

GENERIC_ALL is particularly useful when setting protection to deny access to an object. In this case, someone assigning protection typically wants to be sure no access is allowed to the person being denied access. Denying the person GENERIC_READ, GENERIC_WRITE, and GENERIC_EXECUTE isn't sufficient to guarantee that all access types are explicitly denied. However, by denying GENERIC_ALL, the person will be explicitly denied all access types.

The following table defines the standard generic access type mappings to standard access types:

Table 3.15 Generic access to standard access mappings

Generic access	Implied standard access
GENERIC_READ	READ_CONTROL
GENERIC_WRITE	READ_CONTROL
GENERIC_EXECUTE	READ_CONTROL

In addition to these, GENERIC_EXECUTE typically maps to SYNCHRONIZE access for object types that support synchronization.

Access Mask Special Types

As shown in Figure 3.13, Windows NT uses two access mask special types—MAXIMUM_ALLOWED_ACCESS and ACCESS_SYSTEM_SECURITY.

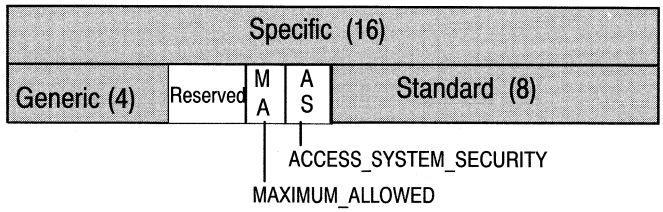


Figure 3.14 Access mask special types data structure

Access mask special types data structure includes these constants:

- RESERVED.
- MAXIMUM_ALLOWED. Used to specific maximum allowed access to an object.
- ACCESS_SYSTEM_SECURITY. Used to assign access to system ACL (must have a specific privilege).

MAXIMUM_ALLOWED_ACCESS is used by programs wanting to open an object and be granted the maximum access allowed by the protection assigned to that object. This access type is ignored if encountered in an access mask assigned as protection of an object. That is, it does not make sense to grant or deny a user or group MAXIMUM_ALLOWED_ACCESS to an object in an ACL.

Generally, asking for maximum allowed access should be avoided. This is in keeping with the principle of least privileges; that is, if a user needs only read access, then the user should be assigned only read access.

In many object-oriented systems, it is typically necessary to declare what access types are being sought to a particular object. To support this model when it comes to accessing system security information, a special access type ACCESS_SYSTEM_SECURITY was defined. This access type provides both read and write access to all system security information. Note that this access type is not used in Discretionary ACLs. That is, the system ACL of an object cannot be controlled by discretionary access control.

If this access type is specified for an object under Windows NT at object access time, a privilege test is made to determine whether the caller should be allowed to access the system security information.

The following table highlights how the accesses from the access mask of a file object manifest themselves in the Windows NT user interface.

	Read	Write	Execute	Delete	Chg Permission	Take Ownership
Specific Rights:						
FILE_READ_DATA	X					
FILE_READ_ATTRIBUTES	X		X			
FILE_READ_EA	X					
FILE_WRITE_DATA		X				
FILE_WRITE_ATTRIBUTES		X				
FILE_WRITE_EA		X				
FILE_APPEND_DATA		X				
FILE_EXECUTE			X			
FILE_ALL_ACCESS						
Standard Types:						
SYNCHRONIZE	X	X	X			
WRITE_OWNER						
WRITE_DAC						
READ_CONTROL						
DELETE						
Generic Types:						
GENERIC_READ	X					
GENERIC_WRITE		X				
GENERIC_EXECUTE			X			
GENERIC_ALL						
MAXIMUM_ALLOWED						
ACCESS_SYSTEM_SECURITY						

The following section details how Windows NT applies access control data structures to objects.

Object Security

A *security descriptor* (SD) is a data structure that contains the security information associated with an object. This information is represented by ACLs and SIDs, two structures introduced earlier in this chapter. An example is shown in Figure 3.15.

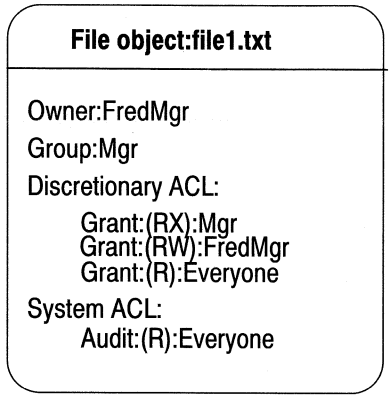


Figure 3.15 Example of a security descriptor

One of the primary goals of the Windows NT object model is to define a standard set of security semantics that applies to instances of objects, regardless of the type or class of object. This security information is found in an object's security descriptor and includes the following:

- An Owner ID. This SID indicates which user or group account owns the object. The owner of an object has the inherent right to change the protection of the object.
- A Group ID. This SID indicates which group the object is associated with. The Group ID is required to support security requirements of POSIX. The group ID is ignored for Win32 applications.
- A Discretionary ACL. This data structure is controlled by the owner of the object. It identifies who can access the object and who can't.
- A System ACL. This data structure is controlled by the security administrators and is used to control audit message generation.

You can use security descriptor structure to set and query an object's security attributes.

Figure 3.16 shows the data structure for a security descriptor. In addition to fields for SIDs and ACLs, security descriptors contain a revision field to allow future extensions to the information.

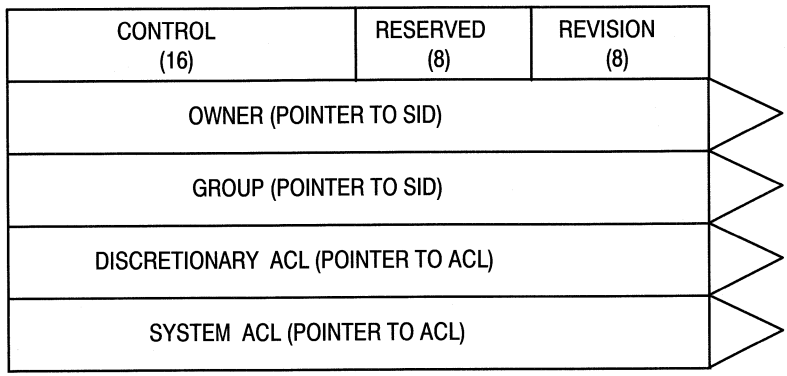


Figure 3.16 Security descriptor data structure

All named objects, plus all named and unnamed process, thread, and token objects have a security descriptor associated with them.

Absolute and Self-Relative Security Descriptor Formats

Two formats are used for representing security descriptor data structures and their components (ACLs, SIDs, and so on). These two formats are referred to as absolute and self-relative.

An *absolute* format security descriptor is one in which the main body of the security descriptor contains pointers to the other components of the security descriptor. This format allows each component of the security descriptor to be separately allocated. This is useful, for example, at application interfaces, where defaults for owner ID, group ID, and discretionary access control may already be available. In this case, all that needs to be done is to initialize a security descriptor body to point to all the existing components.

A *self-relative* format security descriptor is one in which the security descriptor data structure and all the associated components of the security descriptor are stored in a single contiguous block of memory. Furthermore, the sub components of the security descriptor are pointed to using offsets from the beginning of the security descriptor data structure rather than memory addresses (as in the absolute format).

This format of a security descriptor is very useful for storage on secondary media, or transmission in communications protocols, where absolute pointers become invalid. The offsets in this format become valid when the security descriptor is later retrieved.

Creating Object Security Descriptors

When creating a Windows NT object, a security descriptor is used to provide explicit or default security information to be applied to the new object. This can include information about the intended owner, primary group, discretionary ACL, and system ACL to be assigned to the new object. (There may be other mechanisms as well, such as default values in subject security context.)

The values assigned for the security descriptor fields takes the following information into account:

- The user creating the object
- Any explicit values provided by the creator for owner, primary group, discretionary ACL, and system ACL
- Any default values provided by the creator for owner, primary group, discretionary ACL, and system ACL
- Any inheritable discretionary ACL information assigned to the parent container in which the new object is being created
- Any inheritable system ACL information assigned to the parent container in which the new object is being created

Given this information, the values assigned to a new object are as follows.

Owner:

- If an explicit value is provided via a security descriptor, it is assigned as the new object's owner.
- If a value is not provided via security descriptor, then the creator's subject security context is checked for a default value (if the subject context includes default owner information). If found, that value is assigned as the new object's owner.
- If neither an explicit value nor a default value was provided, then the creator's user ID (taken from the subject's security context) is assigned.
- The resulting value is checked against the creator's security context. If the caller is not authorized to assign the SID as an owner, then the object creation fails.

Note Typically, restrictions on who may be assigned as the owner of an object are made upon the subject. If the subject attempts to assign a value without authorization, an error occurs and object creation fails.

Primary Group:

- If an explicit or default value is provided via security descriptor, it is assigned as the new object's primary group.
- If a value was not provided via security descriptor, then the primary group in the creator's subject security context is assigned.

Discretionary ACL:

- If an explicit ACL is provided via security descriptor, it is assigned as the new object's discretionary ACL.
- If an explicit value is not provided, the parent container's discretionary ACL is checked for any inheritable ACEs. If found, these are assigned as the new object's discretionary ACL.
- If neither an explicit ACL nor an inheritable ACL is available, then a default discretionary ACL is sought. First, the system checks for an explicitly-provided default security descriptor. If none is found there, then the caller's subject security context is checked for a default.
- If none of these ACLs are found, then the new object is assigned no discretionary ACL, granting unconditional access to everyone.

System ACL:

- If an explicit ACL is provided, it is assigned as the new object's system ACL. If the creator does not have the security user right, then the new object creation fails.
- The parent container's system ACL is checked for any inheritable ACEs. If found, these are assigned as the new object's system ACL.
- If neither an explicit ACL nor an inheritable ACL is available, then a default system ACL is searched for in the provided security descriptor. If found, the user must have the privilege to assign it.

Note Assignment of a system ACL by a mechanism other than inheritance is a privileged operation and is restricted to security administrators.

Access Determination

When an ACL is used to provide discretionary access control, the following information is needed to determine if access is to be granted or denied:

- The security context of the subject attempting the access
- The accesses requested by the subject (called the *desired accesses*)
- Any generic access bits, which must be mapped to non-generic access types

- The SID assigned as owner of the object
- The discretionary ACL associated with the object

Given this information, the following algorithm is applied:

- If AccessSystemSecurity is requested, check for adequate privilege.
- Reject the request if the subject has insufficient privileges.
- Otherwise, clear the AccessSystemSecurity flag in the desired mask.
- If there is no ACL, then grant access.

A desired access mask is initialized to indicate the desired accesses. All generic access types are mapped to standard and specific access types. The ACEs in the ACL are evaluated as follows:

- The SIDs in the ACE are compared to the set of SIDs in the subject's context.
- If a match is not found, the ACE is skipped.
- Further processing is based upon the type of the ACE:
  - AccessAllowed ACE type. The accesses in the ACE are cleared from the desired access mask. If the desired access mask becomes zero (all bits cleared), further processing is not necessary and access is granted. Otherwise, processing continues with the next ACE.
  - AccessDenied ACE type. The accesses in the ACEs access mask are compared against the desired access mask. If there are any accesses in both masks, further processing is not necessary and access is denied. Otherwise, processing continues with the next requested ACE. If the end of the ACL is reached and the desired access mask is still non-zero, access is implicitly denied. If access is denied, check to see if the original desired access mask contained only a READ_CONTROL and/or WRITE_DAC. If so, see if the requester is the owner of the object. If so, grant access.

The following several sections illustrate how discretionary access is determined in specific situations.

Access Determination Example 1

The user FredMgr is trying to gain Read and Write access to the file object that has the discretionary ACL as show in Figure 3.17. FredMgr security token indicates that he is a member of the groups Users, Mgrs, and Everyone.

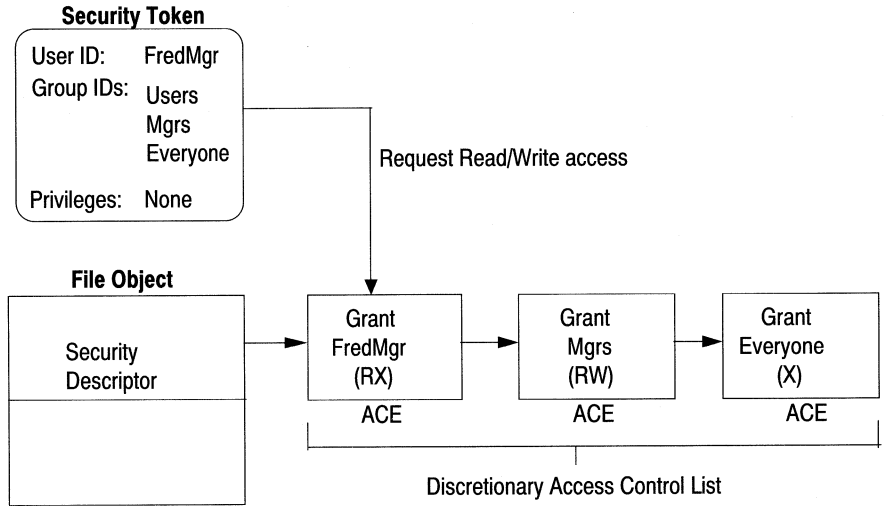


Figure 3.17 Illustration of Access Determination Example 1

In Example 1, the ACL is evaluated as follows: When the first ACE is evaluated the Read bit in the desired access mask is cleared, and processing moves on to the next ACE in the ACL. When the second ACE is evaluated the Write bit is cleared, and processing of the ACL stops even though there is another ACE in the ACL. Processing stops and access is granted because all the bits in the desired access mask have been cleared.

Access Determination Example 2

The user FredMgr is trying to gain Read and Write access to the file object that has the discretionary ACL as show in Figure 3.18. FredMgr security token indicates that he is a member of the groups Users and Mgrs.

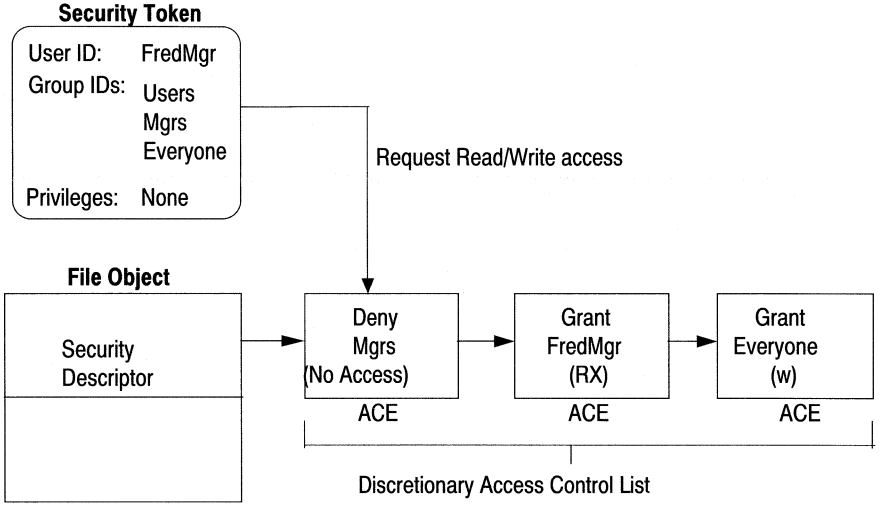


Figure 3.18 Illustration of Access Determination Example 2

Note If you use the Windows NT Permissions Editor from File Manger, deny access ACEs are *always* ordered first in the ACL.

In Example 2, the ACL is evaluated as follows: When the first ACE is evaluated, processing of the ACL will stop, and FredMgr will be denied access to the file because FredMgr is a member of Mgrs. This will happen even though the second, and third ACE would have given FredMgr Read/Write access.

Access Determination Example 3

The user FredMgr is trying to gain Read and Write access to the file object that he owns which has the discretionary ACL as show in Figure 3.19. FredMgr security token indicates that he is a member of the groups Users, Mgrs, and Everyone.

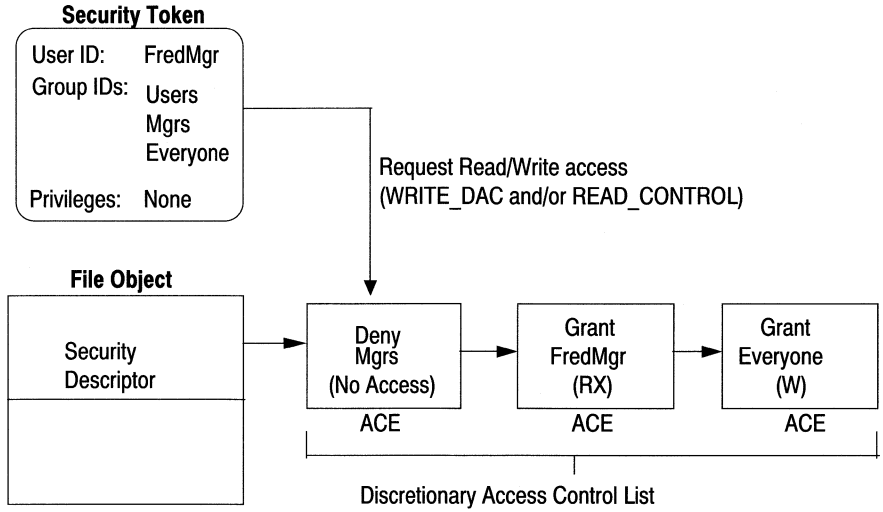


Figure 3.19 Illustration of Access Determination Example 3

In Example 3, the ACL is evaluated as follows: When the first ACE is evaluated, processing of the ACL will stop and FredMgr will be denied access to the file because FredMgr is a member of Mgrs. This will happen even though the second and third ACE would have given FredMgr Read/Write access. However, after failing to gain access via the DACL, it is noticed that FredMgr is the owner of the object. Because of this, he is granted READ_CONTROL and WRITE_DAC automatically. Since this is all the access he is asking for, his request will be granted since this is all the access he is asking for. If FredMgr had asked for any other access in addition to READ_CONTROL and WRITE_DAC, the request would have been denied even though Fred is the owner of the object.

Access Determination Example 4

The user FredMgr is trying to gain Read and Write access to the file object that has the discretionary ACL as show in Figure 3.20. FredMgr security token indicates that he is a member of the groups Users and Mgrs.

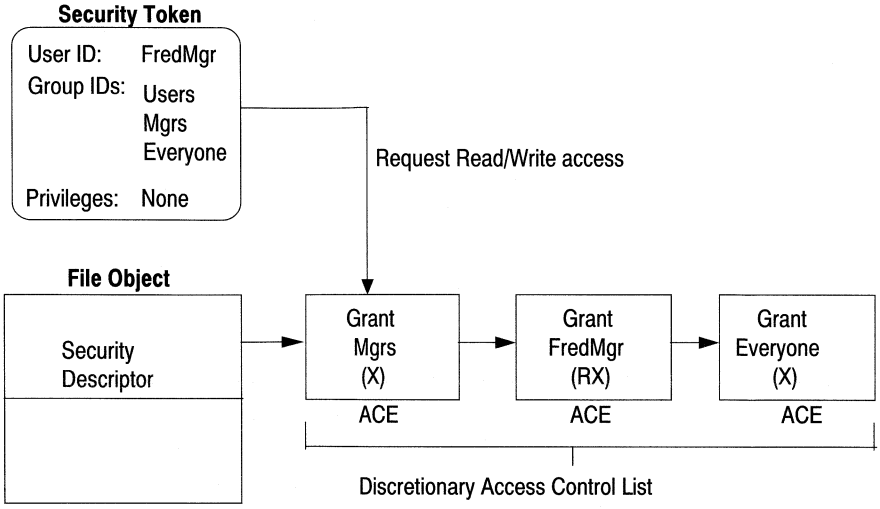


Figure 3.20 Illustration of Access Determination Example 4

In Example 4, the ACL is evaluated as follows: When the first ACE is evaluated it does not grant any accesses that FredMgr is asking for, so processing continues with the next ACE. When the second ACE is evaluated the Read bit in the desired access mask is cleared. After the third ACE has been evaluated, all the bits in the desired access mask have not been cleared, access to the file is denied.

Access Determination Example 5

The user FredMgr is trying to gain Read and Write access to the file object that has the discretionary ACL as show in Figure 3.21. FredMgr security token indicates that he is a member of the groups Users, SnrMgrs, Mgrs, and Everyone.

In this case the following "Maximum Allowed Algorithm" is used:

1. If there is no ACL, then grant access.
2. A desired access mask is initialized to indicate the desired accesses. All generic access types are mapped to standard and specific access types.
3. The ACEs in the ACL are evaluated.
4. A granted and a denied access mask are initialized to zero.
5. The SIDs in the ACE are compared to the set of SIDs in the subject's context. If a match is not found, the ACE is skipped.

6. Further processing is based upon the type of the ACE:
   - For an AccessAllowed ACE type, the granted access mask becomes the accesses that are granted by the result of the following logic:

     ```
     Granted access mask = Granted access mask | (ACE mask & !Denied
     access mask )
     ```

   - For an AccessDenied ACE type Case AccessDenied, the denied access mask becomes the accesses that are denied as a result of the following logic:

     ```
     Denied access mask = Denied access mask | (ACE mask & !Granted
     access mask)
     ```

7. If, at the end of the ACL, the desired access mask is still non-zero, then access is implicitly denied.
8. If access is denied, check to see if the original desired access mask only contained a READ_CONTROL and/or WRITE_DAC. If so, see if the requester is the owner of the object. If so, grant access.

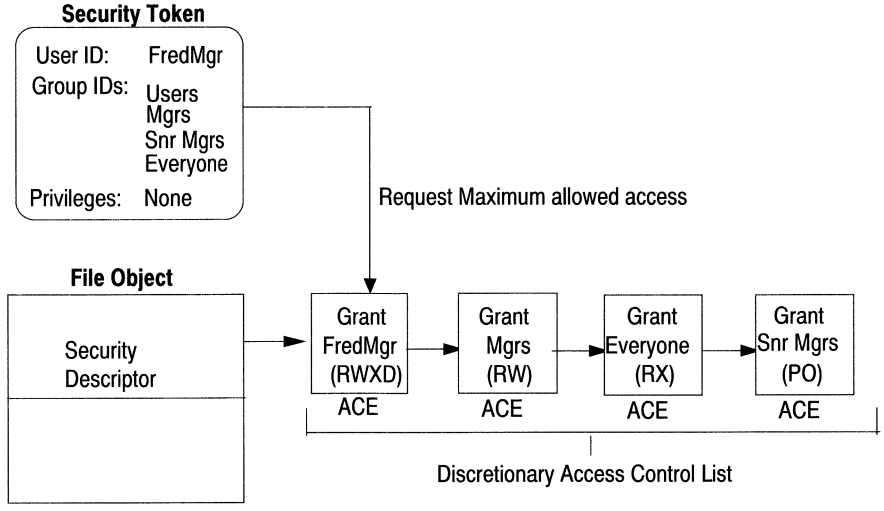


Figure 3.21 Illustration of Access Determination Example 5

The ACL in Example 5 is evaluated as follows:

1. Clear the granted and the denied access mask.
2. When the first ACE is evaluated a match is found the granted access mask has the following bits set: Read, Write, Execute, and Delete.
3. The granted access mask is logically OR'd to the denied access mask and processing continues with the second ACE.

4. A match is found when the second ACE is evaluated but it does not set any additional bits in the granted access mask and processing moves on to the next ACE.
5. A match is found when the third ACE is evaluated and but this too does set any additional bits in the granted access mask and processing moves on to the next ACE.
6. When the fourth and final ACE is evaluated a match is found and the change permission and take ownership bits are set of the granted access mask are set.
7. The granted access is logically OR'd with the denied access mask.
8. At the end of processing the entire ACL, FredMgr has been granted the following permissions: Read, Write, Execute, Delete, Change Permissions and Take Ownership.

Important All of the preceding examples demonstrate discretionary access determination for file and directory permissions that are applied through the Windows NT Permissions Editor (found in File Manager) either directly or by inheritance. If you use a custom application that sets and changes permissions on files and directories, the Windows NT Permissions Editor may not know how to handle the ACL that the custom application creates or modifies.

Even though the algorithm above still applies, there is no way of precisely determining the access to the object. The following examples illustrates this point.

Access Determination Example 6

The user FredMgr is trying to gain Read and Write access to the file object that has the discretionary ACL as show in Figure 3.22. FredMgr security token indicates that he is a member of the groups Users, JnrMgrs, and Everyone.

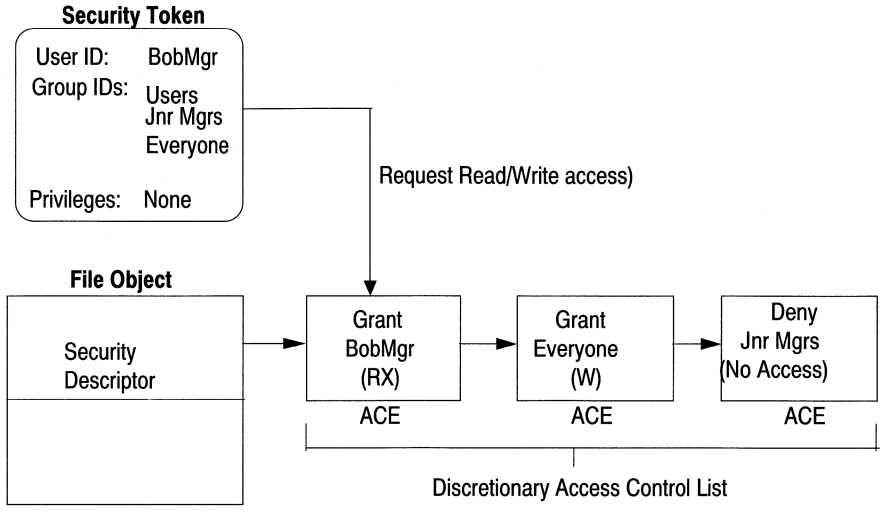


Figure 3.22 Illustration of Access Determination Example 6

In this example, a custom application has been used to manipulate the ACL on a file object. The ACL will be evaluated as follows:

The first ACE is evaluated and the Read bit in the desired access mask is cleared. Then processing moves on to the second ACE where the Write bit in the desired access mask is cleared. After the second ACE is evaluated, all the bits in the desired access mask are cleared, BobMgr is granted Read/Write access to the file object, even though the third ACE explicitly denies JnrMgrs access to the file object.

If the Windows NT Permissions Editor had been used to apply the same permissions to the file object, the deny JnrMgrs ACE would have been ordered first in the ACL and BobMgr would have been denied access to the file.

Now that we have described and illustrated how discretionary access is determined on a Windows NT system, one last subject remains. The next section describes how auditing is determined.

Audit Determination

Following access determination, audit information must be evaluated to determine whether audit messages are to be generated. The following information is required to make this decision:

- The subject attempting the access (that is, the set of identifiers representing the subject)
- The desired accesses with all generic access types mapped to standard and specific access types
- The final determination of whether access is to be granted or denied
- The audit ACL associated with the target object

Given this information, each ACE in the audit ACL is evaluated as follows:

1. The type is checked to see if it is SystemAudit. If not, the ACE is skipped.
2. The identifier in the ACE is compared to the set of identifiers representing the subject. If a match is not found, the ACE is skipped.
3. The desired accesses are compared to the access mask specified in the ACE.
4. If none of the accesses specified in the ACE's mask were requested, the ACE is skipped. The SUCCESSFUL_ACCESS_ACE_FLAG and FAILED_ACCESS_ACE_FLAG flags of the ACE are compared to the final determination of whether access was granted or denied.
5. If access was granted but the SUCCESSFUL_ACCESS_ACE_FLAG flag is not set, or if access was denied but the FAILED_ACCESS_ACE_FLAG flag is not set, the ACE is skipped.

Having made it through all the steps above, an audit message is generated.

The scenario shown in Figure 3.23 illustrates this process. In this scenario, a system access ACL being evaluated. Let's assume that Write access to the file object is granted and the SUCCESSFUL_ACCESS_ACE_FLAG is set in each ACE.

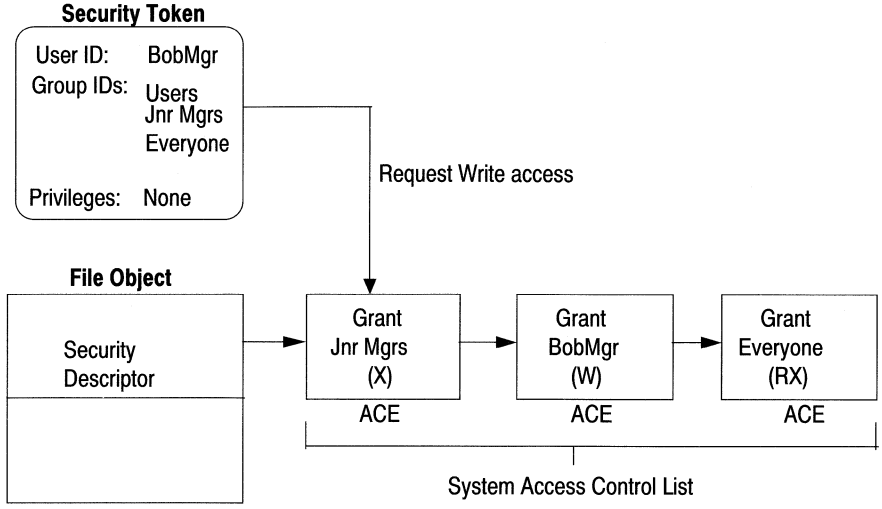


Figure 3.23 Auditing determination example

In this example, the system ACL is evaluated as follows:

The first ACE is evaluated, and a match is found as BobMgr is a member of JnrMgrs. However, when the desired access is compared to the access mask of the ACE, no match is found and the ACE is skipped.

When the second ACE is evaluated BobMgr's SID is found in the ACE and the desired access mask is compared to the access mask of the ACE, a match is found. Evaluation of the ACE continues with a check on access flags, the SUCCESSFUL_ACCESS_ACE_FLAG is set, processing stops, and an audit message is generated.

Summary

Security is an integral part of Windows NT. It is pervasive throughout the entire operating system, and it cannot be bypassed.

The Windows NT security model has a uniform design that is integrated with the Windows NT object model. This uniformity allows Windows NT to potentially support many different security environments. The system is designed so that it can be extended and integrated into existing security environments.

Perhaps most practically for users, the Windows NT security model provides for secure sharing of desktop resources.

CHAPTER 4

Windows NT Configuration Registry

One of the greatest challenges facing the corporate MIS department is the management and maintenance of the corporations computer systems. It's never been easy to manage the hardware, the operating systems, and applications even on mini and mainframe computers. And on a PC, every time you add a new application or hardware device to you system, it seems you need add yet another configuration file to your system. Now imagine a corporation with hundreds or even thousands of PCs which are distributed across a geographically dispersed area and you can begin to appreciate the size of the configuration maintenance problem.

The Windows NT Configuration Registry introduces an entirely new concept in the area of configuration management of an operating system and the applications that run on it. Windows NT not only breaks ground in the management of individual PCs, it also forms the basis of a complete enterprise-wide management solution.

The Windows NT Configuration Registry provides a secure, remotely accessible method for system configuration that could not be provided by .INI files or by the UNIX multi-user design. The Registry is a unified and centralized database for storing operating system, application, and hardware configuration data. The Registry provides for storing this data in a hierarchical form and supporting multiple users. It also offers large capacity and good integrity.

This chapter describes the goals for the Windows NT Configuration Registry and provides a general overview of its structure. Then it describes the configuration information found in the four trees that make up the Registry. The first of these trees contains per-machine information including values that define the computer's hardware configuration and the services and software that run on the computer. The next tree is used for compatibility with the Windows 3.1 registration database. The last two trees define user configuration information.

Finally, the chapter explains how data flows to and from the Registry and how system security affects the Registry.

Registry Design Goals

The introduction to this chapter pointed out some of the main problems that face anyone who has to manage a single or multiple PC environment. Chief among the design goals for the Windows NT Registry was to address these problems and offer a unique solution for effectively manage operating system and associated configuration data. Specifically, the Registry design team set out to accomplish the following:

- Simplify the support burden for configuration information
- Centralize all the configuration information for the Windows NT operating system
- Provide access security which allows for local or remote access to this information
- Provide a means to store per-user, per-application, per-machine configuration information
- Provide support for multiple parallel and multiple serial users
- Provide storage for large, complex bodies of configuration information
- Provide a single set of APIs for manipulating all Registry information
- Provide integrity and crash resistance

The rest of this chapter describes how the Windows NT Registry was implemented to meet these goals. To begin, the next section provides a big-picture view of the Registry.

Overview of the Windows NT Registry

The Registry is a secure database which is designed to centralize all configuration information for a Windows NT system.

The beauty of the Registry is that regardless of which application you are configuring, the update of the configuration data happens at one location—the Windows NT Registry.

While application developers are probably the primary users of the Registry, benefits of the Registry are passed on to both the administrator and the user. For a developer, the Registry provides a clean, structured approach to storing application-specific configuration data. For the administrator, the Registry provides a central, secure location for all of their computer's configuration data. Users may not access the Registry directly, but will appreciate the fact that their administrator can easily access it across the network, or move it from computer to computer (as users change offices, for example).

As Figure 4.1 illustrates, the Windows NT Configuration Registry eliminates the need for multiple configuration files including AUTOEXEC.BAT, CONFIG.SYS, and a host of .INI files. All of the information formerly housed in these files is now centrally located in the Windows NT Registry.

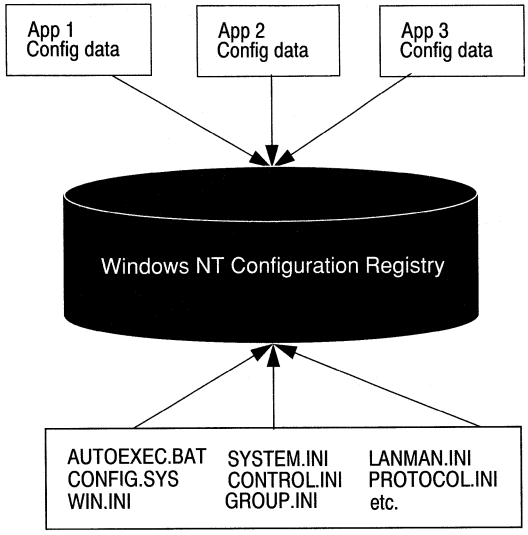


Figure 4.1 Windows NT Configuration Registry data

Note While we discourage using .INI files in favor of using Registry entries, some applications (particularly 16-bit Windows-based applications) will continue using them, at least for the time being. NT supports .INI files solely for compatibility with those applications and related tools (such as installers).

The Registry is structured as a set of four trees. Each tree is named, and each of the trees' nodes, or *keys*, are named. Each key may contain zero or more data items called *value entries* which are associated the key. In the Registry tree structure, keys are analogous to directories, and the value entries are analogous to files.

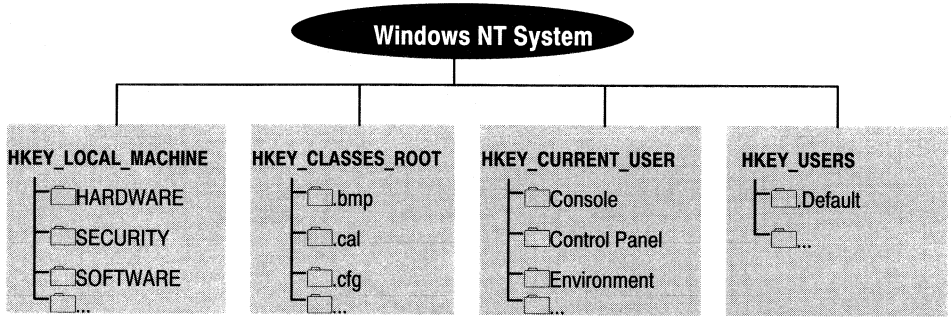


Figure 4.2 Four trees make up the Windows NT Registry

Figure 4.3 shows the four Registry trees as displayed by the Registry Editor.

The Windows NT Registry Editor displays the Registry's hierarchical structure in a way similar to how File Manager displays hierarchical directory structures.

Important The Registry Editor is a tool designed for system administrators. Users should update settings though Control Panel utilities instead of using the Registry Editor. Only people who know what they are changing should modify settings via the Registry Editor.

To start the Registry Editor (REGEDIT.EXE), you can use the Run command, or add the Registry Editor icon to a Program Manger group.

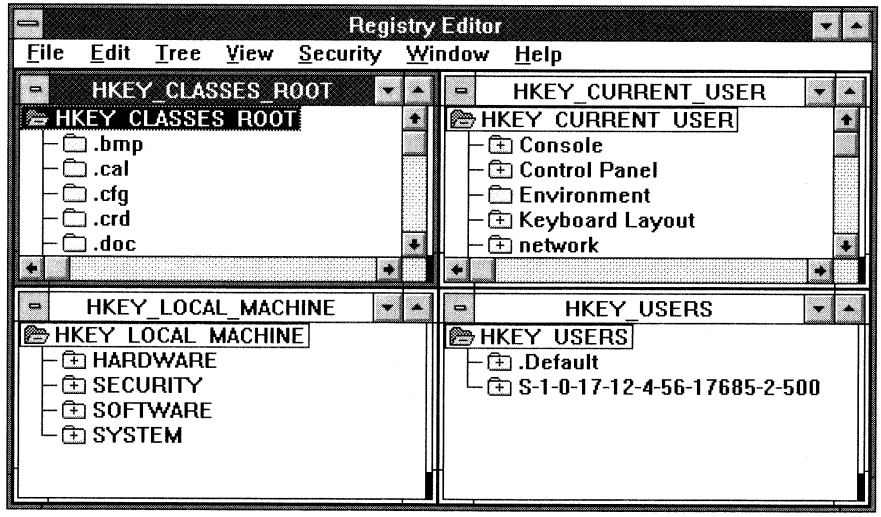


Figure 4.3 The Registry Editor

As mentioned before, the Registry is comprised of four trees which you can see in Figure 4.3. The Registry name space, then, is rooted at the corresponding four predefined key handles (or names) which are responsible for holding the per-machine and a per-user databases. Table 4.1 identifies and defines these four predefined key handles.

Table 4.1 First-Level Registry Keys

Predefined key	Handle description
HKEY_LOCAL_MACHINE	Root of tree containing information about the local computer system. This includes hardware and operating system characteristics such as bus type, system memory, installed device drivers, and boot control data.
HKEY_CLASSES_ROOT	Root of tree containing object linking and embedding (OLE) and file-class association data subkey of HKEY_LOCAL_MACHINE.
HKEY_CURRENT_USER	Root of a sub-tree of HKEY_USERS, containing the per-user environment data for the currently logged on user.
HKEY_USERS	Root of tree containing per-user environment data for all users, for a given computer. All currently active profiles will appear under this point (but *not* all profiles that might ever be active, or that have ever been active.)

Registry data is maintained in the Registry as *value entries*, which are referenced from Registry keys. Each Registry key has a fixed part and two variable parts. The fixed part consist of the key's name, class, security descriptor, and a timestamp.

One of the key's variable parts is the list of child keys, called *subkeys*. The other variable part is the key's set of value entries. Each value entry has a name, a type, and a data field (in that order), as in the following example:

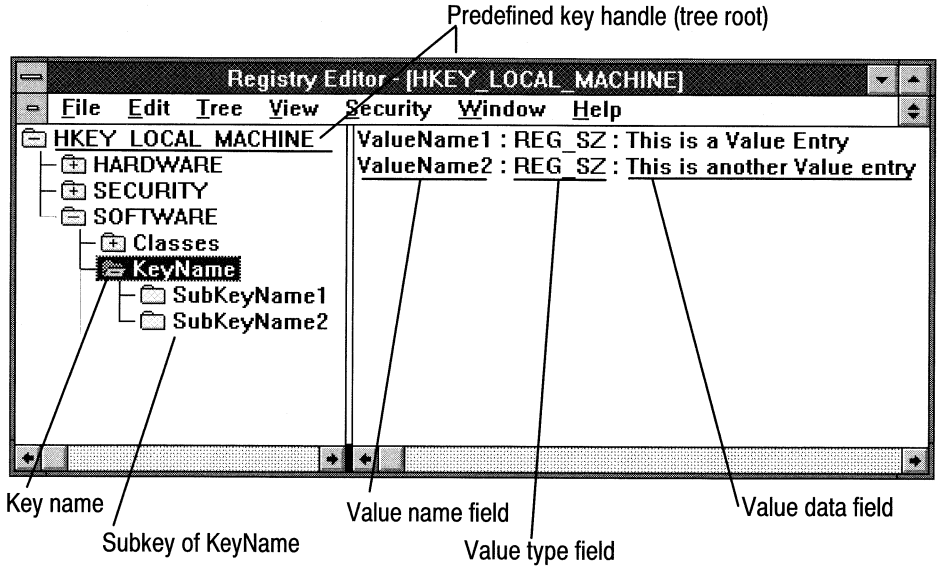


Figure 4.4 Sample node in the Registry

As illustrated by Figure 4.4, the Registry Editor shows data in two panes. Note that all of the values displayed in the right pane are associated with the selected key on the left.

The data field of a value entry can store arbitrary data which can have a size of up to 1MB.

Note This is an artificial limit which is implemented to deter Registry abuse.) However, the Registry is intended to hold objects that are relatively small in size. Large objects such as bitmaps and executables images should normally be stored in the file system and given a pointer within the Registry. Registry quota support is planned for a future release of Windows NT.

Before a key can be accessed by a program, it must be opened by specifying the desired access rights using an Open or Create call. The handle returned by these calls can then be use to manipulate the key. Subkeys can be opened or created by specifying their path relative to the key referred to by the handle. Values can only be referenced by using an Open Key handle.

Atomicity and Sharing

On a Windows NT computer, the Registry is the center of all configuration information. And, since Windows NT is a preemptive, multitasking operating system, it is reasonable to assume that more than one program may try to read and write to the Registry at the same time.

The Registry implicitly handles sharing of its data by implementing a certain amount of atomicity. That is, edits to individual value entries are atomic, even across system crashes. The system promises that anytime a value entry is modified, the next time it is read it will have the old value or the new value. That is, it will have the Type, Name, and Data field of either old value entry or the new value entry, and not some combination of the two. However, atomicity is not guaranteed across writes to multiple value entries of the same key. For example, if the system failed while writing to two value entries, after the failure, you could have two old value entries, two new value entries, or one old and one new value entry.

Many different edits to a particular key's value entries, or indeed to a particular value entry, may occur at once. The Registry ensures that these edits occur atomically. Some may be lost (because later edits overwrite them), but none will be corrupted. Of course, two different value entries in the same key may end up being inconsistent with each other.

Note A program must flush a key to ensure that its change is written to hard disk. Closing a key handle does *not* ensure that the change is written to disk. However, flushing a key is very expensive and should only be done when necessary.

The Registry does have a lazy-flush function that ensures Registry data is never more than a few seconds old. If it is acceptable to lose the change if the computer crashes in the few seconds after the edit, you need not flush the change at all. Changes with serialization or high integrity requirements must be explicitly flushed.

Registry Security

Registry keys are shareable objects, and hence they are subject to Windows NT security. That is, Registry keys can have Access Control Lists (ACLs) applied to them. ACLs can only be applied to Registry keys but the ACL protects all of the key's value entries from unauthorized modification. Hence it is possible to protect users from modifying the configuration data of a Windows NT computer.

For a complete discussion of ACLs, see Chapter 3, "Windows NT Security Model."

Hives and Files

Physically, the Windows NT Configuration Registry exist as a set of files on your hard disk. For the purpose of creating these files, the Registry is divided into sections called *hives*. These hives produce the physical files that make up the Registry.

For example HKEY_LOCAL_MACHINE\SYSTEM refers to the root of the System hive.

A hive maps directly to a file. For example, assuming your system root is C:\WINNT, HKEY_LOCAL_MACHINE\SYSTEM maps to C:\WINNT\SYSTEM\CONFIG\SYSTEM.

Table 4.2 lists the standard hives for a Windows NT system and the paths of their corresponding physical files (assuming the system root is C:\WINNT).

Table 4.2 Windows NT Registry Hives

Hive	Physical file
HKEY_LOCAL_MACHINE \SYSTEM	C:\WINNT\SYSTEM32\CONFIG\SYSTEM \SYSTEM.ALT
HKEY_LOCAL_MACHINE \SOFTWARE	C:\WINNT\SYSTEM32\CONFIG\SOFTWARE \SOFTWARE.LOG
HKEY_LOCAL_MACHINE \SECURITY	C:\WINNT\SYSTEM32\CONFIG\SECURITY \SECURITY.LOG
HKEY_LOCAL_MACHINE \SAM	C:\WINNT\SYSTEM32\CONFIG\SAM \SAM.LOG
HKEY_USERS \DEFAULT	C:\WINNT\SYSTEM32\CONFIG\DEFAULT \DEFAULT.LOG

Table 4.2 shows the hives that are created when you install Windows NT for the first time. Additionally, user profile hives can be created when a user logs on interactively at that workstation—that is, when someone sits at that workstation and uses the attached keyboard and screen interactively. Users have permission to keep a locally-cached copy of their profile if they have read and write access to the following Registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion
   \ProfileList
```

A unique subkey is then created for the user, specifying the full path of the user's locally-cached copy of his or her profile. This path will look something like the following (where C:\WINNT is the system root):

```
C:\WINNT\SYSTEM32\CONFIG\USER000
```

The hive that is created depends on the following algorithms.

For a user with no assigned profile:

- When the user has permission to keep local copy of profile

  A copy of the User Default profile (USERDEF) is loaded under the key HKEY_USERS\<*User's_SID*> and a locally-cached copy of this profile is saved as the user's profile. USERDEF is created when you install Windows NT and its full path is <*sysroot*>\SYSTEM32\CONFIG\USERDEF. The user's local copy resides in the same directory. When the user logs off, all changes made to the profile are saved to the locally-cached profile just created.

- When the user doesn't have permission to keep local copy of profile

  The User Default profile (USERDEF) is copied to a temporary file (<*sysroot*>\SYSTEM32\CONFIG\TMPDEF00) and this copy will be used by the user. When the user logs off, all changes made to the profile will be lost.

For a user with an assigned profile:

If there is a profile assigned to the user, this profile is used if possible. Otherwise, the user gets the locally-cached copy of the profile or a copy of the default profile (if not a mandatory profile).

Note A profile can be assigned to a user and set up on local or remote computer using the Administrative Tools User Profile Editor and User Manager for Domains. Both of these tools are supplied with the Windows NT Advanced Server product and with the *Windows NT Resource Kit*. For information about creating, assigning profiles on local and remote computers, see the *Windows NT Advanced Server System Guide* or the *Windows NT Advanced Utilities Guide*.

- When the user has a mandatory profile (with a filename extension of .MAN)on local computer and on remote computer

  The profile is copied to a temporary file in the <*sysroot*>\SYSTEM32\CONFIG directory, and the user will use this temporary profile as his or her profile. All changes made to this profile are lost when the user logs off. If the mandatory profile is not accessible (for example, if there is no read permission for the user on the file, or if the server where the mandatory profile resides is down), then the user will not be able to log on.

- When the user has a personal profile on a local computer

  If the profile is not accessible, then the user will get a copy of USERDEF.

- When the user has a personal profile on a remote computer
  - If the user has permission to save local copy

    If the remote profile is accessible, and there are no local copy of the profile yet on the local computer, the remote profile is copied to the path for the local copy of the profile (for example, C:\WINNT\SYSTEM32\CONFIG\USER002). And this local copy is loaded into the registry as the user's profile. At logoff, the changes are saved to the local copy and the local copy is copied back to the remote profile file.

    If the remote profile is accessible and the local copy exists, the newest file overwrites the older version and the file is treated as described above.

    If the remote profile is not accessible but local copy exists, the local copy is loaded into the Registry and used as the user's profile. At logoff, the changes are saved to the local copy. The local copy is copied over to the remote profile file at the next log on (if the remote profile is then accessible and if the date on the local copy is newer than the one on the remote profile).

    If the remote profile is not accessible and the local copy does not exist, then a copy of USERDEF is saved to the local profile copy path. It will then be loaded in the Registry and used as the user's profile. At logoff time, the changes made to the profile will be saved in the locally-cached profile.

  - If the user doesn't have permission to save local copy

    If accessible, the remote profile is copied to a temporary file in the directory called <*sysroot*>\SYSTEM32\CONFIG. The user's changes are saved back to the remote profile when the user logs off.

    If the remote profile is not accessible (for example, if the server is down, or there is no permission for accessing the file) and the user does not have a local copy of his or her profile, a copy of USERDEF is saved in the \CONFIG directory as a temporary file. The user uses this temporary default profile for his or her entire logon session. Changes made to the profile will be lost.

Important For secure installations, these files should be on an NTFS volume where they can be secured with access control lists.

The next several sections describe each of the four trees that make up the Windows NT Registry, beginning with the tree containing computer-specific (or "per-machine") data.

HKEY_LOCAL_MACHINE

HKEY_LOCAL_MACHINE is the root of the per-machine configuration data and is arguably the most important predefined key handle. This handle is used to reference configuration data pertaining to the computer which is local to the calling process, regardless of which user is logged on and what software is in use.

HKEY_LOCAL_MACHINE has five subkeys:

- SYSTEM controls boot, device driver loading, Windows NT services, and operating system behavior.
- HARDWARE is the installed hardware database. This subkey represents the physical configuration of the computer including such data as processor type, bus type, video adapter type, which interrupts and I/O address are used, and so on. This data is volatile and is computed each time the systems boots.
- SOFTWARE contains the per-machine software data. This subkey contains data about software physically installed on the local computer. By querying this subkey, it is possible to retrieve information about software that is installed on this computer.
- SECURITY holds the database used by the protected security subsystem, which contains the local security policy. This subkey is almost entirely used by the security subsystem
- SAM holds the security information for user and group account , and for the domain.

These HKEY_LOCAL_MACHINE subkeys are described in the following sections.

HKEY_LOCAL_MACHINE\SYSTEM

The data in the Registry hive called HKEY_LOCAL_MACHINE\SYSTEM key is organized into control sets. A *control set* is simply an instance of a set of system parameters. For all intents and purposes, you can consider that the system has two control sets—a current control set (CurrentControlSet) and a last known good (LastKnownGood) control set.

If you were in the MS-DOS world, you might think of this as having two sets of CONFIG.SYS files—a current one and a backup that is known to be good.

CurrentControlSet is a link to the control set which booted the system. During the boot phase, a choice is made as to which control set to boot from. All data needed to control booting of Windows NT, which must be stored rather than computed during booting, resides in the System hive, HKEY_LOCAL_MACHINE\SYSTEM. A complete copy of the data is stored in SYSTEM.ALT

The following shows the tree structure of the System hive:

```
HKEY_LOCAL_MACHINE\SYSTEM
   ControlSetnnn
   Select
   Setup
   Clone
   CurrentControlSet
      Control
      Services
```

The following sections describe the SYSTEM subkeys.

ControlSet*nnn* Subkeys

The keys named ControlSet*nnn* describe different configurations for the current computer. The CurrentControlSet key contains the same information as the ControlSet*nnn* key that is currently selected. (For more information on selection, see the following section on the Select subkey.) When ControlSet001 is the only such key, it contains the same information as CurrentControlSet.

Applications typically do not modify the information in these keys; modifications to configuration information are usually made under the CurrentControlSet key.

The mechanism for defining additional control sets and selecting among them has not been defined for this release. When the system starts, a copy of the ControlSet*nnn* key that is used to start the system is stored as the Clone key. The CurrentControlSet key is actually a link to the ControlSet*nnn* key that was used to start the system.

The ControlSet*nnn* keys prevent incorrect configuration information from creating a computer that cannot be started. When incorrect configuration information prevents the system from starting, the system checks the Select key to find the last control set that was successfully used to start the system, then uses that control set to complete the start-up routine. The CurrentControlSet key identifies the successful control set.

Each ControlSet*nnn* key has two subkeys—Control and Services. These are described later in the "Control Sets" section.

Select Subkey

The Select subkey maintains information about the current, LastKnownGood, default, and failed control sets for the currently selected computer. The Select subkey has the following named values:

- Current identifies the control set from which the CurrentControlSet key is derived. If it is 0x1, the key producing CurrentControlSet is ControlSet001.
- Default identifies the default control set. If it is 0x1, the default control set is ControlSet001.
- LastKnownGood identifies the last control set that successfully started the system. If this value is 0x1, the last control set known to be good is ControlSet001.
- Failed is a flag indicating whether the control set identified by the Default value successfully started the system. If this value is 0, the default control set was successful. If it is 1, the system wasn't able to use the default control set to start. The Current and LastKnownGood values identify the control set that was used instead.

Setup Subkey

The Setup subkey is used internally by the system for the setup program.

Clone Subkey

The Clone subkey is a volatile copy of the control set the system booted from, and used by the Windows NT Service Control Manager. Because it is volatile, if boot fails or is cut short, it will simply disappear. The Service Control Manager makes a copy of the Clone subkey to the LastKnownGood control set before any changes are made to the control set. Making the copy before any changes can occur helps ensure that a working control set is always present on the computer.

CurrentControlSet Subkey

The CurrentControlSet subkey of the System key contains the current configuration information for the local computer. The information under this key is in exactly the same form as the information for ControlSet*nnn*.

Control Sets

Once more, here is the tree structure for HKEY_LOCAL_MACHINE\SYSTEM:

```
HKEY_LOCAL_MACHINE\SYSTEM
   ControlSetnnn
   Select
   Setup
   Clone
   CurrentControlSet
      Control
      Services
```

As shown in this list, each control set has two subkeys:

- The Control key holds parameters for the Session Manager. This includes information about which subsystems to load, machine-dependent environment variables, size and location of paging file(s), and so on.
- The Services key list all of the Kernel device drivers, file systems drivers, and Win32 service drivers that can be loaded by the operating system loader, the I/O Manager, and the Windows NT Service Control Manager. Services contains subkeys that are static descriptions of hardware to which drivers can be attached.

Control Subkey

Control contains two subkeys which specify how to interpret the services list.

- ServiceGroupOrder is a list of groups in the order in which they are to be loaded. For example, the disk driver must be loaded before the file system can be loaded.
- GroupOrderList is a list of groups that list the load order of the devices within that group.

Services Subkey

A common data structure governs all the subkey entries under Services. The Services subkey uses a format as in the following example:

```
🗀  HKEY_LOCAL_MACHINE\SYSTEM...\CurrentControlSet\Services
  🗀\KeyName
  🗀\Para
```

In this example, KeyName is the name of the service that appears is Service Control database.

Each subkey of Services can have some or all of the following value entries, which govern the behavior of the service:

- Type specifies the type of service.

  Type=DRIVER means this is a Kernel device driver. File systems are also Kernel device drivers.

  Type=WIN32_SERVICE indicates a Win32 program that can be started by the Service Controller and which obeys the service control protocol. (There are two types of Win32 service, one which runs in a process by itself, and one which will share a process with other Win32 services.)

  Type=ADAPTER means this a set of arguments for some piece of hardware.

```
Service Type     Constant
Kernel Driver    0x1
File System Driver  0x2
Adapter 0x4
WIN32 Service    0x10
WIN32 Service    0x20
```

- Start specifies the starting values for the service. This is ignored for adapters. If Type=SERVICE, Start must be AUTO, DEMAND, or DISABLED.

  Start=BOOT means that this node represents a part of the driver stack for the boot volume (bootstrap), and must therefore be loaded by operating system loader.

  Start=SYSTEM means that this node represents a driver to be loaded at Kernel initialization.

  Start=AUTO means that this node, regardless of type, is to be loaded or started by the service controller, automatically for all boots.

  Start=DEMAND means that this node, regardless of type, is available, but will not be started until the service controller is called to start it—for example, Net Start Server.

  Start=DISABLED means that this node is not to be started under any conditions.

```
Start value Constant     Loaded by component
Boot     0x0 Kernel
System   0x1 I/O subsystem
Auto load    0x2 Service Control Manager
Load on demand  0x3 Service Control Manager
Disabled     0x4 Service Control Manager
```

- ImagePath specifies a pathname. For a driver, the default is<*sysroot*>\DRIVERS\<*driver_name*>.SYS . For a service, the default is <*sysroot*>\SYSTEM\<*service_name*>.EXE. This is ignored for adapters.

- ObjectName specifies an object name. If the service type is a WIN32 Service, this is the account name the service will use to log on when the service is run. If the service type is Kernel driver or file system driver, this name is the Windows NT driver object name which the I/O Manager uses to load the device driver. The default is \DRIVER\<*node_name*>.
- ErrorControl specifies the level of error control for the service.

  ErrorControl=NORMAL means that boot should proceed if the node fails to load or initialize.

  ErrorControl=SEVERE means that if boot is not based on LastKnownGood control set, fail it (that is, switch to LastKnownGood, then try again). If this is a LastKnownGood attempt, press on in case of error.

  ErrorControl=CRITICAL means to fail the attempted boot. If it is not using LastKnownGood, try booting with LastKnownGood. If the boot attempt was using LastKnownGood, run a bug-check routine.

```
Error control   Constant
Critical    0x3
Severe  0x2
Normal  0x1
Ignore  0x0
```

- Group specifies a group name. See discussion of load ordering below. Default is NULL_GROUP.
- DependOnService specifies zero or more service keynames. See discussion of load ordering. Default is empty.
- DependOnGroup specifies zero or more group names. See discussion of load ordering. Default is empty.
- Tag specifies a load order within a given group.

Note that some keys, such as an adapter entry, may have a parameters subkey which contains an arbitrary set of values to be passed to the driver. The following is an example:

```
\UB01
   Parameters
      BusNumber
      BusType
      CardType
      InterruptNumber
      IoBaseAddress
      MediaType
      MemoryMappedBaseAddress
      NbProvider
```

Services Load Ordering and Dependencies

Since the nodes in the Service list are Registry keys, no assumptions can be made about the order in which they will appear in an enumeration. Therefore, explicit control of load ordering is supported.

Groups allow nodes to be loaded in an order related to the type of object they are. For example, the SCSI® port driver can be loaded before any of the mini port drivers. The ordering is the one declared in \Control\ServiceGroupOrder:<*List*>.

Assume that we are running on a computer with a section of the Registry that looks like this:

```
MACHINE\SYSTEM\CurrentControlSet
control
   ServiceGroupOrder
      List=...Video... NDIS..TDI .....
services
   elnkii
      Group=NDIS
   Linkage
      Bind=\Device\Elnkii01
      Export=\Device\Elnkii01
      Route='Elnkii01'
   Nbf
      Group=TDI
   Linkage
      OtherDependecies=+NDIS
      Bind=\Device\Elnkii01
      Export=\Device\Elnkii01\Nbf_Elnkii01
      Route='Elnkii' 'Elnkii01'
   Parameter
```

Group ordering will force "Video" to load first, then "NDIS," "TDI," and so on for groups in the list. Thus, we load all the drivers in the correct order.

The OtherDependencies key allow nodes to be loaded in an order related to particular other nodes they are closely associated with. For example, the Nbf transport depends on an NDIS driver being loaded. That is, in order for Nbf protocol stack to load successfully, an NDIS network card driver must have been loaded first.

Service Control Manager

As we mentioned before, the Windows NT Service Control Manager maintains a database of installed services. It also promotes a unified and secure interface for controlling installed services. The Service Control Manager is responsible for maintaining status information on each service, for starting, stopping, and pausing services, and for handling service dependencies.

The Service Control Manager is also responsible for recovering the system that failed to boot due to a bad service configuration. If during the system boot, a service which is crucial to the system fails to start, the Service Control Manager will instigate a reboot using the LastKnownGood service configuration.

The Service Control Manager exports its service to three types of clients:

- Setup and configuration utilities
- User interface programs
- Win32 service programs

A setup or configuration program calls Service Control APIs to install and upgrade a service in the service control database. For example, a software developer might want to create, delete or query a service configuration. These services are used by the Windows NT setup program.

User interface programs will allow you to control the behavior of services that are installed in the service control database. Operations such as start, stop and pause are available through the service control API. These services are used extensively with Windows NT networking services.

The Service Control Manager sets status information for Win32 service programs via service control APIs.

HKEY_LOCAL_MACHINE\HARDWARE

There is a certain amount of hardware data that resides in the Registry that is volatile and needs to be computed at boot time. This includes information about hardware components on the system board and about which interrupts are hooked by which hardware devices. All of this information is stored under the HKEY_LOCAL_MACHINE\HARDWARE key.

The HARDWARE key contains three subkeys—Description, DeviceMap, and ResourceMap—described in the following sections.

Description Subkey

HKEY_LOCAL_MACHINE\HARDWARE\Description refers to the root of the hardware database built by operating system loader. On ARC computers, this is a copy of the ARC configuration database which is extracted from firmware. On non-ARC (usually *x*86) computers, it is a subkey in the same format as the ARC tree, but contains whatever data the system's Hardware Recognizer program could find.

The *Hardware Recognizer* is a program run as part of the Windows NT boot sequence. On *x*86-based computers, the program responsible for recognition is NTDETECT.COM.

The configuration tree built by NTDETECT.COM must conform to the standards defined by the ARC specification. For example, the Windows NT Kernel must be able to process the tree regardless of how the tree is built.

Note If the target computer is not PC-compatible (for example, it has more than one I/O bus or has more than one central processor) OEMs must provide their own Hardware Recognizer. That is, they must supply their own version of NTDETECT.COM.

Hardware Detection

The world would be a simpler place if the Hardware Recognizer could detect every hardware component of a *x*86-based PC. However, in the real world that is not the case. At this time, the following information can be detected by the *x*86 Hardware Recognizer:

Table 4.3 Detectable Hardware Information

Information	Example Registry entry
Machine ID	AT® or AT compatible
Central Processor	Intel 80486-D0
Floating Point Coprocessor	Intel 80486-D0
Bus/Adapter Type	ISA
Video	Video Seven™ VGA DRAM
Keyboard	PCAT_ENHANCED
Communication Port	COM1
Parallel Port	PARALLEL1
Floppy	Floppy1/Floppy2
Mouse	Microsoft Serial Mouse

Associated with each hardware component that is detected are a set of value entries which store configuration data for that component. The following list shows the HKEY_LOCAL_MACHINE\HARDWARE\Description subkeys whose value entries have a specific names set by convention:

- Component Information contains binary data that identifies among other things, version information for the associated key entry.
- Identifier contains the identifier of a component, if specified.
- Configuration Data contains the component specific information such as I/O port addresses, IRQ number, DMA channel, and so on. This entry is not present if no such data is available for a particular key.

Each key describes a piece of hardware component and the pathname represents the type of the component. For example, the following path represents the first disk controller of the first EISA adapter/bus:

```
HKEY_LOCAL_MACHINE\HARDWARE\Description\System\EisaAdapter\0
   \DiskController\0
```

Note The ordering is 0-based.

The following path represents the second floppy drive attached to the controller:

```
HKEY_LOCAL_MACHINE\Hardware\Description\System\EisaAdapter\0
   \DiskController\0\Floppy\1
```

Note that the node associated with the EisaAdapter keyname does not contain any data entry. All the hardware information for the first EisaAdapter is stored in the 0 keyname under the EisaAdapter node.

ResourceMap Subkey

This key and the next (DeviceMap) are concerned with device driver management and communication.

In order to track which hardware resources are in use, and by which drivers, the system will keep a resource map. ResourceMap is the node that holds this map. This data is volatile, and is thus recreated on each boot of the system.

The structure of the subkey is as follows:

```
HKEY_LOCAL_MACHINE\HARDWARE\ResourceMap
□<device_class>
   □<driver_name>
      Device_Name=<Resource_Descriptor_List (List of resources
        currently in use)>
```

In this case,

- *<device_class>* is the name of the general class of devices being described.
- *<driver_name>* is the specific driver from the general class of drivers that is use by the system. This driver name can be found in the list of active services.

All device drivers are expected to report their use of I/O ports, I/O memory addresses, interrupt vectors, and statically-allocated DMA channels. Such reporting allows the system to detect and report conflicts, to guide users (and install programs) in the configuration of new devices and drivers.

DeviceMap Subkey

Like ResourceMap, this key is used for device driver management and communication.

There are classes of device driver which need to communicate the names of the device objects they have created, along with certain properties of those device objects to other device drivers, and to user-mode code.

For example, video drivers need to tell the Windows USER component which type of video hardware is associated with which device object, and which user-mode display DLL should be bound to it.

The device map found in the DeviceMap node offers a solution to this problem. The structure of the DeviceMap subkey is as follows:

```
MACHINE\HARDWARE\DeviceMap
<device_class>
   \<device_object_name>=<back_map_data>
```

In this case,

- *<device_class>* is the name of the general class of devices being described; for example, Video or SerialComm.
- *<device_object_name>* is the name of the device object being described.
- *<back_map_data>* is device class specific data used by user mode software that needs to communicate with the device. It can null. In the case of video, it is the name of the video driver's entry in the service list—for example, VGA or Tiga. The Windows USER component can use this name to find the service entry for the relevant driver, and from that entry, determine which DLL to bind to it.

For example, the VIDEO key might have only the following value, if the local computer had a single VGA device:

```
\Device\Video0:REG_SZ:\REGISTRY\HKEY_LOCAL_MACHINE\SYSTEM
   \ControlSet001\Services\V7vram\Device0
```

This path allows the system to find information about the video driver. This information is stored under the following key:

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\V7vram
```

HKEY_LOCAL_MACHINE\SOFTWARE

The SOFTWARE subkey contains per-machine software data. This subkey contains configuration information about software physically installed on the local computer, and would apply to anyone using this particular computer. By querying this subkey, it is possible to determine what software has been installed on a particular computer.

The Software key contains three keys—Classes, Description, and Program Groups.

The Classes key contains two kinds of subkeys:

- Filename-extension keys, which give the application associated with files whose names have that extension.
- Class-definition keys, which give the shell and Object Linking and Embedding (OLE) properties of documents in that class.

The Description key contain the names and version numbers of the software that is installed on the local computer. Installing applications should record this information in the following form:

```
HKEY_LOCAL_MACHINE\SOFTWARE\<CompanyName>\<ProductName>\<Version>
```

This information is specific to the local computer. (Information about the configuration of the application is stored on a per-user basis under HKEY_CURRENT_USER.)

For a complete description of the subkeys of CurrentVersion, see Appendix XX, "Registry Roadmap." (*Not included in Preliminary Release.*)

The Program Groups key contains the common program groups—that is, the program groups used in common by all users of the computer. (The program groups for individual users are stored under HKEY_CURRENT_USER.) The name of each subkey of the Program Groups key is the name of a common program group. The value of each of these subkeys is binary data describing the common program group.

HKEY_LOCAL_MACHINE\SECURITY

The Security key contains all of the security information for the local computer. None of the Security subkeys can be modified by an application. Applications that need to query security information should use the security functions to do so.

HKEY_LOCAL_MACHINE\SAM

The Security Account Manager (SAM) database key contains user and group account information for the local computer if it is a Windows NT computer. Alternately, if the computer is running Windows NT Advanced Server, this key contains information for the domain. Applications that need to query the SAM must use the appropriate APIs.

HKEY_CLASSES_ROOT

A second tree in the Windows NT Configuration Registry is called HKEY_CLASSES_ROOT. This tree includes information about file association and Object Linking and Embedding (OLE).

As with Windows for MS-DOS, the Windows NT File Manager includes an Associate dialog box that makes it possible for users to associate a filename extension with a specific application. Windows NT stores these associations in the Windows NT Registry.

The sole purpose for HKEY_CLASSES_ROOT is to provided compatibility with the Windows 3.1 registration database.

HKEY_CURRENT_USER

The third tree in the Windows NT Configuration Registry is called HKEY_CURRENT_USER. This is the root of the profile for the user who is currently logged on.

This predefined key handle contains all the information necessary to set up a particular user environment on the computer. Information such as program groups, application preferences, screen colors, and other user-profile specifics are included. The Windows NT Logon Process builds a user's personal profile environment based upon what it finds in HKEY_CURRENT_USER.

The default subkeys for HKEY_CURRENT_USER are these:

- Console contains subkeys that give the options and window size for the a console (the interface between the user and character-mode applications).
- Control Panel contains subkeys that have parameters that are adjusted by applications in Control Panel; for example, the Windows NT Desktop.
- Environment contains value entries that correspond to the current user's setting for environment variables.
- Keyboard Layout contains the value entry that gives the current active keyboard layout.

- Program Groups contains subkeys that describes the names and settings for the current user's program groups.
- Software contains subkeys that describe the current user's configurable settings for installed software that the user can use.

For a complete description of keys under HKEY_CURRENT_USER, see Appendix XX, "Registry Roadmap" *(not included for Preliminary Release).*

Note HKEY_CURRENT_USER is mapped to HKEY_USER\<*SID*>, where <*SID*> is the SID string of the user who is currently logged on.

HKEY_USERS

The final tree in the Windows NT Configuration Registry is called HKEY_USERS. This predefined handle refers to the personal profiles of users that have an accounts in the Security Account Manager's (SAM) database.

A user's profile contains all the information necessary to set up a particular user environment on that computer. Information such as program groups, application preferences, screen colors, and any other user profile specifics are included.

HKEY_USERS has two subkeys—DEFAULT and the SID string for the user who is currently logged on.

DEFAULT has the following subkeys:

- Console contains subkeys that give the default options and window size for the a console (the interface between the user and character-mode applications) for a new user of the computer.
- Control Panel contains subkeys that have the default parameters that can be adjusted by applications in Control Panel; for example, the Windows NT Desktop.
- Environment contains value entries that correspond to the default setting for environment variables for all new users of the computer.
- Keyboard Layout contains the a value entry that gives the default active keyboard layout for a new user.
- Program Groups contains subkeys that describes the names and settings for the default program groups for any new user of the computer.
- Software contains subkeys that describe the default per-user configurable settings of installed software that the user can use.

For a complete description of keys under HKEY_USERS, see Appendix XX, "Registry Roadmap" (*not included for Preliminary Release*).

Registry in Use

Having just provided the hierarchy and theory of the Registry, tree by tree, let's now look at how data flows to and from the Registry. Figure 4.5 provides an overall picture of how the Registry is structured. This figure also highlights how the Registry is used by various components of the system.

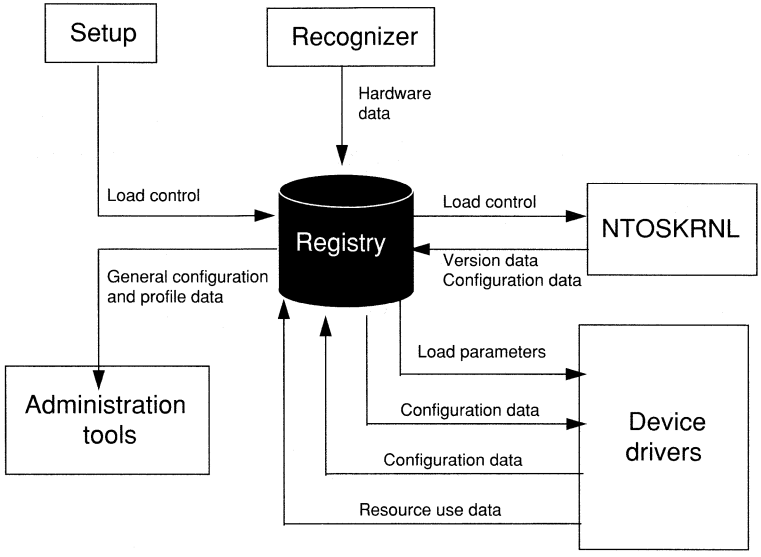


Figure 4.5 How Registry information is used by Windows NT

Starting at the top and moving clockwise, Figure 4.5 first shows the Hardware Recognizer.

When you boot a Windows NT computer, the Registry is populated with volatile hardware configuration data. This information includes a list of hardware detected in your system. On *x*86 computers, this done by a program called NTDETECT.COM. On ARC computer this information is extracted from the ARC firmware.

During boot, the Windows NT Kernel (NTOSKRNL) extracts information from the Registry such as which device drivers to load and in which order. The Kernel also passes back information on the Kernel itself, such as its version number.

Device drivers also send and receive data form the Registry. The drivers receive load parameters and configuration data. This is similar to what you might find on the DEVICE= line in a CONFIG.SYS file. A well-behaved device driver reports system resources that it is using; for example, hardware interrupts, DMA

channels, and so on. Device drivers can also report back discovered configuration data.

The Registry includes a dedicated tool for its administration called the Registry Editor. This tool isn't one you will or should use on an every-day basis, but is helpful for making detailed changes to the system's configuration. Other Windows NT tools, such as those provided in the Control Panel, permit users to modify configuration data easily without ever traversing the hierarchy of the Registry.

Finally, whenever you run a setup program—application, hardware or otherwise—the setup program adds the new configuration data to the Registry.

Summary

Under Windows with MS-DOS, ensuring that the system boots and correctly connects to the network involves ensuring that multiple configuration files contain accurate data and that some form of loose synchronization exist between them. In many cases, this is no easy task.

Windows NT changes all of this. With Windows NT it only necessary to check configuration at one location—the Registry—to ensure that the system boots correctly.

The Windows NT Configuration Registry is a large evolutionary step in the right direction for configuration management of the PC. Let's not forget that Windows NT will also be deployed on servers and "super servers" in environments that will demand the advanced configuration management features that Windows NT has to offer.

For the software and hardware vendor deploying applications and device drivers on Windows NT, the Registry provides ample facilities for "smart" installation and configuration programs that are self-configuring.

The Registry's remote management facilities will appeal to anyone who has to mange multiple Windows NT computers, and will spawn a wealth of innovative software distribution, installation, and inventory management tools.

CHAPTER 5

New Technology File System

Windows NT includes a new file system called New Technology File System (NTFS). This file system is designed with current and future applications in mind. It meets the needs of operating systems and applications running on today's powerful computing systems. This attribute-based file system supports object-oriented applications by treating all files as objects that have user- and system-defined attributes.

NTFS features the speed, reliability, and security required for file servers and high-end PCs in a corporate environment. It is designed to quickly perform standard file operations including read, write, and search—and even advanced operations such as file-system recovery—on very large hard disks. Using certain file attributes, NTFS supports data access control and ownership privileges important for the integrity of corporate data.

This chapter details the features of NTFS. To begin, the next section provides some background about the evolution of the FAT and HPFS file systems as progenitors to NTFS.

Some Background

In 1981, IBM introduced its first personal computer which ran a new operating system designed by Microsoft, MS-DOS. The computer contained a 16-bit 8086 processor chip and sported two drives for low-density floppy disks. The MS-DOS file system, FAT (named for its File Allocation Table), provided more than enough power to format these small disk volumes and manage hierarchical directory structures and files. The FAT file system continued to meet the needs of personal computer users even as hardware and software power increased by leaps year after year. However, file searches and data retrieval took significantly longer on large hard disks than on the original low-density floppies of the first IBM PC.

By the end of the 1980s, Bill Gates' prediction of "a computer on every desk and in every home" was less a dream and more a reality. Personal computers now had 16-bit processors and hard disks of 40MB and more—so big that users had to partition their disks into two or more volumes because the File Allocation Table's limit was 32MB per volume. (Later versions of MS-DOS allowed for larger disk volumes.)

In 1990, a high-performance file system (HPFS) was introduced as a part of the OS/2 operating system version 1.5. This file system was designed specifically for large hard disks on 16-bit processor computers. On the heels of HPFS came HPFS386. It was introduced as part of Microsoft LAN Manager and was designed specifically for the 32-bit 80386 processor chip.

Because of features like speed and universality, both FAT and HPFS are now used in a vast number of corporate settings. The next two sections describe the features of FAT and HPFS in more detail.

FAT File System

The FAT file system is named for its method of organization—the File Allocation Table. This table of values provides links from one allocation unit (one or more sectors) to another, as shown in the Figure 5.1.

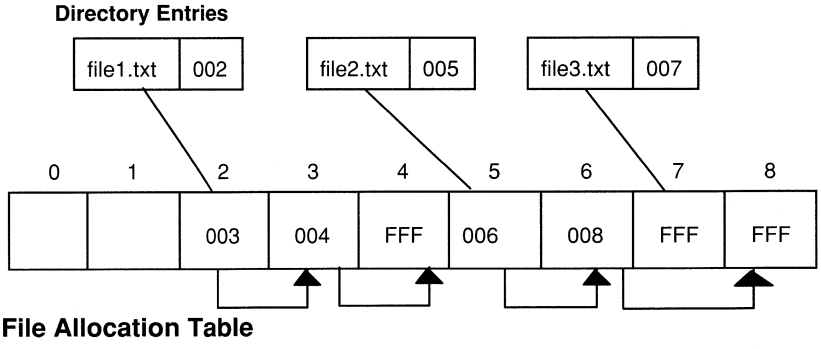


Figure 5.1 File Allocation Table

This illustration includes three files. The file named FILE1.TXT is a file that is large enough to take the space of three allocation units. A small file FILE3.TXT takes space in only one allocation unit. The third file, FILE5.TXT, is a large, fragmented file. In each case, the directory entry points to the first allocation unit containing the file. If the file contents go beyond one allocation unit, the first allocation unit points to the next in the chain. FFF indicates the end of the chain.

The FAT file system is a simple file system originally designed for small disks and simple directory structures. Its design has been improved over the years to work more effectively with larger disks and more powerful personal computers. With MS-DOS version 4.0, the FAT entries grew from 12 bits to 16 bits in size, thus allowing for partitions larger than 32MB.

The FAT file system organizes the disk in this way:

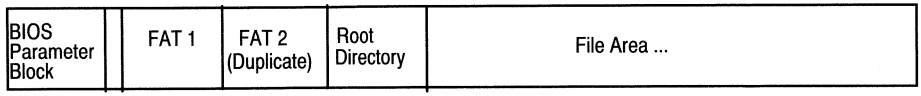


Figure 5.2 A FAT disk partition (volume)

The root directory has a fixed size and location on the disk. Directories are special files with 32-byte entries for each file contained in that directory. Each directory entry includes the following information:

- Filename (eight-plus-three)
- Attribute byte (8 bits)
- Modification time (16 bits)
- Modification date (16 bits)
- Starting allocation unit (16 bits)
- File size (32 bits)

FAT file attributes include the file's subdirectory, the volume label, and four special attributes that can be turned on or off—archive file, system file, hidden file, and read only.

A key feature of the FAT file system is simplicity. This file system has very little overhead. It is FAT's simplicity that yields performance.

However, FAT's performance slows noticeably on large disk volumes. In addition, it lacks more sophisticated features such as recoverability and security. And its "eight-plus-three" naming convention is commonly regarded as restrictive. These were some of the main reasons why HPFS file system was developed.

High Performance File System

The high-performance file system (HPFS) is an installable file system with features that make it a fast and powerful manager of large hard-disk volumes. HPFS also supports long filenames (up to 255 characters) which allow users to give files descriptive names.

While the FAT file system organizes the disk into partitions and logical drives, HPFS organizes the disk into *volumes*. When HPFS formats a volume, it reserves the first 17 sectors for the Boot Block, the Super Block, and the Spare Block. These three structures are used to boot the operating system, maintain the file system, and recover from possible errors.

HPFS also reserves space for a pair of 2K bitmaps at 16MB intervals throughout the volume. Each bitmap contains one bit for each allocation unit (equal to one sector) in the 8MB band, showing which allocation units are in use.

Figure 5.3 illustrates how HPFS organizes a volume:

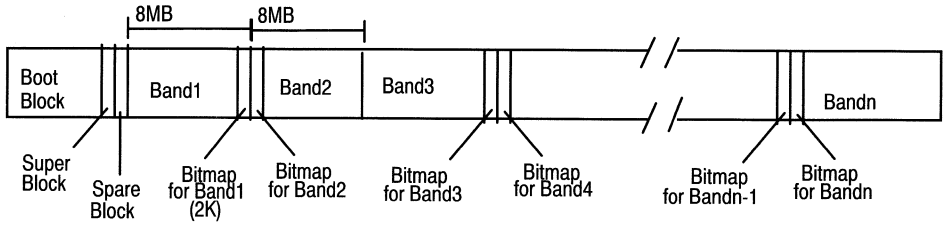


Figure 5.3 An HPFS Volume

The bitmaps are alternately located at the end and beginning of each band to allow a maximum amount of contiguous space for data (almost 16MB instead of almost 8MB). In addition, HPFS plans where it writes new files, leaving room between new and existing files so that each has room to expand into contiguous space on the disk. This feature is one of many that HPFS has to allow very fast data retrieval and minimize file fragmentation.

Another feature that accounts for fast data access is HPFS's use of Btrees. A *Btree* is tree structure with a root and several nodes. It contains data organized in some logical way so that the whole structure can be quickly traversed. The root contains a small amount of administrative information, a map to the rest of the structure, and possibly a small amount of data. The nodes contain most of the data. On large, well-populated disks, Btrees perform much better than linear lists used by the FAT file system.

HPFS uses Btrees to structure each of its directories and each of its files. Each directory entry points to Fnodes for files contained in that directory. An *Fnode* has a Btree structure that is 512K in length. This structure contains a header, the filename (truncated to 15 characters), the file length, extended attributes and access control list information, and the location of the file's data. Figure 5.4 shows the Fnode for a file whose data is contained in Extent1, Extent2, and Extent3 (where an *extent* is a range of contiguous sectors).

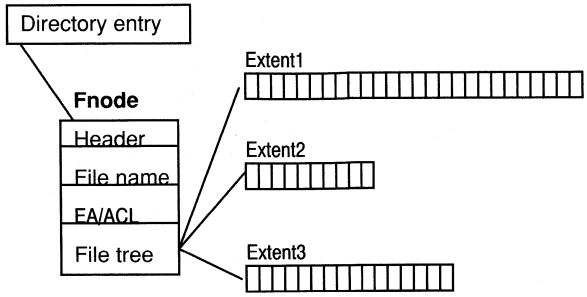


Figure 5.4 An HPFS file

Because of the arrangement of bitmaps on the volume as shown in Figure 5.3, a file extent can be nearly 16MB in length. Depending on the size of the file, the Fnode can point to as many as eight extents. If the file is so large that it cannot be contained within eight extents, the Fnode includes up to 12 pointers to allocation nodes with space for more file extents.

If the extended attribute and access control list information cannot be contained in the Fnode, the Fnode includes a pointer to that information.

Disk caching and *lazy writing* are two more features that speed operations in HPFS. In most cases, when a program needs to write information to the hard disk, it is not necessary that the information be written immediately. When you use HPFS, information that a program sends to be written to disk is temporarily stored in a cache in memory. This is called disk caching. Later, when the disk would otherwise be idle, HPFS writes the information to the disk as a background task. This is called lazy writing.

Because HPFS uses lazy writes, it is conceivable that a write error could occur long after an application is in a position to do anything about it. HPFS uses a technique called *hotfixing* to handle write errors. When an error is detected, the file system takes a free block out of a reserved hotfix pool, writes the data to that block, and updates the hotfix map. HPFS also replaces any bad sector numbers with the corresponding number of the good sector holding the data. A copy of this map is written to disk. If more than a specified number of hotfixes occur, the file system displays a message to encourage the user to run CHKDSK /F to check and possibly repair the volume.

In summary, the HPFS is a powerful file system designed to work quickly and efficiently on disks of up to 2GB. It includes powerful features not offered by FAT, including use of extended attributes.

The HPFS design does have some weaknesses. For example, if something damages the first portion of the volume, which contains boot information and a pointer to the root directory, use of the volume is lost. HPFS's use of CHKDSK at each system boot and to repair disk errors can be time-consuming. In addition, its design requiring 512-byte sector is not well-suited for larger volumes.

Note Some HPFS features are implemented differently for Windows NT. For example, with Windows NT, HPFS does not support access control list information or hotfixing. Also, disk caching and lazy writing are managed by Windows NT's Cache Manager and not the file system.

NTFS Features

NTFS provides a combination of performance, reliability, and compatibility not found in either FAT or HPFS. The rest of this chapter describes how the NTFS design accomplishes this.

NTFS has a simple, yet very powerful design. From the file system's perspective, everything on the NTFS volume is a file or part of a file. Every sector on an NTFS volume that is allocated belongs to some file. Even the file system metadata is part of a file.

The Master File Table

Each file on an NTFS volume is represented by a record in a special file called the Master File Table (MFT).

NTFS reserves the first 16 records of the table for special information. The first record of this table describes the Master File Table itself, followed by a MTF "mirror" record. The third record is the log file, used for file recovery. The log file is discussed in detail later in this chapter. The seventeenth and following records of the Master File Table are for each file and directory (also viewed as a file by NTFS) on the volume. Figure 5.5 provides a simplified illustration of the MTF structure.

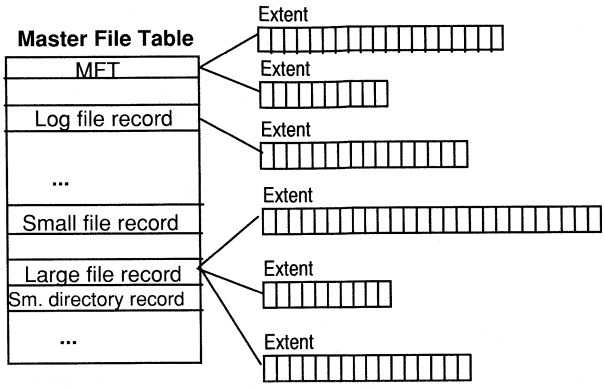


Figure 5.5 Master File Table

The Master File Table allocates a certain amount of space for each file record. The attributes of a file are written to the allocated space in the MFT. Small files and directories, such as the file illustrated in Figure 5.6, can be entirely contained within the Master File Table record.

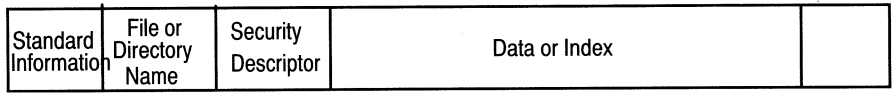


Figure 5.6 MFT record for a small file or directory

This design makes file access very fast. Consider, for example, the FAT file system. That system uses a File Allocation Table to list the names and addresses of each file. When you want to view a file, FAT first looks up the filename in the File Allocation Table and assures that it exists. Then, using the address listed in that table, FAT retrieves the file. With NTFS, as soon as you look up the file, it's there for you to use.

Directory records are housed within the Master File Tree just as file records. Instead of data, directories contain index information. Small directory records reside entirely within the MFT structure. Large directories are organized into Btrees, having records with pointers to external clusters containing directory entries that could not be contained within the MFT structure.

NTFS File Attributes

NTFS views each file (or directory) as a set of file *attributes*. Elements such as the file's name, its security information, and even its data are all file attributes. Each attribute is identified by an attribute type code and optionally an attribute name.

When a file's attributes can be written within the MFT file record, they are called *resident* attributes. For example, information such as filename and timestamp are always included in the MFT file record. When a file is too large to fit all of its attributes in the MFT file record, some of its attributes are *nonresident*. The nonresident attributes are allocated one or more runs of disk space elsewhere in the volume.

Any resident attribute of a file can be referenced as a stream for searching, sorting, and other operations. To this extent, NTFS deals with all resident attributes equally.

Table 5.1 lists all of the file attributes currently defined by NTFS. This list is extensible.

Table 5.1 NTFS File Attribute Types

Attribute type	Description
Standard Information	Includes timestamps, link count, and so on.
Attribute List	Defines valid attributes.
Filename	A repeatable attribute for both long and DOS filenames. The long name of the file can be up to 255 Unicode characters. The DOS name is the eight-plus-three, case-insensitive name for this file. Additional names, or "hard links" required by POSIX may also be included in Filename.
Version	Specifies the update version of the file.
Security Descriptor	Shows information about who can access the file, who owns the file, and so on.
Data	Contains file data. NTFS allows for multiple data attributes per file. Each file typically has one unnamed data attribute. In addition, a file can have one or more named data attributes, using a particular syntax.
Index Root	Used to implement directories.
Index Allocation	Used to implement directories.
Volume Information	Used only in volume system file and includes, among other things, the version and name of the volume.
Bitmap	Provides a map representing records in use on the MFT or directory.
Extended attribute Information	For extended attributes, there for file servers that are linked with OS/2 systems. This attribute type isn't useful to Windows NT.
Extended attributes	For extended attributes, there for file servers that are linked with OS/2 systems. This attribute type isn't useful to Windows NT.

Long and Short Filenames

One of the improvements HPFS implemented on the MS-DOS design was the ability to use long filenames of up to 255 characters. Unfortunately, files with long names on an HPFS volume couldn't be accessed from an MS-DOS operating system, which has an eight-plus-three filename limitation.

Like HPFS, NTFS allows filenames of up to 255 characters, but solves the problem of access from MS-DOS. NTFS automatically generates an MS-DOS-readable (eight-plus-three) name for each file. This way, NTFS files are accessible over a network by OS/2 and MS-DOS operating systems. This is a particularly important feature for file servers, especially in an office using personal computers with two or all three of these operating systems.

NTFS filenames use the 16-bit Unicode character set.

Generating MS-DOS Filenames

Since NTFS uses the Unicode character set for its names, there are potentially several "illegal" characters that MS-DOS cannot read in any filename. To generate an MS-DOS filename for a file, NTFS converts all of these characters from the long filename to underscore characters (_) and removes any spaces. Then, since an MS-DOS filename can have only one, NTFS removes all extra period (.) characters from the filename. Next, NTFS truncates the filename, if necessary, to six characters and appends a dash and a number. For example, each non-duplicate filename is appended with "-1". Duplicate filenames end with "-2", "-3", and so on. Filename extensions are truncated to three or fewer characters. Finally, NTFS translates all characters in the filename and extension to uppercase.

Viewing MS-DOS Filenames

Both File Manager and the **dir** command are able to display either the long NTFS filenames or the short MS-DOS filenames. This enables you to look up and manipulate files using either NTFS or MS-DOS filenames.

To see MS-DOS filenames from File Manager, choose Full File Details.

From the command line, to see both the long NTFS filename and the MS-DOS filename for each file in the directory, type the following command:

```
dir /x
```

Other Special Attribute Types

NTFS support multiple data streams. The stream name identifies a new data attribute on the file. Streams have separate opportunistic locks, file locks, allocation sizes, and file sizes, but sharing is per file. An example of an alternate stream is the following:

```
myfile.dat:stream2
```

This feature permits related data to be managed as a single unit. For example, the AppleShare Server uses this type of structure to manage resource and data forks on the Macintosh. Further, imagine a library of files where the files are defined as alternate streams, as in the following:

```
library:file1
       :file2
       :file3
```

Or consider the possibility of a "smart" compiler which created a file structure like the following:

```
program:source_file
       :doc_file
       :object_file
       :executable_file
```

POSIX Compliance

POSIX compliance permits UNIX applications to be ported more easily to Windows NT. Windows NT is fully compliant with the Institute of Electrical and Electronic Engineers (IEEE) standard 1003.1, which is a standard for file naming and identification.

The following POSIX-compliant features are included in NTFS:

- Case-sensitive naming. NTFS permits file names that differ only in case when used by the POSIX subsystem. (The Win32 subsystem is a "case-preserving" subsystem. However, Win32 application names differing only in case are not allowed.)
- Hard links. A file can be given more than one name.
- Additional timestamps.

(For related information, see the section on the POSIX subsystem in Chapter 1, "Windows NT Architecture.")

NTFS System Files

NTFS includes several system files, all of which are hidden from view on the NTFS volume. A system file is one used by the file system to store its metadata and to implement the file system. System files are placed on the volume by the format program.

The NTFS system files are listed in Table 5.5.

Table 5.2 NTFS System Files

System File	Description
Master File Table	A list of all contents of the NTFS volume.
Master File Table2	A mirror of the important parts of Master File Table, used to guarantee access of the Master File Table in the case of a single-sector failure.
Log File	A list of transaction steps, used by the Log File System for recoverability.
Volume	The name, version, and other information about the volume.
Attribute Definitions	A table of attribute names, numbers, and descriptions.
Root Filename Index	Root directory.
Cluster Bitmap	A representation of the volume showing which allocation units are in use.
Boot File	Includes the bootstrap for the volume, if this is a bootable volume.
Bad Cluster File	A location where all the bad clusters in the volume are located.

So far in this discussion of NTFS features, we have described the efficient design of the file structure. We have also described features that make NTFS compatible for use by other networked clients including MS-DOS and OS/2 clients. The next few sections describe the safety and security aspects of NTFS which make it a good choice for a corporate file system.

Data Recoverability

Until now, there were two types of file systems—careful write file systems and lazy write file systems. NTFS introduces a third type—a recoverable file system.

A *careful write file system* is designed upon the idea that it is important to keep the volume structure consistent. Two examples of careful write file systems are FAT and Digital's ODS 2.

A careful write file system works this way: When its modifying the volume structure, it orders the disk writes. Most volume updates are serialized and the disk writes for each update are ordered so that if the system failed between two disk writes, the volume would be left in an understandable state with the possibility of an expected inconsistency. The disk remained usable. Running utilities such as CHKDSK was rarely needed. (On FAT, for example, CHKDSK is needed only to recover from system failure and provides a way to restore file system consistency quickly.)

The disadvantage of careful write file systems is that serialized writes can be slow.

A second kind of file system, such as HPFS and most UNIX file systems, is called a *lazy write file system.* This type was designed to speed up disk access. Assuming that disk crashes were not a regular occurrence, a lazy write file system was designed to use an intelligent cache-management strategy and provide a way to recover data (such as the CHKDSK utility) should something happen to the disk.

All data is accessed via the disk cache. While the user searches directories or reads files, "dirty data" is allowed to accumulate in the cache. Thus, the user never has to wait while data is written to disk. Plus, the user is able to access to all the file-system resource that might otherwise be allocated for disk writing. Data gets written to disk when the computer's resources are in low demand, rather than in serial fashion.

If the same data is modified several times, all those modifications are captured in the disk cache. The result is that the file system needs to write to disk only one time to update the data. That is, the file system opens the file once, then performs all of the updates together before closing the file.

The disadvantage of a lazy write file system is that, in the event of a disk crash, recovery could take much longer than with a careful write file system. This is because a program such as CHKDSK must then scan the entire volume to recover.

NTFS is a third kind of file system—a *recoverable file system.* It combines the speed of a lazy-write file system with virtually instant recovery.

NTFS guarantees the consistency of the volume by using standard transaction logging and recovery techniques. It includes a lazy writing technique plus a system of volume recovery that takes typically only a second or two after the computer is rebooted. The transaction logging, which allows NTFS to recover quickly, requires a very small amount of overhead compared with careful write file systems.

Processing NTFS File Transactions

Each I/O operation that modifies a file on the NTFS volume is viewed by the file system as a transaction and can be managed as an atomic unit.

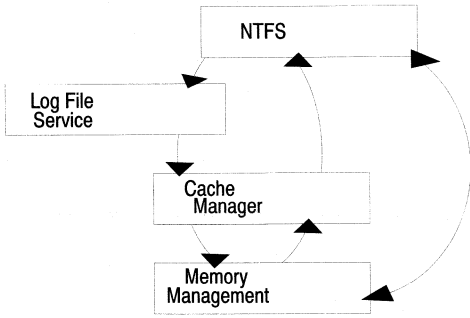


Figure 5.7 Interaction between NTFS and other Windows NT components

When a user updates a file, the Log File Service logs all redo and undo information for the transaction. For recoverability, *redo* is the information that tells NTFS how to repeat the transaction and *undo* tells how to roll back the transaction that was incomplete or that had an error.

If a transaction completes successfully, the file update is committed. If the transaction is incomplete, NTFS ends or rolls back the transaction by following instructions in the undo information. If NTFS detects an error in the transaction, the transaction is also ended.

File system recovery is straight-forward with NTFS. If the system crashes, NTFS performs three passes—an analysis pass, a redo pass, and an undo pass. During the analysis pass, NTFS appraises the "damage" and determines exactly which clusters must now be updated, per the information in the log file. The redo pass performs all transaction steps logged from the last checkpoint. The undo pass backs out any incomplete (uncommitted) transactions.

Lazy Commit

Lazy commit is important is an important feature of NTFS. It allows NTFS to minimize the cost of logging to maintain high performance.

Lazy commit is similar to lazy write. Instead of using resources to mark a transaction as successfully completed as soon as it is performed, the commitment information is cached and written to the log as a background process. If the power source and/or computer system should fail before the commit is logged, NTFS will recheck the transaction to see whether it was successfully completed. If NTFS cannot guarantee that the transaction was completed successfully, it backs out the transaction. No incomplete modifications to the volume are allowed.

Periodic Log File Checkpoints

Every few seconds, NTFS checks the cache to determine the status of the lazy writer and marks the status as a checkpoint in the log. If the system crashes subsequent to that checkpoint, the system knows to back up to that checkpoint for recovery. This method provides for more expedient recovery times by saving the amount of queries that are required during recovery.

Note This level of recoverability protects metadata. User data can still be corrupted in the case of power and/or system failure.

Fault Tolerance and NTFS

When used on a partition on a single device, NTFS can recover from any type of system crash, yet it may lose data as the result of an I/O error. In conjunction with the mirroring or parity striping support implemented by the Fault Tolerance (FT) driver, NTFS can survive any single point of failure.

NTFS also supports bad cluster replacement. When a sector goes bad on a single drive, NTFS will migrate the cluster containing the sector to the Bad Cluster File (so that it will not be used again), but the data on that cluster will be lost. The cluster is reallocated, but written with all -1's.

However, when NTFS is used on a fault-tolerant device and an error is detected on one copy of a cluster, data can be recovered. The bad cluster is migrated to the Bad Cluster File, and it is replaced by another cluster. Then a copy of the original data is written to the new cluster.

Note NTFS supports cluster sizes of 512, 1024, 2048, and 4096.

For more information about the Windows NT fault-tolerance capabilities, see Chapter 6, "Fault Tolerance Features of Windows NT."

File System Security

NTFS is the only truly secure file system for use with Windows NT. It implements Microsoft's corporate security architecture for discretionary access control.

With NTFS, every file has a security descriptor containing an owner field and an access control list (ACL). The ACL is made of access control entries (ACEs), bit fields which define access permissions for an individual or a group. For each action a user attempts, the action is checked against the appropriate ACE for that file.

No other Windows NT file systems support ACLs.

For more information about ACLs and ACEs, see Chapter 3, "Windows NT Security Model."

Creating an NTFS Volume

There are three ways to create an NTFS volume.

- The boot partition can be made an NTFS volume during the installation process.
- The existing volume can be formatted to NTFS, thus eliminating the existing files in the partition.
- The partition can also be converted to NTFS. This leaves the existing files intact.

Windows NT includes two tools—Format and Convert—that you can use to create NTFS volumes after the Windows NT setup program is run.

With Format, you can format a partition as an NTFS volume. To do this, use the standard Format command and specify the /FS:NTFS option. This destroys all existing files on the partition.

With Convert, you can convert an existing partition from FAT or HPFS to NTFS without losing data. Type the Convert command using this form:

```
convert x: \fs:ntfs
```

Convert can be used on the boot partition as well as on secondary partitions. However, the Convert utility cannot convert the open system disk. Instead, if you specify the active partition, an entry is added to the Registry so that the Convert utility is run the next time the system is booted.

Note that Convert only converts to NTFS, not to other file systems, such as FAT or HPFS.

For more information on creating an NTFS volume, see Appendix A.

Summary

NTFS takes the best parts of both FAT and HPFS and improves upon those designs. From FAT, NTFS borrowed the " is that simplicity yields performance" philosophy. Performance increase when the number of disk transfers for common operations is minimized. From HPFS, NTFS borrowed techniques for speed and flexibility. For example, NTFS uses Btrees similar to those used by HPFS to maximize performance.

NTFS supports both long and short (eight-plus-three) filenames for compatibility with MS-DOS, HPFS, and other networked clients including OS/2, UNIX, AppleShare®, and NFS. NTFS also provides for multiple extended attributes and allows future applications to define other extended attributes.

NTFS offers data security on fixed and removable disks, a feature important to corporate users and other power users.

In addition, it provides a recovery system that is more reliable than either FAT or HPFS. NTFS also meets POSIX requirements.

The following table compares some of the features of the FAT, HPFS, and NTFS file systems as implemented on Windows NT:

Table 5.3 Comparative Details for FAT, HPFS, and NTFS

	FAT file system	High Performance File System	NT File System
Filename	Eight-plus-three ASCII characters (one "." delimiter allowed)	255 bytes of double-byte characters (multiple "." delimiters allowed)	255 Unicode characters (multiple "." delimiters allowed)
File size	2^{32} bytes	2^{32} bytes	2^{64} bytes
Partition	2^{32} bytes	2^{41} bytes	2^{64} bytes
Maximum path length	64	No limit	No limit
Attributes	Only a few bit flags	Bit flags plus up to 64K of extended-attribute information	Everything, including data, is treated as a file attribute
Directories	Unsorted	Btree	Btree
Philosophy	Simple	Fast and powerful	Fast, recoverable, and secure
Built-in security features	No	No	Yes

CHAPTER 6

Fault Tolerance Features of Windows NT

What will it cost your organization if the file is lost? Another question—namely, What is the cost if a file is read or manipulated by the wrong person?— is the flip side of the security question:

The costs may be tangible or intangible. For example, if a word processing file containing a memo about the company picnic five years ago is lost the cost could be zero. If the memo was for this year's picnic and had to be re-entered, the cost would be the labor cost of rekeying. If the memo was a response to a bid which is due tomorrow the cost becomes time-sensitive. The momentary cost is the cost of labor, but the real cost is the profits of the contract if the file cannot be re-entered in time to make the bid closing.

Fault tolerance and data recovery are techniques used to minimize the cost of lost data while attempting to limit the cost of resources needed.

A number of resources can be deployed minimize the cost of lost data. These tools include uninterruptable power supply services, tape backup, disk mirroring, disk duplexing, and striping with parity. This chapter includes a discussion of each of these tools and their uses. In addition, it covers volume sets and striping without parity. Volume sets do not enhance fault tolerance in the system, but are closely related to the disk organization features we will be talking about.

Uninterruptable Power Supply

Unlike LAN Manager for OS/2, both Windows NT and Windows NT Advanced Server include Uninterruptible Power Supply (UPS) services as part of the base operating system. Including them in the base operating system allows for easier access to the file systems in the event of a power problem.

UPS systems allow the computer to continue to function from battery reserves for a short period of time in the event of a power failure. They also protect from both power sags (brown-outs) and power spikes (typically experienced when the power returns). The UPS service provided with Windows NT is sensitive to signals from the UPS unit and performs orderly shutdowns of applications, services, and file systems if the UPS power is depleted.

A UPS system provides two levels of protection. First it keeps the computer active during power outages of short and or medium duration. This allows the computer to continuing processing data without time loss. It also allows the computer to be shut down in an orderly fashion before the battery power is fully depleted. The main advantages is to allow the file system to flush its internal cache buffers to the disk before the power to the computer is lost.

As with other components in a distributed system, connectivity during a power outage is only as good as the weakest link. Placing UPS units on a computer will keep that computer up and active. If the computer is a server, then the server will remain up. If the computer is a workstation, it will remain active. However if there is no UPS on hubs, concentrators, routers, or bridges, the workstation and the server may still be unable to communicate with other network nodes. Planning which pieces of the environment should be protected by a UPS device depends on the expectation of its use. If the UPS is only used to prevent data lost at the server, then placing a UPS system only on the server is enough. If the expectation is to be able to continue with business at some level, then having some number of workstations and printers on UPS may also be needed. If there are electrically active components in the network, they too may have to be on a UPS to accomplish the business goals during power outages.

How UPS Works

All power is run through the UPS system, where the battery is constantly being charged. When there is a power sag or a power failure, the battery automatically begins providing power to the computer, and the appropriate communications line is raised. This alerts the UPS service to take the appropriate action based on the configuration.

Within the UPS industry there is a standard for connecting UPS units to a COM port on a computer. This connection has 3 pins—the CTS pin, the DCD pin, and the DTR pin. The actual pin number used depends on the type of connector (DB-25 or DB-9). Pins are "signaled" when a problem is detected. Signaling can be either setting the pin high (from its normal low state) or setting the pin low (from its normal high state) depending on the needs of the UPS device being used.

The CTS pin (5 on a DB-25, or 8 on a DB-9) is used to used to signal a power failure. When the power coming into the UPS device is lost or sags below a threshold, this pin is signaled.

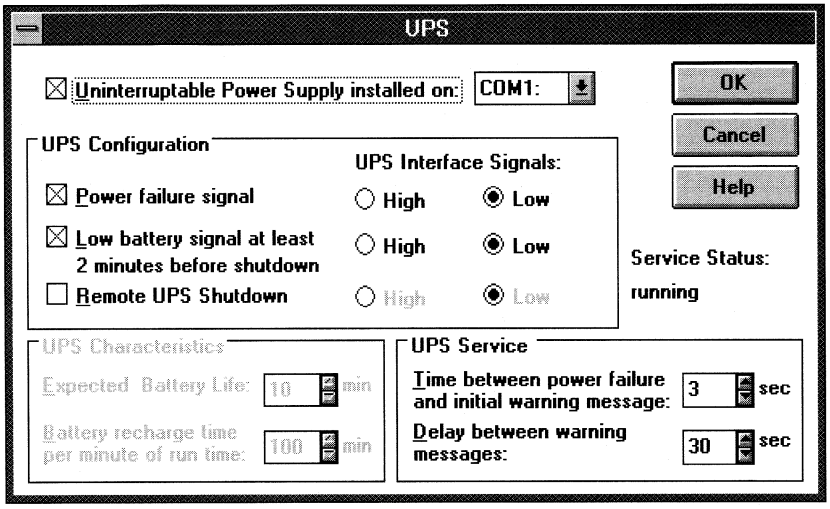
The DCD pin (8 on a DB-25, or 1 on a DB-9) is signaled when the battery is about to run out of power. When this pin is signaled Windows NT expects to have two minutes to perform the needed shutdown activities.

Finally, the UPS service can communicate with the UPS device via the DTR pin (20 on a DB-25, or 4 on a DB-9). This pin is signaled to inform the UPS device to shut down.

(Two other pins are provide for support of contact closure type UPSs.)

UPS Control Panel

The UPS control panel controls the configuration of the interface between the UPS device and the UPS service.



The UPS service is selected to be run by using the check box at the top of the screen. Though you'll notice it says "installed," it's actually always installed. This really means that the service is marked for automatic startup during the boot sequence. The COM port is selected from list box. The signals generated by the UPS and the values on those pins to indicate it is being signaled is provided.

If the UPS device can generate the power failure signal, then certain additional characteristics can be entered. First is the life expectancy and the battery recharge time. This informs Windows NT the expected amount of time the battery can be expected to keep the system up. This also informs it the amount of time it will take to recharge the battery to that level. These fields are not enabled if the UPS has a low battery signal, since the UPS device can inform the UPS service directly.

If the UPS can signal immediately that there was a power failure, it is advisable that someone be alerted to the problems. The amount of time between the power failure and the initial alert can be set. Thus, if the power failure is only for a short amount of time, no error messages are generated. If the power failure is persistent, the frequency of additional warning messages can be set.

Running and Starting the UPS Service

The first time the UPS service is activated, it is marked as an automatically started service. The UPS service does not, however, start the UPS at that time. The UPS Control Panel prompts the user to start the service via the Service Manager Control Panel. The computer does not have to be rebooted to start the service. Once the service is marked as automatic, it will be started during subsequent system boots.

The UPS service runs as a background task. When the power failure (CTS) pin is signaled, the initial warning timer is started. If the pin is not "unsignalled" by the timer expires, the initial warning message is sent. This message tells the user a failure has occurred and the computer is now running on backup. When the low battery (DCD) pin is signaled, or when the life expectancy of the battery has expired, the system automatically begins shutdown procedures.

Tape Backup

First line of defense for fault tolerance is a good tape backup scheme. Windows NT provides a full-function tape backup system based on the Maynard tape system. This is a departure of the OS/2 tape system, Sytos Plus®. This new tape system was developed because of internal limitations imposed by the Sytos system. Note, however, that tapes prepared by the Sytos system are readable by the Windows NT Maynard system.

The tape strategy which is right for you depends greatly on the volume of data being backed up and the frequency of the modifications being made to the files.

There are three types of three tape backup schemes—normal, incremental, and differential. Let's take a look at each one.

Normal backups means that the selected files are always copied to backup tape and that each file is marked as having been backed up. This gives users the ability to restore files quickly from the most recent tape. However, this scheme increase the time to make the backups since even files which have not changed since the last backup are copied to the tape.

Incremental backups means that only those files which were changed since the last normal or incremental backup are written to the backup tape. Each file is marked as backed up once copied. This scheme saves time during the backup process. If users combine normal backups with incremental backups, restoring requires users to start with the last normal backup and work forward through all the incremental tapes.

The last scheme is differential backups. This means that only those files which were changed since the last normal or incremental backup are written to backup tape. The difference is that the files are not marked as backed up. Combining normal backups with differential backups means that restoration only needs the last normal and the last differential backup tapes.

Tape Rotation

Rotating set of tapes used during the backup process is a very common practice. It helps reduce the cost of tape backup by reusing tapes. And, compared to the cost of recovering data, tapes are inexpensive.

Rotation schemes are as individual as the needs of the organization. One of the most popular tape rotation schedules is to use 19 tapes over the course of one year. This schedule uses four tapes Monday through Thursday. These are incremental or differential backups. Every Friday a normal backup is created. The tapes used Monday through Thursday are recycled to be used again each week. The Friday tapes are kept during the month. On the last Friday of each month, two normal backups are created. One stays in the monthly rotation, the other goes to offsite storage. The tapes in offsite storage stay for one year, then rotates back into the cycle to backup the same month they were used for last year.

With the new tape system there is some new terminology. (You'll see these terms used in the user interface screens.) The first new term, *backup set*, refers the set of files, directories, or drives selected for a single backup operation. The *family set* is the set of tapes the backup set resides on. This term is used in the case of backup sets which required one or more tapes, though usually it is not used unless more than one tape is involved. Information describing the backup set is stored in the *catalog*. The catalog is always stored on the last or only tape in the family set.

Backup and Restore Security

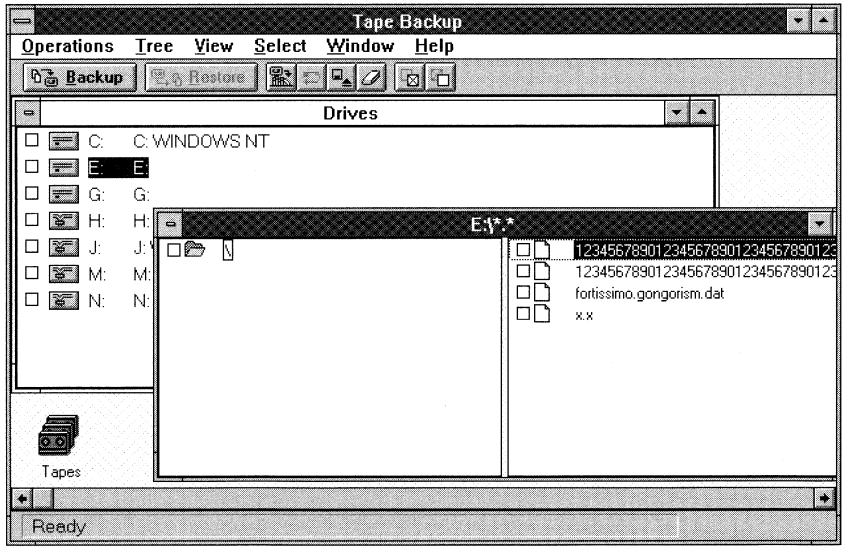
Tape backup and restoration are special events with special permissions. Users with read and write permissions may certainly backup and restore certain files. Quite often, however, there will be a special user known as the tape operator (called *backup operator* for Window NT) who will be called upon to create a backup or restore a file from tape. The backup operator has the privilege to backup and restore files for which they have no read or write permissions. By

adding their user IDs to the Backup Operator group, users are recognized as designated backup operators.

Peer services greatly eases the burden of backing up remote computers. If the computer is running Windows NT or Windows for Workgroups, a centralized backup server can backup up remote computers simply by establishing a NET USE relationship, then backing up the files using the redirected drive letter. The shared directory on the remote computer may either an entire drive or a subdirectory.

If the remote computer does not have sharing capabilities, then the workstation must establish a relationship with either the backup server or some other server. Once the connection is established, the workstation's data is copied to the server. Once on the server, the backup server can either back the data up from a local disk, or establish a relationship to the intermediary server and back up the data.

All the tape backup activities are done through the tape backup utility. This is a very simple user interface. Simply click on the files you want to back up and select the backup options.



The Backup dialog box allows you to specify a tape name, backup set description, and the type of backup. The type corresponds to normal, incremental, or differential options described earlier. The backup can also produce a log of all tape activities.

After selecting which tape(s), backup set(s), or files(s) to restore, choose the Restore command to display the Restore Information dialog box. In this dialog box, the first section provides information about the backup sets(s). For each backup set, you must specify the drive to which you want the information

restored. You may also specify an alternate directory path. As with backups, a log is maintained for restored files.

Using Tape for Data Transfer

Tapes not only make good backup and recovery media, they also make excellent data transfer facilities.

Consider the following scenario:

> Your organization has two sites located across at opposite ends of town. On a daily basis, Site A has 500MB of data to send to Site B. The low-entry cost solution would be to use switched-line modems. Assuming the line stayed up, at 19,200 bps, it would take 7.6 hours per day to transfer the data.
>
> A more timely solution would be to send a courier across town to deliver a tape containing the data. Even in the worst city traffic, the courier probably could be there in an hour.

Offsite Storage and Disaster Recovery

Depending on your organization's Electronic Data Processing auditing requirements, or laws and regulations of your industry, you may be required to keep a set of backup tapes offsite. This allows you organization to pick up the pieces in the event of a catastrophe such as a fire, earthquake, or severe weather. In addition, tapes will most likely play a key roll in re-establishing a disaster recovery site for your organization.

A disaster recovery site is away from normal operational site and is prepared with the equipment needed to run the organization. In the event of a disaster which cripples the normal operational site, the last backup tapes are loaded at the recovery site and the organization continues processing.

For businesses where disaster recovery procedures are mandated or expected, they should be exercised regularly to ensure that the current staff is well trained.

Windows NT Disk Protection

Windows NT employs several strategies for ensuring the protection of data on disk. Most of these strategies are based on *RAID*, which stands for Redundant Arrays of Inexpensive Disks. RAID was initiated in 1987 by Patterson, Gibson, and Kratz of the University of California at Berkeley, who drew together various options for building virtually-large drives from smaller, inexpensive drives. This notion is counter to the SLED (Single Large Expensive Disk) concepts commonly used on mainframes and mini-computers.

The Berkeley paper establishes five different schemes for deploying disks. It is a common mistake to think one builds on top of another. Each, including the following, has their advantages and disadvantages:

- RAID 1 represents disk mirroring.
- RAID 5 represents striping with parity.

RAID 0 has since been added by the industry to represent striping without parity. (In Windows NT, this is known as *striped sets.*) Windows NT supports RAID 0, 1, and 5 with an additional variant of RAID 1 in disk duplexing.

Before going on, let's clarify our terms:

A *disk* is a physical component of hardware. It may be divided into one or more *partitions*, which are logical subsections of the disk.

A *drive* is one or more partitions logically associated to a single system identifier (drive letter). Many computers have one hard disk, with one partition and thus a single drive C. Windows NT can still do this, but it can do so much more. A *redirected drive* is the logical association of a single system identifier (drive letter) to a network sharepoint.

There are two special types of partitions in Windows NT—the boot partition and the load partition.

The *boot partition* is the partition which the ROM BIOS selected, from which the computer's operating environment will begin loading. This is also called the "active partition" by certain programs, such as FDISK.

The *load partition* contains the operating system—Windows NT or another—being booted. This is the partition that the BOOT.INI file points to. This is sometimes called the "*sysroot*" in reference to the internal environment variable which points to this path.

In Windows NT, the boot partition and the load partition can be the same, can be on different disks, and/or on different controllers. They may be formatted as FAT, HPFS, or NTFS. They may be mirrored or duplexed. However, these partitions may not be striped, parity striped, or defined as volume sets.

Updating Windows NT and Disks

Disk are handled differently in Windows NT than in other systems. In Windows NT each disk is given a unique signature. This signature is stored in the Master Boot Record area on the disk (physical sector 0). This signature is used by Windows NT to identify disks. It allows disks to be moved from controller to controller, or within a SCSI chain without problems. The signature is used to look up the disk and the partitions on those disks in the Registry.

In the Registry there is an entry called DISK which informs Windows NT of the participation of each partition on each disk. This eliminates reliance on hidden information on the disk. It does however present a minor problem, since DISK information is stored in the Registry and the Registry is overwritten when a new version of Windows NT is installed. The Disk Administrator utility provides a simple solution to this problem.

In Disk Administrator, the Partition includes a menu item called Configuration. It offers three options—Save, Restore, and Migrate.

Save allows you to save the DISK information to a floppy disk. It does this by saving a current copy of the system component of the Registry.

Restore allow the DISK information to be restored from this floppy disk.

Migrate is used if both the SYSTEM file from the previous version of Windows NT is available on disk. The current disks are scanned for previous version(s) of Windows NT and the user is allowed to select from which director to load the DISK information. Only the DISK information is loaded from the old SYSTEM file.

Note Activities done with Disk Administrator—such as creating or deleting partitions, breaking mirrors, building sets—are not committed to the disks until the user tries to exit Disk Administrator. At this point, the user is given one last chance to confirm that the modifications made are correct.

Disk Mirroring

Disk mirroring is a method which protects against hard disk failure. As illustrated in Figure 6.1, disk mirroring uses two partitions on different drives connected to the same disk controller. All data on the first (primary) partition is mirrored automatically onto the secondary partition. Thus, if the primary disk fails, no data is lost. Instead, the partition on the secondary disk is used.

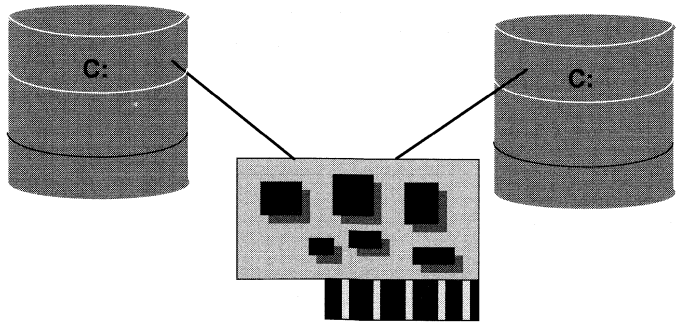


Figure 6.1 Disk mirroring

Mirroring does not have to be done at the drive level. Thus, unallocated space on the primary drive can be allocated into a logical drive, as can any unallocated space on the secondary drive. With disk mirroring, both partitions have the same drive letter.

Any file system—including FAT, HPFS, and NTFS—can make use of disk mirroring.

Mirroring is not restricted to a partition of identical size, number of tracks and cylinders, and so on, to the primary partition . This eliminates the problem of acquiring an identical model drive to replace a failed drive when an entire drive is being mirrored.

For practical purposes, though, the mirrored partitions will usually be created to be the same size as the primary partition. The mirrored partition cannot be smaller. However, if the mirrored partition is larger than the primary, the extra space will be wasted.

Disk Duplexing

Disk duplexing is simply a mirrored pair with an additional adapter on the secondary drive. This provides fault tolerance for both disk and controller failure. (The use of multiple adapters connecting to one drive is not supported.) In addition to providing fault tolerance, this also has the potential of improving performance.

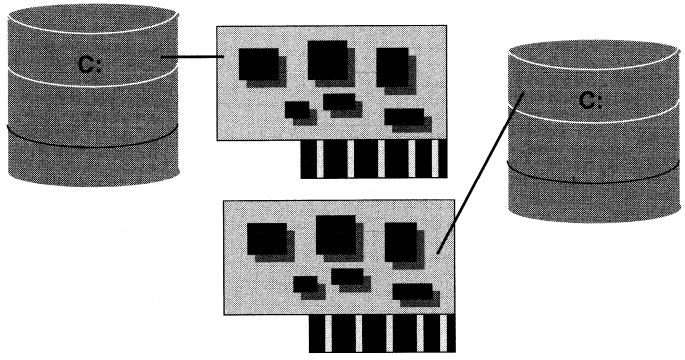


Figure 6.2 Disk duplexing

Like mirroring, duplexing is also performed at the partition level. To the Windows NT operating system, there is no difference between mirroring and duplexing. It is simply a matter of where the other partition can be found.

Creating Mirrored and Duplex Sets

The processes for creating mirrored sets and for creating duplex sets are identical. Each can be done by using the Disk Administrator utility by following these steps:

1. From the Disk Administrator, select the partitions that you want to duplicate and an area of free space of the same size or greater on another hard disk.
2. To do this, while holding down the Ctrl key, first select the partition to be mirrored, then select the free space on another disk
3. From the Disk Administrator menu, select Fault Tolerance.
4. Choose Establish Mirror.

The Disk Administrator creates an equal sized partition in the free space for the mirror and assigns the drive letter to the set.

Disk Volumes

Disk volumes (also called disk spanning) is not a fault tolerance strategy. Rather, it is simply a way of combining multiple partitions into a single logical disk. This does not increase disk performance.

Volume sets are created using the Disk Administrator utility by following these steps:

1. Select the first free partition, then hold down the Ctrl key and select the others.
2. Select the Create Volume Set option from the Partition menu.

Once a volume set is created, it can also be extended using the Create Extended Volume Set option. Only volume sets formatted with NTFS can be extended. A volume set cannot contain mirrored or striped components in it composition.

Disk Striping

Disk Striping (sometimes called Raid 0) is a method which can increase both read and write performance since multiple I/O commands can be active on the drives at the same time.

A striped set must have between 2 and 32 disks. If the disks are different sizes, the smallest is used as the common partition size. The remaining free space may be used individually or in a volume set.

Note that this method is different than the Windows NT Advanced Server method called "disk striping with parity" (described later in this chapter). Disk striping as applies to Windows NT provides no parity stripe. Because the parity stripe is not present, the set is not fault tolerant. Once a stripe has been lost there is no way to recover it. Some disk drivers do provide in drive hot-fix capabilities which can be used to help ensure the safety of the data.

Creation of a striped set is done through the Disk Administrator utility by following these steps:

1. Select the disks for the set.
2. Select the Create Stripe Set option under the Partition menu.

Detecting Errors in Volumes and Striped Sets

As mentioned before, volumes and striped sets are not fault tolerant. Once an error is detected the set is not recoverable. If the problem is detected during system initialization, a severe error will be logged to the event log and the volume set or striped set will not be available for use by the system.

Disk Striping with Parity

Note Disk striping with parity is a feature available with Windows NT Advanced Server. The information in these last few pages of the chapter is included to provide as part of the "big picture" of fault tolerance.

Disk striping with parity is a method based on concepts put forth in RAID 5. Here, a number of different partitions on different disks are combined to make one large logical drive. The partitions are used and arranged in such a way to ensure multiple single points of failure in the array.

There must be at least three disks and no more than 32 disks in a striped set. A partition of approximately the same size must be selected from each disk. The disks may be on the same or different controllers. SCSI disks are best since advanced recovery features such as bad block remapping can be used during the recovery process. Data is written in stripes across all partitions in the set. In addition to the data a parity stripe is written interleaved with the data stripes. The parity stripe is simply a byte parity of the data stripes at a given stripe level or row.

For example, suppose you have five disks in the striped set. At level 0, you have stripe block 0 on disk 0, 1 or 1, 2 on 2, and 3 on 3 and the parity (eXclusive OR) of the stripe blocks on disk 4. The size of the stripes (striping factor) is currently 64K. The size of the parity stripe is the size of the data stripes. On the next row the parity stripe is on disk 0. Data is on the rest of the disks. Because the parity

stripes are not all on the same disk (as in RAID 4) there is no single point of failure for the set.

When using any of the fault tolerant disk schemes, Windows NT uses a device driver called FTDISK.SYS to receive commands and respond appropriately based on the type of fault tolerance that is being used. Thus when the file system generates a request to read a given section of a file, the normal disk system receives the request from the file system and passes it to the FTDISK.SYS driver. This driver then determines the stripe the data is in. From this and the information on the number of disks in the set the disk and location on the disk is located. The data is read into memory. Striping can actually increase read performance since each disk in the set can have an outstanding read at the same time.

Writing to a parity striped set is a little more difficult. First the original data from the stripe that is to be written must be read along with the parity information for that stripe level. The differences to the parity information are calculated. The differences are added to the parity stripe. Finally both the parity and the new information are written to disks. The reads and the writes can be issues concurrently since they must be on different disks, by design.

Fault Tolerance With Parity Striping

There are two general cases of fault tolerance with parity striping.

First is a data stripe is no longer readable. Even though the data stripe is no longer readable the system may still function. When the bad data stripe is to be read, all of the remaining good data stripes are read along with the parity stripe. Each data stripe is subtracted (XORed) from the parity strip, the order isn't important. The result is the missing data stripe. Writing is a little more complicated but works very much the same way. All the data stripes are read and backed out of the parity stripe leaving the missing data stripe. The modifications needed to the parity stripe can now be calculated and made. Since the system knows the data stripe is bad it is not written; only the parity stripe is written.

The other general case is the loss of a parity stripe. During data reads this does not present a problem. The parity stripe is not used during normal reads. Writes become much less complicated as well. Since there is no way to maintain the parity stripe the writes behave as a data stripe write without parity. The parity stripe can be recalculated during regeneration.

Identifying When a Set Is Broken

The process of error detection and recovery is very similar for both mirrored sets and parity striped sets. The exact system response to the problem will depend on when the problem occurred.

A *broken set* is defined as any time one or the other partitions in a mirrored or duplexed set cannot be written, or any time a stripe can no longer be written.

When an I/O error is first detected, the system performs some routines in an attempt to keep the set from breaking. The system's first priority is to try reassigning the sector which failed. This is done by issuing the a command to remap the sector to the disk.

Windows NT will only attempt remapping if the disk is supported by a SCSI controller. SCSI devices are designed to support the concept of remapping. This is why SCSI devices work better as fault tolerant devices. Some ESDI devices also support the concept of remapping, but there is no standard for the command.

If the disk does not support sector mapping, or if the other attempts to maintain the set fails, a high severity error is logged to the event log.

The partition which has failed is called an *orphan*. It is important to note that if the process of orphaning a partition does not occur during a read; only during writes. This is because the read cannot possibly affect the data on the disks, so performing orphan processing would be superfluous.

During system initialization, if the system cannot locate each partition in a mirrored set, a severe error is recorded in the event log and the remaining partition of the mirror is used. If the partition is part of a parity striped set, a severe error is recorded in the event log and the partition is marked as orphan. The system then continues to function using the fault tolerant capabilities inherent in such sets.

If all of the partitions within a set cannot be located, the drive is not activated, but the partitions are not marked orphan. This save the recovery time of simple problems like disconnecting the SCSI chain from the computer.

Recovering Orphans

When a partition is marked as orphan, the system will continue processing until a replacement disk or partition is available to recover from the problem and ensure fault tolerance again. A set with an orphan is not fault tolerant. Another failure in the set can, and most likely will, cause the loss of data.

Recovery procedures should be performed as soon as the problem is discovered. To recover, follow these steps:

1. Break the mirror-set relationship using the Break Mirror option within the Disk Administrator utility.
2. This will convert the remaining active partition of the set into an "normal" partition. This partition receives the drive letter of the set. The orphan partition receives the next available drive letter.
3. You can then create a new set relationship with existing free space on another disk in the local computer, or replace the orphan drive and re-establish the relationship with space from this disk.
4. Once the relationship is established, restart the system.
5. During the system initialization, the data from the original good partition will be copied over to the new mirrored partition.

When a member of a parity striped set is orphaned, it can be regenerated from the remaining data. This uses the same logic discussed earlier for the dynamic regeneration of data from the parity and remaining stripes. Select a new free space area that is as large as the other members in the set. Then choose the Regenerate command from the Fault Tolerance menu. When the system is restarted, the missing stripes will be recalculated and written to the new space provided.

CHAPTER 7

Windows NT Installation and Configuration

Windows NT minimum hardware requirements are a 386/25 megahertz computer with 8MB of RAM. To install Windows NT and set aside a paging file of approximately 20MB, you will need approximately 75MB of free disk space. Your system must also include a VGA (640x480) video display adapter and a floppy drive A. A CD-ROM drive is optional.

Note This chapter addresses installation considerations on the Intel *x*86 platforms. Though the techniques for installing on R4000-based computers are very similar, the specifics are not covered in this chapter.

This chapter introduces the overall installation and configuration of Windows NT. It presents the system requirements and the tradeoffs related to installation and configuration. It also describes the steps involved in the installation procedures, and identifies key information required during setup. Finally, this chapter describes how to configure Windows NT for your specific work environment.

Upgrade or Boot Loader?

Before discussing the Windows NT setup process, let's consider upgrade issues with Windows NT.

Windows NT has a Boot Loader that allows the user to boot either the previously installed operating system (such as MS-DOS or OS/2), or Windows NT. Initially, you may like the idea of keeping your old Windows 3.1 or Windows for Workgroups installation on your computer as you move to this new, powerful platform. However, please note that there are dependencies that your Windows-based applications have installed in the old Windows environment, but may not be apparent in Windows NT on a computer where the Boot Loader is installed.

For example, the Program Manager groups that you have installed in Windows 3.1 will not be automatically available in Windows NT if you do not upgrade Windows 3.1 to Windows NT.

Three Methods for Setup

Windows NT has three different options for installation:

- Installing Windows NT from a floppy or CD-ROM
- Installing over a network
- Installing by using the Computer Profile Setup

Computer Profile Setup is designed for large corporate environments where there are a number of identical computer platforms. Computer Profile Setup is designed for a specific set of customers and is provided in the *Windows NT Resource Kit.*

The next section focuses on the standard installation process. WINNT and Computer Profile Setup options are described later in this chapter.

Standard Setup

The standard method for installing Windows NT is from floppies or from a CD-ROM. The Setup boot floppy disk accompanies the CD-ROM installation and is also Disk 1 of the floppy-based installation. This disk contains the key files needed to start Setup, detection mechanisms for existence of SCSI adapters on the computer, the Setup application itself, and the instruction routine to Setup (called TXTSETUP.INF). The *.SYS files are the drivers for the most popular SCSI devices installed.

Hint If you know you have a SCSI device that is not supported on the Setup boot floppy, you can copy a Windows NT device driver for your device onto the Setup boot floppy to allow Windows NT to install from CD-ROM.

To allow Windows NT Setup to create the Emergency Repair Disk, Setup requires a floppy disk drive A. This disk is useful in recovering from damage caused to a Windows NT installation. For example, suppose someone accidentally deletes a system file or runs the SYS utility on the boot drive, wiping out the Windows NT Boot Loader. The Emergency Repair Disk can be used to rectify these situations.

To begin Setup, insert the boot floppy in drive A and reboot the computer. Setup is completed in two main phases—text-mode setup and graphical-mode setup. Between these two phases, you are prompted to reboot your computer. Each phase is described below.

Text-Mode Setup

When you begin Setup, the boot floppy runs the Setup Loader file (SETUPLDR), which in turn calls the NTDETECT code to identify the computer's hardware configuration.

Next, you are prompted for the level of customization (Express or Custom Setup) you want to perform. In general, Express setup is preferred. If you want to monitor each step of the installation, select Custom Setup. Custom Setup is preferable for computers that contain "special" or "100%-compatible" components. The major difference between Express and Custom Setup is how much information is presented to the user and the options to override the auto-detection of the hardware devices.

Hardware and Software Detection

Once you have selected your installation technique, Windows NT Setup will search your computer for any attached SCSI devices (therefore checking for a CD-ROM-based installation). If one is not located, Setup continues checking your hardware configuration for the machine type, display adapter type, mouse, and keyboard.

Installing Windows NT Files

Next, Setup asks for information about where the Windows NT software will be installed. It displays disk configuration information with the option to create or delete partitions on the disks it detects. You can also change file systems on the partition where you want Windows NT installed. For example, if the partition is presently formatted as a FAT partition, you can choose to convert to NTFS or keep the existing file system intact.

Important If you choose an option to *reformat* (rather than convert) from one file system to another, this *does* destroy the files. If you choose to *convert* a file system to NTFS, Setup does not destroy the existing files. Please pay attention when selecting your option.

The next step is for Setup to run the CHKDSK utility, which verifies that the destination disk is good condition.

Then Setup searches the disk for any previous version of Windows. If Setup finds a previous installation, Setup will recommend that you upgrade the installation. Optionally, you can type "N" for New Path and provide Setup with a path for the Windows NT software (\WINNT is the path Setup recommends).

Setup installs the core Windows NT files in the destination path. These core files are enough to boot the computer and bring up graphical mode Setup, where installation continues. In addition, it updates the Registry information to allow Windows NT to boot.

Reboot

At this point, you are prompted to remove the Setup floppy from drive A and reboot the computer.

When the computer reboots, you might catch a glimpse of the Boot Loader before Windows NT begins to load. The delay time on the Boot Loader has been purposely set to zero during the installation of Windows NT to complete the install process. The computer displays a blue screen reporting Windows NT is being loaded, and lists the startup devices being loaded in the system.

Graphical-Mode Setup

Next, Setup displays a graphical-mode screen and prompts you for your name and the name of your company. A second dialog box prompts for the name of your computer (15 characters or less).

Note The computername must not be the same as the domain name.

At this point, if you selected Custom Setup, you can choose which components of Windows NT you would like to install. From the displayed list, choose whether or not you want to install network support, setup printers, and/or setup applications already installed on the hard disk.

Virtual Memory Options for Custom Setup

If you selected Custom Setup, a configuration screen for virtual memory is displayed. *Virtual memory* is memory that Windows NT allocates on the hard disk for use as an extension of the computer's physical memory. You can specify the destination drive where a paging file will be created for virtual memory. Setup displays the space available on the disk, the minimum size for the file, and its recommended size. Typically, the minimum size is about 20MB, with the recommended size being between 20 and 25MB. The recommended size is automatically displayed in the Size Chosen field. You can override that entry with any number between the minimum size and the space-available size.

If you specified Express Setup, the size of the virtual memory paging file is automatically set to the recommended size.

Note The Windows NT virtual memory paging file is named PAGEFILE.SYS and is stored at the root level of the selected drive. Windows NT does not use the Windows 3.1 virtual memory files. These swap files may coexist, but are not shared by the different versions.

Printer Setup

The next step is to install a printer. Express Setup requests that you configure a default printer, or specify that you have none to configure. For Custom Setup, you may have chosen earlier to skip this step.

Windows NT will install printer drivers to support both local and non-Windows NT shared printers. If you have a printer shared by a Windows NT computer, it is not necessary to have a locally-installed printer driver on the client workstations. You only need one on the Windows NT computer that is sharing the printer.

Therefore, if you are going to be printing to a locally-attached printer or to a printer connected to a non-Windows NT computer, then select the appropriate printer and configure for the corresponding port.

Note If you intend to print to a non-Windows NT network printer, you should temporarily assign the printer to LPT1, until the network components are installed, then redirect the printer port to the network print share via the LocalPort option.

Network Setup

If you elected to install Network support, Setup will proceed with installing the necessary network components.

Note If at some future point a network card causes the auto-detection code to hang, you can run the Custom Installation to bypass the auto-detection code.

Windows NT Setup automatically detects your network card. For Custom Setup, this information is displayed so that you can approve or change it. The information displayed for a network card typically includes that I/O port address, the memory address setting for the card, and the IRQ setting.

Note The default settings may not be correct for your card. Be sure to pay attention to the fields in the dialog box.

Setup completes network setup by installing the network services necessary to support your network card. It also installs the default workstation and server components and necessary support files.

Next, the Network Control Panel displays with information about your network hardware and software configuration. When you click OK, Setup runs the binding process.

Domain Setup

Windows NT allows you to join either a workgroup or a domain.

- A *workgroup* is a set of server and workstations grouped together for efficient browsing and peer-to-peer resource sharing.

- A *domain* is similar in function to a workgroup, but can include dedicated servers and has the additional feature of security, limiting access to strategic computing resources and data.

If you join a workgroup, you will be prompted for your username and password.

If you join a domain, you are prompted for a domain name. If no account for your computer exists on the domain, you must provide an administrator name and password to create a new account for your computer.

Next, whether you join a workgroup or a domain, Setup prompts you for a password for the computer's Administrator account.

User Interface Setup

The last pieces to install on your hard disk relate to the user interface. Setup now creates several Program Manager groups—namely Main, Accessories, Administrative Tools, Games, and Startup.

Then, if you selected the option, or are using Express Setup, all applications Setup can locate on your local hard disk(s) are installed with icons in Program Manager. (However, this is not true if you install Windows NT into the directory formerly used by Windows 3.1.)

Next, Setup creates the Emergency Repair Disk. Setup carefully checks that you didn't accidentally put the Setup disk back into drive A, then formats the floppy disk, and save the default configuration information necessary to restore Windows NT on the disk.

Important The Emergency Repair Disks *are not interchangeable* with other systems! The directory structure and Registry information—including computername—is stored on this disk, so it needs to be left with the specific computer. Placing the disk in disk sleeve and taping the sleeve to the main chassis cover for the PC is not a bad idea.

Finally, Setup will ask you for the time zone setting. Unless you happen to live in the Greenwich Mean time zone, you will need to select the correct time zone for your PC.

Setup is now complete. To prepare to boot Windows NT, remove the Emergency Repair Disk from drive A, and reboot.

The next two sections briefly describe the two other Setup options—WINNT and Computer Profile Setup.

WINNT Setup

As mentioned in the beginning of this chapter, WINNT Setup is a second option for installing Windows NT. This method allows you to install Windows NT across MS-DOS-supported network. The WINNT option is very handy for large corporate installations, and allows common sharepoints to be used for wide distribution. This process is similar to the **setup /n** option in Windows 3.1, in that it takes all of the files from the floppy disks and/or CD-ROM and places them in one large directory on a server sharepoint. For example, the command line which instructs Setup to create the file layout is as follows:

```
SETUP -n -i initial.inf -s <source_path> -d <destination_path>
```

Note This command *must* be run from within Windows NT, therefore you must install Windows NT on at least one computer before you can create the WINNT sharepoint.

Now, let's take a look what happens when you use WINNT Setup.

The first thing WINNT Setup does is to create a Setup boot floppy, just like the boot floppy used for the CD-ROM boot disk, or Disk 1 of the floppy disk set. Next, WINNT downloads a complete Windows NT installation source onto the local partition, and places it in the WIN_NT.~LS directory.

Then, WINNT Setup instructs you to leave the boot floppy in drive A, and reboot the computer. Windows NT Setup then runs just like the standard installation described earlier. However, instead of copying the files from a floppy disk or CD-ROM, Setup simply moves the files on the hard disk into the appropriate destination directory that you specify.

WINNT Setup has several benefits. First, it is a relatively fast way to install Windows NT. Due to generally faster access times to the hard disk than CD-ROM, WINNT Setup is typically faster than sharing the CD-ROM drive on the network. With WINNT, the files are transferred across the network, and during the local Setup, files are simply moved on the hard disk. Also, multiple computers can run WINNT Setup simultaneously; you don't have to shuttle the floppy disks from computer to computer.

WINNT Setup does not require a Windows NT-supported network. So even if you do not yet have drivers for your network, you can install Windows NT. Note, however, that Windows NT won't see such a network once it is installed.

Finally, if you customized the Windows NT installation files, all customizations can be done in one common place.

Computer Profile Setup

A third setup option—perfect for very large corporations—is called Computer Profile Setup.

In many large corporate environments, you typically find large installed bases of identically-configured computers resulting from a long-term contract or bulk purchase. Large number of identically configured equipment allows for fewer spare parts, easier troubleshooting, and allows the user to easily move from computer to computer with no performance loss.

The Computer Profile Setup option addresses these large corporate environments, allow rapid installation of Windows NT on these computers. This option is only valid for computers that have MS-DOS already installed. This utility is included in the *Windows NT Resource Kit.*

There are two main parts to the Computer Profile Setup utility—Create Profile and Install Profile.

The Create Profile utility is run on a computer that is preconfigured with all of the appropriate user information, including user database, Program Manager groups, hardware drivers, and so on. Create Profile automatically extracts this information and place it in a secure file with the filename extension .CPS. This file also includes all installation files and is used as a template for that specific computer type.

Install Profile takes a specific template that you provide and installs the Window NT configuration on the target computer. To provide the necessary unique identification for the target computer, you will need to provide the domain name (if appropriate) and the computername.

Now that we have described each of the ways you can install Windows NT, let's take a look at what happens when you reboot your computer after installation.

Windows NT Boot Loader

After Setup is completed, you are prompted to reboot your computer. At that time, it either boots directly into Windows NT, if there was no other operating system previously installed, or the Windows NT Boot Loader displays a list of boot options. From this list, you can select which operating system you want to boot. If you select Windows NT, that operating system boots and displays the Windows NT logon screen, then prompts you to press Ctrl+Alt+Del to log on to the system.

Note This key sequence, Ctrl+Alt+Del, is significant because it provides for security from the beginning of the operating system's initialization. In the history of computers, so-called "Trojan Horse" applications have been developed to defeat system security. These applications look like the system logon screen, but instead are only there to garnish your username and password.

Windows NT avoids that problem by using the System Reset key sequence such that if you press Ctrl+Alt+Del, any MS-DOS-based Trojan horse is rebooted.

After pressing Ctrl+Alt+Del key sequence, you are prompted to log on to a domain account, or for a workgroup member, to a Local User account. The dialog box looks the same in either case; it simply depends on the way the computer was configured during installation.

Once you have logged onto the system, you should check out the configuration. The next section describes how.

Configuring Windows NT for Users

If you have successfully logged onto Windows NT, you should set up the necessary user accounts if there are to be multiple users accessing the computer. For a computer in a domain, you can set up accounts for domain users. For a workgroup computer, you can configure local user accounts.

To set up a new user account, use the User Manager and select New User from the User menu.

Another way you may want to configure Windows NT for users is to customize the groups in the Program Manager. Windows NT has two different kind of Program Manager groups—the Common Group and the Personal Group. The Common Group is available to all users on the computer (unless explicitly denied access). The Personal Groups are user-specific, allowing each user can have his or her own set of personal groups in the Program Manager. This means that even though the computer may be used by many different individuals throughout the day, since each logging on under his or her username, the Windows NT desktop can be customized for that individual.

You can use the User Profile Editor to configure Windows NT for users. This is a handy utility that network administrators can use to help users focus on the intended tasks by simply removing the user's temptation to navigate around on the hard disk or to spend valuable time playing games. With the User Profile Editor, you can set restrictions on specific users. For example, you can restrict access to certain Program Manager capabilities, such as changing group entries. Or you might restrict access to the File Run menu option so that users cannot run programs other than those in their program groups. Similarly, you can lock or unlock access to specific Program Manager groups.

To apply user profiles created with User Profile Editor, use the User Manager utility. With User Manager, you can specify which local groups the user is a member of. You can also specify the user profile path (to the profile created in the User Profile Editor), as well as the a logon script to run when the user logs onto the system, and the user's home directory. User Manager is also useful when setting the user account information related to the automatic expiration of the account, or when you want to manually enable or disable the account.

The next section describes how to use the Event Viewer utility to help troubleshoot any problems you may encounter while installing or configuring Windows NT.

Troubleshooting Hardware Configuration Problems

The Event Viewer is the key utility to use to determine if any system, security, or application errors have occurred in the system. This utility is located in the Administrative Tools Group. The Event Viewer sequentially lists any errors the system may have encountered. To inspect the error more closely, simply double-click on an entry and a dialog box with further details displays.

As you search for a specific event, be sure to note the time of an event. Some of the entries may be from the previous day or a previous execution of Windows NT. If you are working to isolate a series of problems on a specific computer, you may want to clear the Event Log after you have worked through a specific error to allow yourself to focus on the current errors. It is also worth noting that you can save the event log to a file if you wish to give a copy of the log to an administrator, or Helpdesk personnel.

Possible Problems

Windows NT operating system is designed to optimize software control of the computer and the related hardware peripherals. Windows NT reads the computer configuration information and uses that information to ascertain hardware configurations. However, you may run into a case where Windows NT does not detect something about your configuration.

At Microsoft, as we loaded new versions of the product during the development cycle onto new computers, we noticed three typical problems:

- Windows NT could not see all the memory on the computer
- No access to the network
- Problems with different video cards or configurations

Let's look at each of these cases.

Suppose you install Windows NT and it does not detect the full amount of memory in your computer. One typical reason why this would happen is that the OEM configuration utilities was not run on the computers since the memory upgrade. On EISA, MCA, and on some ISA computers, you should run the configuration utility (such as the EISA configuration utility, the PS/2® Reference Disk, or the CMOS setup program).

If after you install Windows NT, you cannot access the network, first check to see that other parts of the network are functional. Next, check the network adapter card, paying close attention to the interrupt setting, the I/O address, and any jumper settings on the card. Check to see whether you have any potential interrupt conflicts with any other devices in the system. Then, double-check to make sure that the driver you installed for the card is the most appropriate driver for that specific adapter.

Video-related problems typically occur on computers that did not run Windows 3.1 previously. This is because users who have previously installed Windows typically know what type of display adapter is loaded in the system. However, Windows NT does perform additional hardware checking on display adapters beyond the checking performed in the Windows 3.1 product. If you have video problems, make sure the card has the memory necessary to run at the specified resolution and that the monitor can support that resolution. Then double-check any switch settings on the card to make sure the card is correctly configured.

Interrupt Conflicts

In the previous section, we touched on the issue of interrupts. Interrupt conflicts are the most common conflicts on a personal computer. At this point, let's briefly look at how to avoid interrupt setting conflicts.

First, it is important to note that certain hardware configurations that worked okay under MS-DOS, are not guaranteed to work under Windows NT. It is because Windows NT is a more advanced operating system, demanding higher performance of the hardware devices in the system. So in a case where you may have been able to get away with hardware conflicts in your system under MS-DOS or Windows, Windows NT could report a warning that a conflict exists with your hardware devices. This is not an error in the software. Rather, Windows NT wants to be able to communicate effectively with the devices. Any contention for these devices might indicate that characters or data might be lost or that a device may not work correctly.

To avoid interrupt conflicts like these, note that there are the common IRQ settings for typical devices in the system.

For example, the network card likes to have an assigned interrupt. On some network cards use hardware jumpers to specify interrupt settings. Newer cards let you set their interrupt via software configuration.

Other set interrupts include the following:

- The PS/2 mouse port (not specific to PS/2 computers) uses IRQ 12.
- Typical SCSI devices use IRQ 11.
- Sound boards may either use one or two interrupts, typical IRQ settings are 5, 10, and various others—see adapter manual.
- The COM ports use and share interrupts—COM2 and COM4 use IRQ 3; COM1 and COM3 use IRQ 4.

I/O Address Conflicts

Second to interrupt conflicts, I/O address problems are the most common conflicts on personal computers. The Event Viewer is a very handy tool for identifying I/O address conflicts between two devices. In most cases, the Event Viewer will even tell you which two devices are in contention.

Typically, the I/O address contention is between the video card and either the SCSI adapter or the network adapter card. In many cases, due to the additional functionality of Windows NT, you may find yourself adding hardware to your system to take advantage of the new feature, such as CD-ROM support.

EISA and MCA Computer Issues

Windows NT reads the computer configuration information only if it is available. To make sure the configuration information is up to date for an EISA or MCA computer, you may want to run the computer manufacturer's configuration utility. (Two examples of these are EISA configuration utility and the PS/2 Reference Disk utility.)

The manufacturer's configuration utility often tells you which interrupts are in use in the system and which interrupts are available. Make a note of these.

If you have an ISA card in an EISA computer, you may choose to create a configuration file for the ISA card, so the EISA configuration utility can detect if the ISA card settings conflict with any of the other cards in the system.

ISA Computer Issues

If you have an ISA computer, you will want to make sure that the CMOS setup utility accurately displays the current computer configuration. For example, if you decide to use IRQ 3 for a network card, we advise that you disable COM2 in the CMOS configuration to avoid any possible conflicts. Again, though this never posed a problem for MS-DOS and Windows, Windows NT gives control to the devices on a first-come-first-served basis, so whichever device gets registered with the interrupt first gets the interrupt.

When you are running the CMOS setup utility, also note the Shadow RAM addresses in use. Hardware devices will not be able to use that address space while Shadow RAM is enabled.

This utility also shows information relating to EMS page frame. This information is not significant for running Windows NT.

For further troubleshooting suggestions, see Chapter 11, "Troubleshooting."

CHAPTER 8

Windows NT Boot Sequence

Windows NT includes a multiple-stage boot process. Some of these stages are specific to Windows NT, some are common to all Intel- and ARC-based computers. This chapter describes the boot sequence in the Intel architecture and what happens during each phase of the boot process.

This chapter also examines the structures in the Registry which affect booting and some steps for troubleshooting problems once Windows NT is installed.

Power On Self Test

When an Intel-based computer is first powered on, the CPU is reset to run in real memory mode and the instruction pointer is positioned to 0xffff:0x0000. This is the beginning of the Power On Self Test (POST) routine. The POST routine is responsible for determining the amount of real memory, determining if the needed hardware components (such as the keyboard) are present, and allowing other adapter cards to initialize.

What appears on the screen during the POST routine is determined by the ROM developer and the adapter card manufacturers. Normally however, the POST routine is identified by an incrementing tally of the amount of memory found in the computer.

Once the computer has run its POST routine, each adapter card is allowed to run its own POST routine.

Determining the Boot Drive and Partition

At the end of the POST routine processing the CPU is instructed to process the commands at software interrupt 19 hexadecimal (Int 19h). This is the reboot-computer interrupt. The routine attempts to locate the boot device by first checking drive A. If no disk is found in drive A, then drive C is checked.

Determining if drive C is a boot device is a two step process. The first step is to read the first sector of a hard disk, which contains the Master Boot Record (MBR). This record is loaded into memory. The MBR contains a program which is run. The second step is to scan the remaining portion of the MBR, which contains the partition table. The MBR program locates the partition which has the "active partition" flag in the partition table.

The Boot Record

After the active partition is located, its boot record is loaded into memory.

The boot record starts at the first logical sector in the partition. In a FAT-formatted partition, the boot record is only one sector long. In HPFS, the boot record is sixteen sectors long.

The program in the boot record loads the first layer of program needed to start the operating system. This program is different for each operating system. When Windows NT is installed on your hard disk, the boot record is changed in support of the new operating system. The new boot record is instructed to load a program called NTLDR which is located in the root directory of drive C.

If by chance no active partition is found on the drive C, processing passes to the ROM BASIC software interrupt processor, interrupt 18 hexadecimal (Int 18h). It's been some time since most computers actually had ROM BASIC at this address. However, the hook is still very useful. In remote boot situations, where the operating system image is being loaded from a server, this interrupt is redirected to the ROM on the network adapter card by the POST processing routine on the network adapter card.

Note The initial release of Windows NT cannot be remotely booted in this manner.

Once the boot record is found, it is loaded from the disk into memory and executed. The boot record is instructed to find the NTLDR program in the root directory, load it into memory and run it. You can tell the when NTLDR starts executing because the screen will be cleared and the Boot Loader banner will appear.

NTLDR and NTDETECT.COM

NTLDR is a new program with Windows NT and is very special. When it is first loaded the memory of the computer is still in real mode (using the old-style segments-and-offsets addressing mechanisms). The first task of NTLDR is to switch the memory into 32-bit flat mode. Once that is accomplished, the appropriate "Mini file system" is initialized.

Mini file systems are special versions of the FAT, HPFS, and NTFS file systems. The mini file systems are part of the NTLDR module. NTLDR must load this component so that it can easily gain access to other files on the disk which may be in different directories, and so on.

BOOT.INI

Once the NTDETECT has built the hardware components list, it reads in the BOOT.INI file. This is the file used to build the operating system selection menu on Intel computers. The BOOT.INI file must be in the root of the partition from which the NTLDR program was loaded.

BOOT.INI is made of two parts:

- The boot loader section contains the default operating system which will be booted if another is not selected, and the number of seconds the user has to make that selection.
- The operating systems section contains a list of operating systems which may be booted.

Each line in the operating section has three fields, as in the following example:

```
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\winnt="Windows NT"/NODEBUG
C:\"MS-DOS"
```

The first field is before the equal sign. This tells NTLDR when the image for the operating system is located. This field may be coded in two different ways. The first method is the normal MS-DOS format—drive letter, colon, slash, and so on. The second method is the ARC format. Either format may be used on an Intel-based computer.

The second field is the string after the equal sign. This field must be in quotation marks and can be any literal constant. This is the text displayed in the menu from which the user makes a selection.

Windows NT supports a third, and optional, field on the line. Currently Windows NT uses this field to support the debug option. If the field is coded as /DEBUG or is not present Windows NT will run in Debug Mode. If the field is coded as /NODEBUG, it won't. The /NODEBUG option should be used for all user environments.

Note Debug mode is not useful for most user environments. In fact, it is useful for only a very few developer environments, mainly where device drivers are developed. The debug mode will allow another Windows NT computer connected via a modem cable to "break" into the Windows NT kernel and view memory, execute instructions, and so on. The debugging computer must also be running Windows NT.

Possible Problems During the Boot Loader Phase

Let's look at some possible problems which may occur in the boot sequence before Windows NT is started.

- If the NTLDR program was deleted from the hard disk, the following message is displayed and the system frozen, waiting for a disk with the NTLDR and a reboot.

  BOOT: Couldn't find NTLDR
  Please insert another disk

  The easiest way of avoiding this problem is by marking the NTLDR file as a system file so that the user cannot delete the file in the first place.

- If NTDETECT.COM was deleted from the hard disk the following message.

  Error opening NTDETECT
  Press any key to continue

  Again, the easy way to avoid this problem is by marking the NTDETECT.COM file as a system file.

- The time-out option in the Boot loader section is set to 0. NTLDR will not wait for the user to select an option, but boots directly into the default operating system.

  This may actually be viewed as a feature by some. For example, the user always gets Windows NT, but some other operating system, such as MS-DOS, is available for emergencies or support technicians.

- The path of the default operating system in the Boot Loader section is different than any in the operating system section and a new line will appear at the bottom of the menu selection.

  This is an indication that some inappropriate modification may have occurred to the BOOT.INI file.

- The device or partition component of a path is incorrect and the following message displays when the operating system is booted.

  OS Loader V2.10
  The system did not load because of a computer disk hardware configuration problem.
  Could not read from the selected boot disk. Check boot path and disk hardware.
  Please check the Windows NT™ documentation about hardware disk configuration and your hardware reference manuals for additional information. Boot failed.

  Correct the path of the operating system you were attempting to boot and try again.

- The directory component of the path is incorrect and the following message displays when the operating system is booted.

  OS Loader V2.10
  loading file mult(0)disk(0)rdisk(0)partition(1)\winnx\system\ntoskrnl.exe
  The system did not load because it cannot find the following file:
  <*sysroot*>\SYSTEM32\NTOSKRNL.EXE
  Please re-install a copy of the above file.
  Boot failed.

  (The line "load file ...." may be different from the line you receive.) The problem indicates one of two things. First, the BOOT.INI file was modified and will need to be repaired. If the path is OK, however, NTOSKRNL.EXE may have been erased while running under a different operating system. In this case, the path was changed from PARTITION(1)\WINNT to PARTITION(1)\WINNX.

- You try to boot the alternate operating system which uses a file called BOOTSECT.DOS (even if booting OS/2 1.*x*). This file is a hidden system file. However, if it is not found on the root directory it the following message will be displayed.

  Couldn't open boot sector file
  multi(0)disk(0)rdisk(0)partition(1)\bootsect.dos

  Getting a lost boot sector back is difficult since there is information about the physical layout of the hard disk it is associated with. If you have another computer with the *exact* disk layout (tracks, heads, cylinders, sectors) and operating system configuration, you may try putting its BOOTSECT.DOS file on the damaged computer.

Warning This is not without its potential for failure, which could make the disk totally unreadable. Do not attempt this recovery method without detailed knowledge of low-level disk structures (MBR) and binary editing tools.

- The BOOT.INI file is not found in the same directory as NTLDR and the following message appears.

  OS Loader V2.10
  loading file multi(0)disk(0)rdisk(0)partition(1)\winnt\system\ntoskrnl.exe
  The system did not load because it cannot find the following file:
  ***<sysroot>*\SYSTEM\NTOSKRNL.EXE.**
  Please re-install a copy of the above file.
  Boot failed.

  Make sure BOOT.INI and NTLDR are both in the system root directory.

Booting Another Operating System

Before we talk about the boot process for Windows NT in detail, we want to understand what occurs if MS-DOS or OS/2 is booted.

If the alternate operating system is to be booted, a hidden file called BOOTSECT.DOS (always called .DOS) is loaded into memory at location 0x700:0xC000. This is the location where boot records are loaded. The processor is then returned to real mode processing. Next, NTLDR jumps to the start of the boot sector program and processing continues as if this were the original boot sector on the disk.

Booting Windows NT

Once the Windows NT operating system is selected, NTLDR calls another program in the rood directory, NTDETECT.COM.

The NTDETECT program examines the type of hardware components of the computer and builds a list. The list includes processor type and version (or stepping), existence of a coprocessor, the bus type, video adapter type, keyboard type, mouse device, and several other items. This program is only needed on Intel-based computers. On MIPS processors, the needed information is passed to the NTLDR program from the ARC firmware.

The first program loaded is NTOSKRNL.EXE. Since the processor is already in 32-bit flat memory model mode, this is done the same as loading any other program. When the program is started, the hardware information collected by NTDETECT.COM is passed to it. This explains the error we see when starting Windows NT if the NTDETECT.COM has been deleted from the disk.

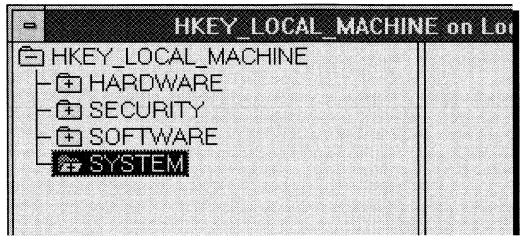
Once the NTOSKRNL.EXE is loaded, the Hardware Abstraction Layer (HAL) loads. This component isolates Windows NT from any platform specific hardware issues. (For more information about HAL, see Chapter 1, "Windows NT Architecture.")

Boot Instructions in the Registry

The other component which is loaded is called the System hive. This is a component of the Windows NT Registry which contains, among other things, the information needed to load the appropriate device drivers, subsystems, and services for this installation. (As these three modules load, messages display that information on the screen.)

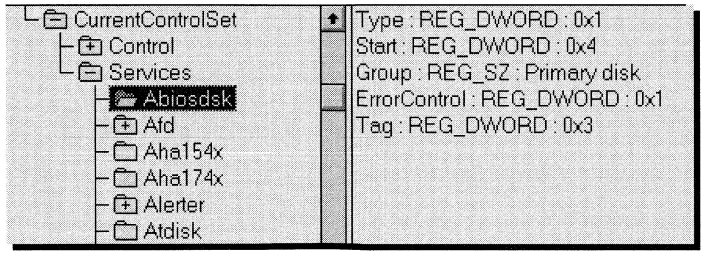
Once the System hive is loaded, all booting instructions are read from it. You can use the Registry Editor (REGEDIT.EXE) program to view these instructions.

In the Registry, the major key is called HKEY_LOCAL_MACHINE. Under the SYSTEM subkey there is a further series of subkeys named ControlSet*nnn*(of which there may be more than one), Clone, and CurrentControlSet.



A *control sets* is a booting sequence. One of the ControlSet*nnn* keys is used during the boot process to bring up Windows NT. The control set which is actually running is called CurrentControlSet. The Clone key contains a workspace used in the Registry during the boot process.

Within each Control Set there are a list of controls and services. *Services* describe components which can be loaded. *Controls* determine what components are loaded, in what sequence, and what additional activities are performed during the boot sequence. The boot sequence will make several passes through these list during the booting process.



Each service has several common data items associated with it. Three of importance are the Type, the Start, and the Group. The values of these data items control if and when a service is loaded, and in what sequence.

For more information about the Windows NT Registry, see Chapter 4, "Windows NT Configuration Registry."

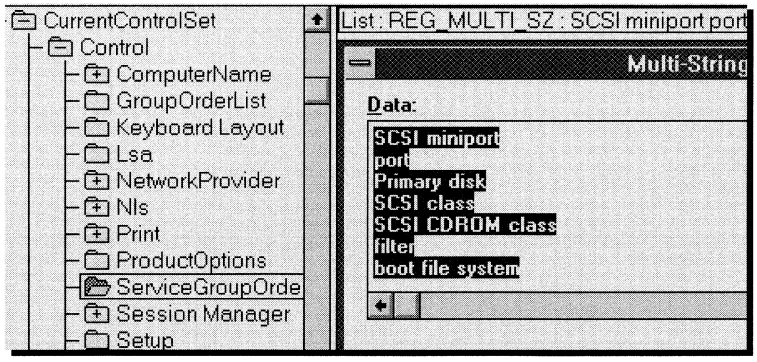
Lowest-Level Device Drivers

The first time the System hive is scanned, the boot process looks for services with type value of 0x1 and a start value of 0. The type value 0x1 indicates it is a kernel device driver. A start value of 0 indicates it is to be loaded but not initialized *before* the actual Windows NT Kernel is initialized.

Services loaded during this phase are typically the lowest-level hard disk device drivers. Commonly loaded in this phase are the drivers called ATDISK, FTDISK, and SCSIDISK. These drivers are loaded via BIOS calls. Having these drivers available improves booting time as more and more files are read from the disk. These drivers are loaded into memory, but are not initialized until all are present and the Kernel is initialized.

The drivers are not simply read into memory in the sequence they are found in the Registry. Associated with each is a data item called Group. Group is a text string which identifies which group the driver belongs to.

The list of services is scanned and a list of all services to be loaded is prepared. The value in the Group data item is used as a key to sort this list before services are actually loaded. The sorting is done based on the position of the value of the Group in the Control\ServiceGroupOrder\List. The following illustration shows an example. From the list, services with a group of "SCSI miniport" will be loaded first, followed by "port," then "Primary disk," and so on.



Initialization

Once all of the device drivers from this list are loaded, the Kernel is initialized. This stage is recognizable because the screen is cleared, and then painted blue. The Windows NT Kernel signature is displayed on the top line of the screen.

Once the Kernel is initialized, the loaded device drivers are initialized. If a problem occurs during initialization, the system will take action based on the ErrorControl value in the Registry for this driver. These values are discussed in detail later in this chapter.

Load Next Level

After initialization, the Registry is scanned again, this time for device drivers with a start value of 0x1. As with the device drivers loaded earlier, a list of devices is created, then sorted by Group value.

There are three differences between this loading and the first:

- These drivers are loaded via the device drivers loaded earlier, not via BIOS calls.
- Each of these driver is loaded and initialized, instead of loading all drivers then waiting for initialization. This is done to allow the drivers to discard unneeded initialization sections.
- If there is an error while these device drivers are loaded, the system response is determined by the ErrorControl data item associated with the service.

The ErrorControl data item contains three different values:

- "Normal" (value 0x1) instructs Windows NT to ignore the error and continue the boot process. Most device drivers have an ErrorControl of Normal.
- "Severe" (value 0x2) is a little more complex. In the event of an error, the boot process is to switch over to the LastKnownGood control set, then restart the boot process. (LastKnownGood is discussed later in this chapter.) If the control set being booted is LastKnownGood, the error is ignored and processing continues.
- "Critical" (value 0x3) is very much like Severe, however if the control set being booted is the LastKnownGood, the boot process is stopped and a failure message is displayed.

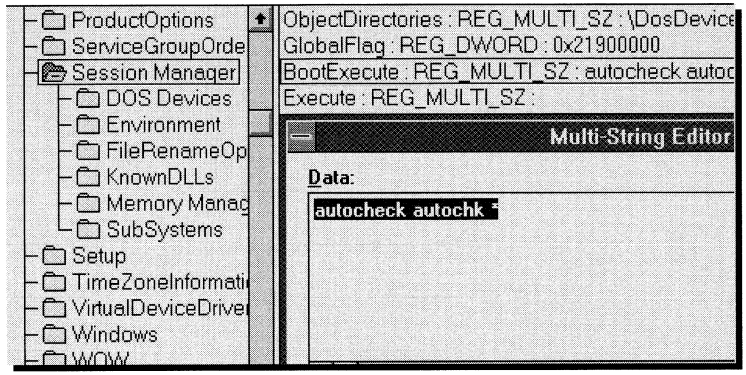
Once all of the drivers are loaded and initialized the Kernel does some housecleaning. Freeing memory blocks used by the device drivers, initializing more data structures.

The next step is to load the Session Manager.

Session Manager

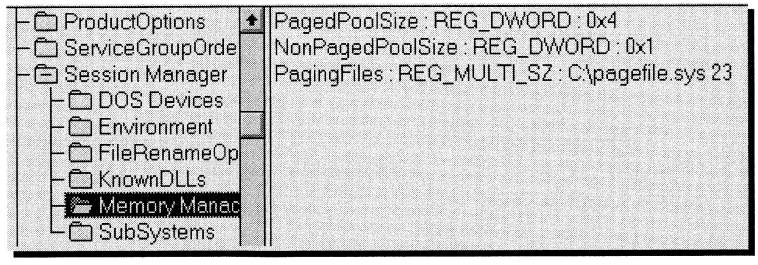
Now that the Kernel has its house in order, the boot process can start higher-order subsystems and services. This is done by the Session Manager (SMSS.EXE).

Information for the Session Manager is located in the Registry in the Control section of the ControlSet under Session Manager, as shown by the following illustration. Once loaded and started, the Session Manager finds the BootExecute data item.

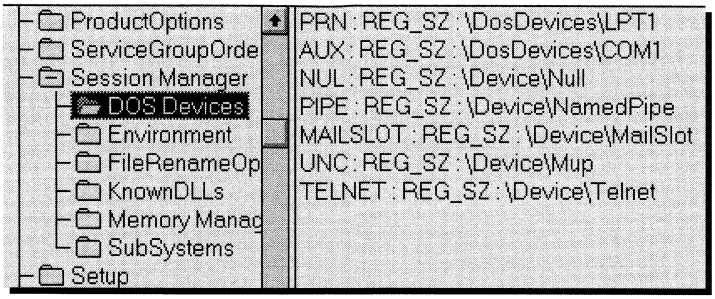


The BootExec data item contains one or more commands to be run before any services are loaded. This is typically AUTOCHECK, which is the Windows NT version of CHKDSK. Results of AUTOCHECK's activities on the screen are a clue to where you are in the boot sequence.

Once the disks have been checked, the Session Manager creates the page files needed by the Virtual Memory Manager. (For more information about what the Virtual Memory Manager does, see Chapter 1, "Windows NT Architecture.") Information for this file is located in the Memory Manager section under the Session Manager in the Registry.

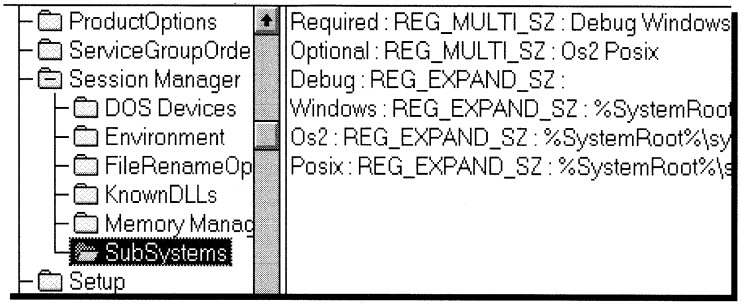


Once the Memory Manager's needs are satisfied, symbolic links are created. These links are used to direct certain classes of commands to the correct component in the file system. A list of these symbolic links are located in the DOS Devices section of the Session Manager.



Loading Subsystems

At this point, the subsystems are loaded. These are listed in the Required data item of the SubSystems section under Session Manager.



Because of the messaging architecture of the subsystems, the Windows subsystem must be included. This is because all access to the video screen is controlled via this subsystem. The Windows subsystem process name is CSRSS. (This name is also used by the Performance Meter utility.)

The Windows subsystem automatically starts the WinLogon process. WinLogon is starts a number of subsystems vital to running Windows NT. A list of these services is found in the System data item in the WinLogon section under the Software\Microsoft\WindowsNT section in the Registry.

Loading Services

The last phase of the boot process includes loading services. After loading the subsystems and WinLogon, the boot process displays a "Welcome" screen. Then it loads and initializes the Services Controller, and makes one last scan of the Registry.

This time, the boot process searches for all services with a start type of 0x2 and loads them. The loading sequence, however, is different than with the device drivers. Instead of using Group order, these are loaded according to dependencies. Each service specifies which services and/or groups it depends on. The earlier device drivers were loaded serially, one after another, so the Group order was sufficient to ensure each driver was loaded correctly.

The Service Manager starts services in parallel services, and must do so without violating current dependencies. There are two different types values for services loaded by the Service Manager.

- Type 0x10 services allow only one service to run in a given process space. This allow the same executable file (.EXE) to be spawned multiple times with different names, each receiving a different process space. This is the more common service type. The EventLogger is one example.
- Type 0x20 services have multiple services packaged in a single executable file (.EXE). When additional services are started one or more new threads are created for the service, but no new process is created. The LanmanWorkstation and LanmanServer services are examples of services which share the same process resources.

Once the type 0x1 and 0x2 services are loaded, Windows NT is ready for the user to logon and begin work.

Two Other Service Types

There are two additional start values used in the Registry.

- Type 0x3 are used for services which may be manually started by the user. These services may be started through the Services Control Panel, or by typing the following at the command prompt:

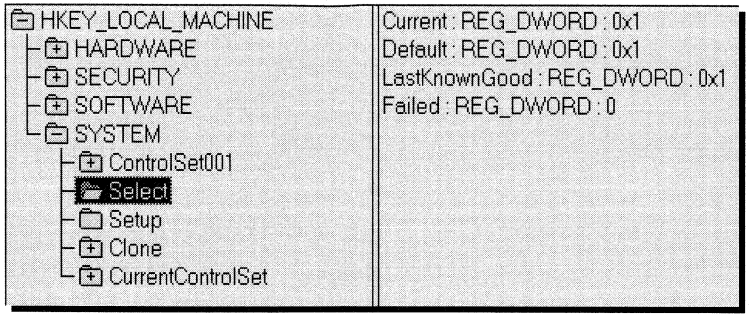
  ```
  net start service_name
  ```

- Type 0x4 are services which are never started by the system or by the user.

LastKnownGood

Finally, we want to include some notes about a special Registry entry called LastKnownGood.

This entry is found in the System\Select key. This is the first section read by the NTOSKRNL when it is preparing to scan the Registry.



System\Select includes several data items, including Default and LastKnownGood.

Default tells the system which control set to boot from. For example, if Default is 0x1, the system is booted from ControlSet001.

The LastKnownGood control set is updated after the services have completed loading.

In general, when a user select Windows NT from the Boot Loader menu, the system loads the Default control set. However, there are two situations in which the system will attempt to load the LastKnownGood control set instead

- If the system is recovering from a Severe or Critical device driver loading error.
- If the your asks Windows NT to load the LastKnownGood.

In addition, the user can ask for the LastKnownGood by holding down the space bar immediately after selecting the Windows NT operating system from the Boot loader menu. In this case, the system displays a prompt asking the user to select between Default and LastKnownGood.

CHAPTER 9

Mail

Windows NT includes an electronic mail application, Mail, that can be used to exchange information with other Windows NT workstations. Mail also can work interactively with many other Windows-based applications. This chapter describes the components that make up Mail and explains how to use mail functionality from within other Windows-based applications such as Microsoft Word for Windows and Microsoft Excel for Windows. This chapter also describes four customization features you can use to tailor Mail for your office—custom commands, custom message types, the Messaging Application Program Interface (MAPI), and user information templates.

About Mail

The Mail application provided with Windows NT has a client side, a mail-server side, and an interface between them. The client side includes a visual user interface, made up of viewers for messages, folders, and address lists. The server simply contains a directory structure known as the *postoffice* and has no programmatic components. The interface between client and server manages message storage and retrieval, name validation, and directory access.

Mail uses a "store and forward" design. A user sends mail to, and receives mail from, a *message store* on his or her own computer. As a user sends a message, it is forwarded from the local computer's message store to the *postoffice* located on the mail server. The postoffice has a mailbox for each user, giving users access to the messages they've received when they log on to Mail.

Mail is based on the "shared file system," meaning the postoffice can reside on any computer in the workgroup running Windows NT in 386 enhanced mode. As mentioned before, the postoffice requires no extra programmatic components. Instead, all postoffice file manipulation is handled by the Mail client.

The postoffice is a directory structure in which the main directory is called WGPO. The structure of the postoffice is efficient because Mail only stores one copy of each mail message, even when a message is addressed to multiple recipients. For example, if you sent a 75K message to 10 people, Mail stores only 75K of information on the postoffice, not 750K. The postoffice is a temporary message store, holding the message until the recipient's workstation retrieves it. When it is retrieved, the message is removed from the postoffice.

The following sections describe the features of the mail client, the server, and the interface between them.

The Mail Client

Mail is a small but robust tool. Its easy-to-use and versatile user interface includes these features:

- Rich message content supported by the mail editor
- Easy addressing supported by Mail's address books
- Flexible message organization supported by the folder filing system
- The ability to work online or offline

These features are readily available to users so they can customize their own work environment. Later in this chapter, we'll discuss some other advanced techniques that you might use as the workgroup administrator.

This section describes each feature in detail and shows how you can use these features to make your job as workgroup administrator easier.

Rich Message Content

Mail includes a text editor that lets you create messages using a variety of typefaces. It also features automatic word wrapping and tab support. The best part of Mail messages is that you can include the contents of other files—formatting and all—in your message. You can exploit any or all of these options when you use custom message types to create special message forms for your workgroup. (See the "Custom Message Types" section later in this chapter for more information.)

You can use all of these techniques to create a message:

- Type text directly. Mail includes a text editor with automatic word wrapping, a proportional font, and tab support.
- Embed objects, such as pictures or spreadsheets, from other applications.
- Attach documents to appear as icons in the message.

Figure 9.1 shows a message with text, an attachment, and an embedded object.

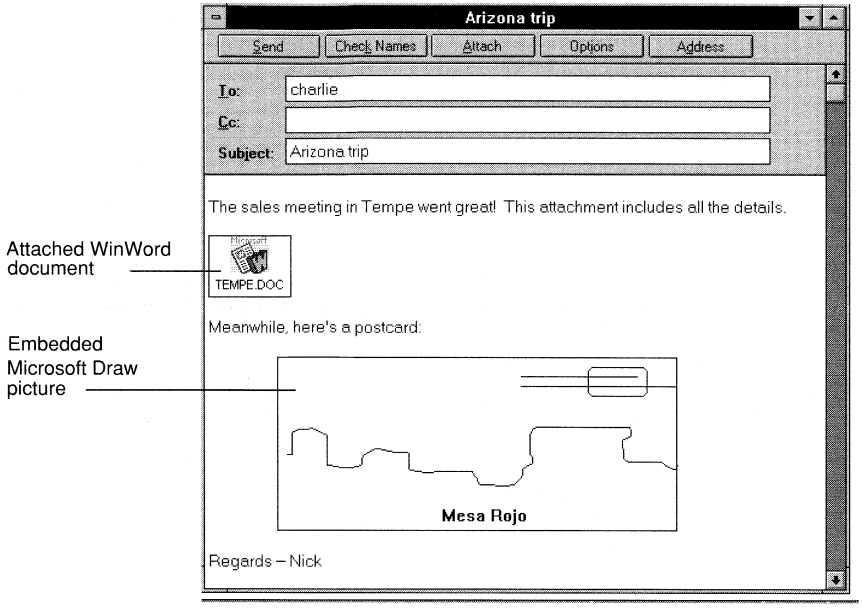


Figure 9.1 You can attach documents and embed objects in your mail messages

You can have multiple compose forms open at the same time, making it simple to share text, objects, and file attachments between multiple messages. Mail also supports cutting and pasting between messages and from other Windows-based applications.

Attached files are represented in the message by their icons. Recipients using the new Mail client can launch attachments by double-clicking the file attachment icon (assuming the recipient has the appropriate application associated to the attachment). Besides launching attachments from the mail message, the recipient can save them to disk.

Because Mail supports object linking and embedding (OLE) you can embed objects in mail messages. Typical objects might include part of an Excel spreadsheet, part of a richly-formatted Word for Windows document, or a vibrant graphics object. Objects can be pasted into any location in the message and can easily be moved to another location and/or re-sized. Message recipients can manipulate and edit the embedded object without leaving Mail so long as they have the appropriate associated application for that object installed on their systems.

Easy Addressing

Mail users don't have to memorize the sometimes cryptic e-mail aliases for everyone in your workgroup. Instead, they can look up a person's real name in Mail's address book to identify the recipient for a message. The address book also enables users to track other information about workgroup members, such as office number and phone number.

The address book is provided by the address book library, AB32.DLL, the software component used to manage the names of the users on the workgroup.

The address book also has a name-resolution feature. Here are two examples of how the name-resolution feature works:

- Suppose you are not sure about how to spell the name of the person to whom you want to send a mail message. You can type the name (taking your best guess at its spelling) into the address fields, then click the Check Names command button. Mail will compare the name to your default address list. If you typed the name correctly, it will display as a resolved name. Resolved names are underlined in the message header and are objects that you can delete, or copy and paste into other messages. If you typed the name incorrectly, an error message is displayed.
- Suppose you want to send a memo to the new guy named Dave or David Something. You can type "Dav" and choose Check Names to see a list of all names beginning with those letters. Then you can choose the correct name from that list.

Flexible Message Organization

Mail uses the traditional folder metaphor for organizing messages. Users can create their own folder organization to suit individual work styles.

Mail supports shared folders, which are folders shared by all members of a postoffice. These can be used as topical bulletin boards for workgroup members. By choosing New Window from the Windows menu, a user can view the contents of multiple private and shared folders simultaneously.

Mail's search viewers give users the ability to view a cross-section of their folder organization by searching for messages that match any of a number of criteria. Users can have multiple folder viewers open simultaneously. Each of these windows displays both the hierarchy and the list of messages in a folder. Notification of changes to the database keeps all the multiple viewers in sync.

By using the Message Finder command on the File menu, users can also create search viewer windows. Figure 9.2 shows an example of two open search-viewer

windows—one showing the search information and one resized to display only the search results. A search window icon is also shown.

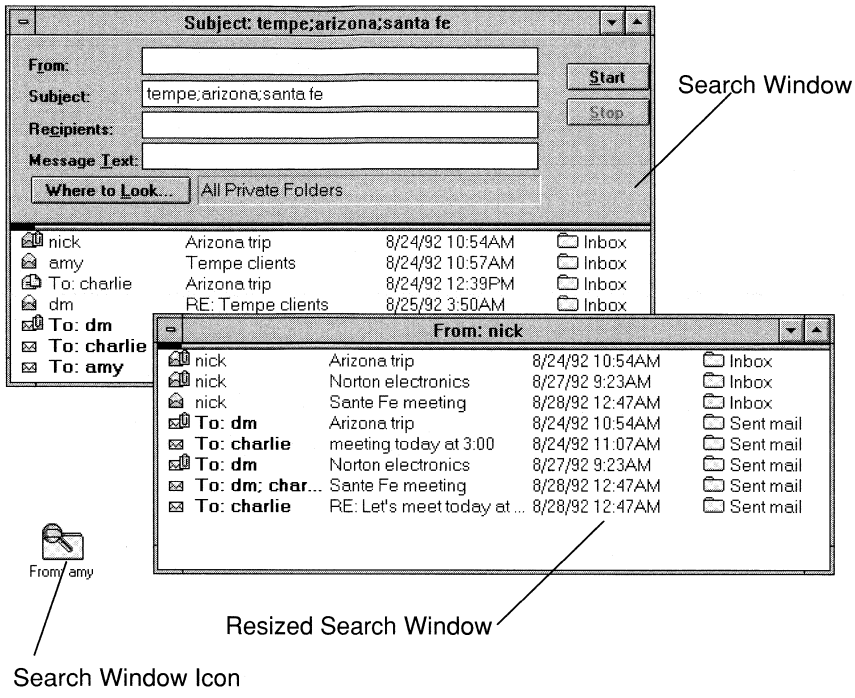


Figure 9.2 You can create multiple windows for search viewers as well as folder viewers

The search-viewer windows are dynamically updated. A search runs in the background when first invoked and can be paused or resumed at any time. Once it has made a full scan of its domain, notification from the Notification engine keeps it up to date. This dynamically-updated search information can be maintained as permanent as a folder and will remain in tact even when you stop and restart Mail.

This is a handy feature for workgroup administrators, for example, who might want to track workgroup-related help requests in a separate folder. You might even keep a separate file for workgroup printer issues or upgrade information.

A particular message may be listed in several folders, but actually is stored in only one location. What appear to be duplicates are merely pointers to the original message.

Working Online or Offline

Both Mail and Schedule+ are designed to be used at the office and at home or on the road. When your computer is connected to the mail server and a session is active between them, you can work "online." If the mail server is temporarily unavailable, you can still work offline to compose outgoing messages or respond to received messages without interruption. You can also work offline when using a laptop or home computer to compose mail.

Mail keeps the messages each user creates offline in the local outbox until the user once again is working online. At that time, the user's outbound messages are transported to the postoffice on the mail server until the recipients collect them.

You can also export your message file (with an .MMF extension) to work away from the office. To do this, you can use the Export File command from the File menu. You can then select which folders you want to copy or move to an export message file (usually EXPORT.MMF). You can even specify a range of dates for the messages you want to export. Your personal address book is also copied with the messages to the export message file.

When you take messages back to the office, you can use the Import File command from the File menu.

For more information about how to work offline, see the Mail Help topic called "Working Offline."

The Mail Postoffice

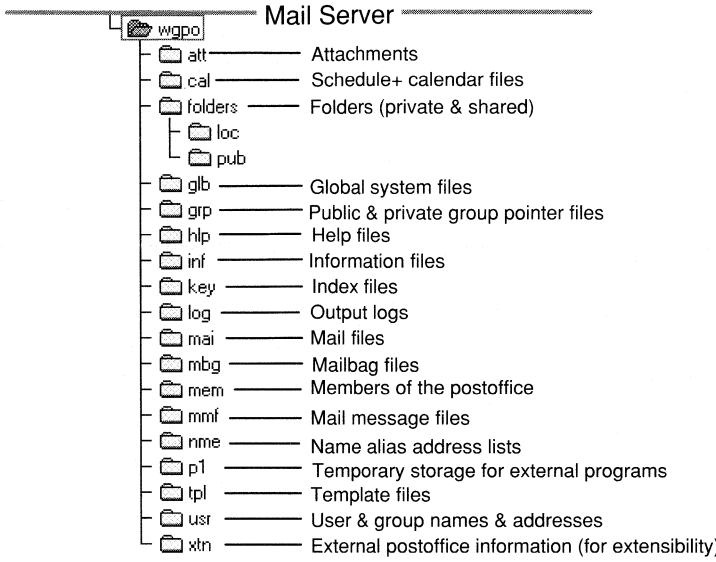


Figure 9.3 Postoffice directory structure

The postoffice, as we mentioned before, is no more than a directory structure. Figure 9.3 shows the postoffice directory structure:

All subdirectories must be present for the Mail system to function correctly. The following is a list of what's in each directory of the Mail postoffice:

- The ATT subdirectory contains encrypted file attachments.
- The FOLDERS subdirectory contains shared and private folders (with a filename extension of .FLD) for use by MS-DOS workstations. (Windows NT workstations' folders are in their .MMF files on the workstations.)
- The GLB subdirectory contains global system files for Mail. These files contain local user logon information and control files to generate mail files.
- The INF and TPL subdirectories contain file information relating to postoffice-defined templates. INF contains information files and TPL contains templates. ADMIN.INF and ADMIN.TPL contain template information for local postoffice users.
- The KEY subdirectory includes index files that contain pointers to header records in the mailbag (.MBG) files.

- The MAI subdirectory stores mail messages in encrypted form until the recipients' workstations retrieve them.
- The MBG subdirectory contains mail headers that point to the mail (.MAI) files. For each file in this directory, there is a matching index (.KEY) file.
- The MEM subdirectory contains a list of the postoffice's members.
- The NME subdirectory contains pointer files for the name alias address lists. ADMIN.NME and ADMINSHD.NME list members of the postoffice address list.
- The GRP, LOG, MMF, USR, and XTN subdirectories are useful only if you decide to upgrade to a multi-postoffice configuration. This can be done with the Microsoft Mail and Schedule+ Extensions for Windows NT. In that case, USR is used to list user names and group names for the other network and XTN is used to list other external information.

Workgroup Postoffice Administration

The workgroup postoffice administrator is responsible for creating and managing the postoffice. Administering a workgroup postoffice is easy. The only difference between an administrator and other Mail users is that you can perform the following tasks from the mail server:

- Backup the postoffice (good to do on a regular basis)
- Add users to the postoffice
- Change user information, including passwords, if someone should forget his or her current one
- Check the status of shared folders

The workgroup postoffice manager library, WGPOM32.DLL, is the software component on the postoffice that supports administrative functions such as adding or deleting users and changing passwords.

For more information about performing administrative tasks, see the *Windows NT System Guide*.

In the previous sections, we've discussed the key features of the user interface and the directory structure that make up the Mail server postoffice. In the next section, we describe how the client and server sides of Mail interact and also explore the components that make up the interface between the client and server sides.

Interface Between the Mail Client and Postoffice

Mail has a modular architecture. While some of the modules comprise the user interface and postoffice, most make up the interface between the Mail client and the Mail postoffice. Several powerful features are included in the interface between the user tools and the postoffice. Figure 9.4 shows four key components that make up this interface:

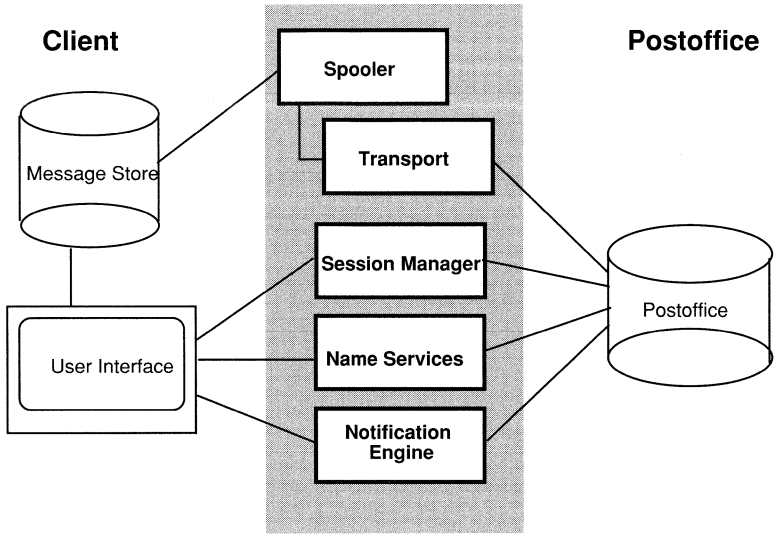


Figure 9.4 Components that make up the interface between client and server sides of Mail

This section describes each of these components.

The Mail Spooler and the Mail Transport

When you have a message to send, the mail spooler (MAILSP32.EXE) tells the mail transport (MSSFS32.DLL) to move it from your outbox to the postoffice. When a message arrives for you at the postoffice, the mail spooler tells the mail transport to deliver it to your inbox.

The mail spooler's primary job is to parcel out system idle time to the mail transport. This enables the mail transport to transfer messages in a timely way but without interfering with the work you really want to do in the foreground. The spooler also provides a safety net for the transport by retrying any operations that fail.

The spooler also resolves address book entries, adds message recipients to the personal address book, and generates non-delivery reports. It checks for new mail and deletes mail from the server.

The Mail Session Manager

Once a message is received at the postoffice, the mail session manager (MAILM32.DLL) logs it on the server. The mail session manager also validates users' identities and manages connections with the message store, directory, and transport.

The mail session manager is the component that enables you to work even when the mail server is unavailable (for example, when you work offline). Mail continues to work with the resources that are available at your computer and automatically connects to the server's resources when the server is again available. Maintaining the security of messages across transitions from offline to online and back is another feature of the mail session manager.

The Name Service

The name service (PABNSP32.DLL) manages functions related to browsing and filtering lists of names. At the user interface, the name service is apparent whenever you browse file folders, specify search criteria, or ask for a recipient's name to be checked.

With the exception of the personal address book, the name service treats the directories it uses as read-only lists. Addresses as used in Mail consist of three parts:

- Display name
- Mail address type
- Mail address

The display name is usually the full descriptive name of a person, group, or resource. This is the name you most often see in the user interface.

The mail address type helps the transport route mail and provid? the syntax of the mail address to the Mail client.

The mail address is the part actually used to identify the routing destination for a message.

The Notification Engine

When you receive new mail, the notification engine (STORE32.DLL) lets you know it has arrived.

The notification engine also works with the session manager when you browse or filter messages. Folder management and message searches use the services of the notification engine. The message finder and other folders detect notification of every change to a message. As the message changes, it is reflected in the appropriate folders and search windows.

For example, Mail enables you to have multiple windows viewing the same folder. If you delete a message from the folder in one of the viewers, it automatically disappears from the other because each viewer listens for notification of events like messages being deleted.

When a new message arrives and is written to the store by the mail spooler, the Inbox viewer is updated by the notification engine. When you place a message in your outbox, the notification engine alerts the spooler that there is a message to deliver to the postoffice.

So far we've described the standard features of Mail. In the following section, we'll show you ways you can customize Mail to suit the specific needs of your workgroup.

Customizing the Mail Client

You can use any of three Mail features to customize the way Mail works on a Windows NT workstation:

- *Custom commands* are commands you add to Mail menus. Once installed, custom commands appear to be built-in Mail features. You can install custom commands on an individual computer for use by a single Mail user or on a network file server to be shared by many Mail users.
- *Custom message types* enable you to define and install mail forms customized for your own requirements. You can install custom message types on an individual computer for use by a single Mail user or on a network file server to be shared by many Mail users.
- The *Messaging Application Program Interface* (MAPI) is a set of calls you can use to easily add mail-enabled features to other Windows-based applications.

Custom Commands

You can add custom functions to your Mail menus with custom commands. A custom command is one you add to an existing Mail menu to perform a unique function tailored for your needs. There is a second type of custom command not associated with any menu. Instead, this type of custom command can be set to run whenever the user starts Mail, ends Mail, or receives a new message.

A custom command is implemented as a dynamic-link library (DLL). Any executable code—including batch files or applications—can be run from a custom command. Here are some examples of how custom commands can be used to tailor Mail for specific needs:

- Displaying a window that contains information specific to the place the user works, such as a parts list or a list of stock quotes.
- Enabling the user to perform a query on a database without leaving Mail.
- Enabling the user to launch another application and open a specific file. For example, users can create a command to open a MEMOS.TXT file in the Windows Notepad application.

Installing Custom Commands

You can install custom commands on the local computer for use by a single user or centrally on the postoffice to be shared by many users. Installing the command on the postoffice is handy because it eases installation and administrative tasks, and saves space on other users' hard disks.

Once installed, these custom commands can appear to users as built-in features. You can install up to 100 custom commands (but not new menus) to Mail. If you install several custom commands, you can add separator bars between groups of commands to help organize the menu's appearance.

To install custom commands on an individual workstation, you need to modify the user's MSMAIL32.INI file for each custom command and add the appropriate DLLs to the user's computer. To install custom commands one time on the postoffice is easier than installing on several workstations. It requires modifying each user's MSMAIL32.INI file only once, then adding the DLLs to the postoffice and modifying the postoffice's SHARED32.INI file. The following sections explain how to install custom commands both ways.

Note Typically, the program that creates the custom commands automatically modifies lines in MSMAIL32.INI and SHARED32.INI so you don't have to. The information in the following sections is for your reference if you need to modify these files manually.

Starting Separate Applications

Mail expects that custom commands will be implemented as dynamic-link libraries (DLLs). The DLL can also pass information about the message to the application when it starts.

Although Mail always calls a DLL, the DLL can pass information to an application by using Windows dynamic data exchange (DDE), command line parameters, or by using a disk file. If the called application is already running, the DLL transfers information most efficiently using DDE. If a disk file is used, care should be taken so that multiple temporary files are not added to the user's disk. The application that implements the custom message type runs independently of Mail.

Installing Custom Commands on a Single Workstation

To install a custom command on an individual workstation, follow these steps:

1. Copy the DLL for the custom command to the WINDOWS directory of your hard drive.
2. Add a custom command entry to the **[Custom Commands]** section of the MSMAIL32.INI file on your workstation. Use this syntax:

   ```
   tag=Mail version; menu name; position; DLL name; command;
      event map; status text; Help file; Help context
   ```

   (Note that this is a single command line in MSMAIL32.INI. Don't add any carriage returns even if line-wrapping occurs.)
3. Quit Mail if it is running, and then restart it. Mail rereads your MSMAIL32.INI file and adds the custom command.

Installing a Custom Command on the Postoffice

If you have a custom command that you want to make available to multiple users, you can install the command on the postoffice instead of on each user's computer. When you create a new custom command, you must include a **SharedExtensionsDir=** entry in the **[Microsoft Mail]** section in each user's MSMAIL32.INI file. This entry instructs Mail to check the server's SHARED32.INI file for custom command entries.

Mail finds the **SharedExtensionsDir=** entry in MSMAIL32.INI, then reads entries for custom commands in SHARED32.INI before returning to MSMAIL32.INI to read any custom command entries there. For example in Figure 9.5, Mail reads the **SharedExtensionsDir=** entry in MSMAIL32.INI first. Then it reads the entries in the **[Custom Commands]** section of SHARED32.INI (in this case, the lines labeled tagA= and tagB=). After reading all of the custom command entries in SHARED32.INI, Mail reads entries in the **[Custom**

Command] section of MSMAIL32.INI (in this case, the lines labeled tag1= and tag2=).

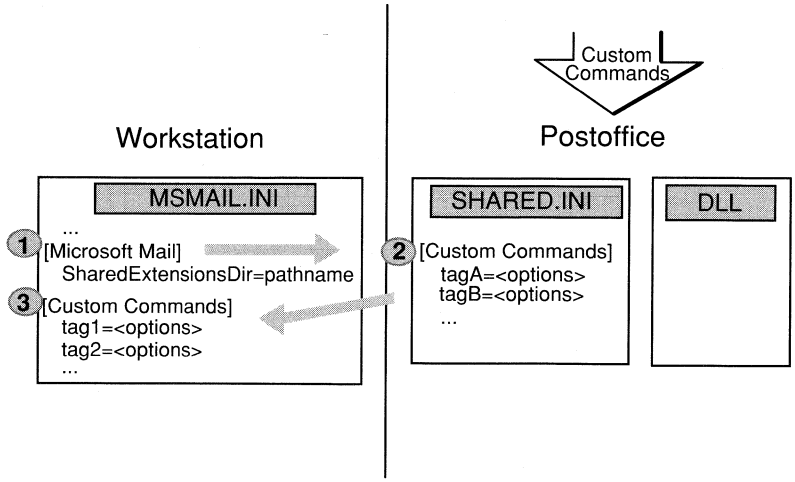


Figure 9.5 File interaction with shared custom commands

The directory on the postoffice specified by **SharedExtensionsDir=** contains the SHARED32.INI file as well as the DLLs that implement shared custom commands.

Mail can check a server for custom commands and custom messages in one of two ways:

- When you use a dynamic connection to the postoffice
- When you manually connect to the postoffice before starting Mail

The syntax you use with **SharedExtensionsDir=** reflects which way you choose to connect. The **SharedExtensionsDir=** entry uses the following syntax:

SharedExtensionsDir= [*pathname*] | [*\\server\share\pathname*] [*password*]]

When you want to dynamically connect to the server's shared disk, use the *\\server\share\pathname* option with **SharedExtensionsDir=**. Use the *pathname* option with **SharedExtensionsDir=** to point to a manual connection to the server's shared disk.

To install a shared custom command, follow these steps:

1. Create a shared Mail extensions directory on the postoffice.
2. Copy the custom command DLL to the shared extensions directory.
3. A sample SHARED32.INI file is included on the *Windows NT Resource Kit* disk. Copy this file to the shared extensions directory.

4. Add custom command entries to the **[Custom Commands]** section of the SHARED32.INI file on the postoffice, using this syntax:

   ```
   tag=Mail version; menu name; position; DLL name; command;
   event map; status text; Help file; Help context
   ```

   (Note that this is a single command line in SHARED32.INI. Don't add any carriage returns even if line-wrapping occurs.)

5. Add a **SharedExtensionsDir=** entry in MSMAIL32.INI to reference the shared extensions directory. Use this syntax:

   SharedExtensionsDir= [pathname] | [\\server\share\pathname] [password]]

   (Note that this is a single command line in MSMAIL32.INI. Don't add any carriage returns even if line-wrapping occurs.)

6. If you have chosen the manual connection method, make the necessary network connection using Windows File Manager.

7. Quit Mail if it is running, and then restart it.

   Mail rereads your MSMAIL32.INI file and adds the shared custom command to the specified menu.

8. Test the shared custom command.

9. When the custom command works successfully on the postoffice, modify each user's MSMAIL32.INI file to have access to the shared extensions directory. (See steps 5 and 6.)

Custom Message Types

A custom message type is a special type (or class) of message for delivery between two or more Mail recipients or mail-enabled applications. A custom message type can define a particular way to perform standard Mail operations such as composing, replying to, and forwarding messages. Custom message types can define a message's appearance and content, or the behavior of the forms displayed to the user when sending, reading, and replying to messages of that type.

Most custom message types are received in your inbox just like standard Mail messages. A second kind of custom message type doesn't appear in the inbox when it arrives, and yet is available to other mail-enabled applications. This allows mail-enabled applications to define their own message classes.

Custom message types have many similarities to custom commands. Like custom commands, a command for composing a message of a custom type can be added to an existing Mail menu. When the Mail user chooses this command, the custom message type DLL displays a dialog box or form that enables the user to compose a message of the corresponding type. These dialog boxes or forms can include features specific to the message type.

You can install up to 100 custom message types, but you can't add a new menu to Mail.

When a message of a custom type arrives in a Mail folder, it can appear in the message list the same as other standard Mail messages. But when read or replied to, the custom dialog boxes or forms associated with the message type can be displayed in place of the standard read or reply form. You can also design a custom message type that doesn't appear in Mail's inbox when it arrives, and yet is available to other mail-enabled applications.

Note If a custom message type is delivered to a user who does not have that custom message type installed, the message is treated as a standard message type.

Custom message types can provide special messaging functionality such as the following:

- Messages that are composed or read using a special form.
- Messages accessed by the user through an application other than Mail.
- Messages that are pre-addressed to a particular recipient.
- Messages that are used to order parts or services or to describe an event. The dialog boxes displayed when the user composes these types of messages can include special structured fields specific to the purpose of the message.
- Messages that help route work-flow events
- Messages that are specific to group scheduling
- Messages that enable some type of game, such as chess, to be played between two mail users.

Schedule+ is an example of an application that defines its own custom message types. Schedule+ uses the following custom message types:

- IPM.Microsoft Schedule.MtgReq—used to generate the Meeting Request form
- IPM.Microsoft Schedule.MtgRespP—used to generate the Positive Meeting Response form
- IPM.Microsoft Schedule.MtgRespN—used to generate the Negative Meeting Response form
- IPM.Microsoft Schedule.MtgRespA—used to generate the Tentative Meeting Response form
- IPM.Microsoft Schedule.MtgCnc—used to generate a Meeting Cancellation message

For more information about these custom message types, see Chapter 10, "Schedule+."

Installing Custom Message Types

Installing a custom message type is similar to installing a custom command. Just like custom commands, you can install custom message types on an individual workstation or on the postoffice as shared custom message types. Typically, though, you'll install them as shared custom message types because you'll want both sending and receiving parties to take advantage of the custom type.

When you install custom message types on an individual workstation, you must modify the MSMAIL32.INI file by adding a custom message type entry in the **[Custom Messages]** section. Each custom message type entry uses this syntax:

```
MessageClassName= Mail version; menu name; command name;
   command position; ExtsDir DLL name; ExtsDir command string;
   operation map; status line; ExtsDir Help file name; Help context;
```

(Note that this is a single command line in MSMAIL32.INI. Don't add any carriage returns even if line-wrapping occurs.)

For shared custom message types, you must add the custom message type entry to the postoffice's SHARED32.INI file instead of the workstation's MSMAIL32.INI file. Then, on each workstation using the shared custom message type, you must add a **SharedExtensionsDir=** entry. As with custom commands, when Mail sees this entry in MSMAIL32.INI, it reads SHARED32.INI for custom message type entries before reading custom message type entries in MSMAIL32.INI.

Installing Custom Message Types on the Postoffice

To install a shared custom message type on the postoffice, follow these steps:

1. Create a shared Mail extensions directory on the postoffice.
2. Copy the SHARED32.INI file and custom message type DLL (and/or .EXE) to the shared extensions directory.
3. Add custom message type declarations to the **[Custom Messages]** section of the SHARED32.INI file on the postoffice, using this syntax:

   ```
   MessageClassName= Mail version; menu name; command name;
      command position; ExtsDir DLL name; ExtsDir command string;
      operation map; status line; ExtsDir Help file name; Help context;
   ```

   (Note that this is a single command line in SHARED32.INI. Don't add any carriage returns even if line-wrapping occurs.)
4. Add a **SharedExtensionsDir=** entry in MSMAIL32.INI to reference the shared extensions directory. Use this syntax:

```
SharedExtensionsDir= [pathname] | [[\\server\share\pathname]
   [password]]
```

(Note that this is a single command line in SHARED32.INI. Don't add any carriage returns even if line-wrapping occurs.)

5. If you have chosen the manual connection method, make the necessary network connection using Windows File Manager.
6. Quit Mail if it is running, and then restart it.
7. Mail rereads your MSMAIL32.INI file and adds the shared custom command to the specified menu.
8. Restart Mail and test the shared custom message type.
9. When the custom message type works successfully on your computer, modify each user's MSMAIL32.INI file to have access to the shared extensions directory. (See steps 4 and 5.)

Modify each user's MSMAIL32.INI file to have access to the shared extensions directory. (See step 4.)

Messaging Application Program Interface (MAPI)

The Messaging Application Program Interface (MAPI) is a set of functions that developers can use to create mail-enabled applications. Mail includes a subset of 12 functions called Simple MAPI. Simple MAPI functions enable developers to send, address, and receive messages from within Windows-based application.

With Simple MAPI functions, developers can easily add the power of messaging to any Windows-based application. Simple MAPI supports the standard interface for simple integration of a Windows-based application with Mail.

All of the Simple MAPI functions are designed to be called from C or C++ programs, but they can also be called from high-level languages, such as Visual Basic, Actor, Smalltalk, and Object Vision. Simple MAPI functions can also be called from applications with macro languages that can call a DLL. Two examples of these are Microsoft Excel or Microsoft Word.

Some Simple MAPI functions include a user interface (a dialog box) but can also be called without generating an interface. The seamless integration with Mail is convenient for applications such as word processors and spreadsheets, which manipulate files that users may want to exchange through mail. The style of the user interface is not defined by Simple MAPI, so you can design your own.

Here's an example of how you can use Simple MAPI functions in your workgroup:

An application developer can incorporate mail functionality into his or her application by calling the MAPI functions. For example, if an application creates data files that need to be distributed to other users in a workgroup, the application developer can use the MapiSendDocuments MAPI function to create a mail messages and send the data files as an attachment to the mail message. Sending mail message is fully controlled from the application and the MAPI support library (MAPI.DLL). The user doesn't need to have the Mail program running to do this.

In this case, one function call is all that is required. The MapiSendDocuments function creates and initializes a message and supplies all the standard Mail dialog boxes for the user to send messages. The function can be compiled into the native code of the application, or if the application includes a macro facility that can dynamically link to a code library (a DLL), the developer can integrate the DLL as an added macro command.

Simple MAPI consists of these 12 functions:

Table 9.1 Simple MAPI Functions

Function	Description
MapiAddress	Addresses a Mail message
MapiDeleteMail	Deletes a message
MapiDetails	Displays a recipient details dialog box
MapiFindNext	Returns the ID of the next (or first) Mail message of a specified type
MapiFreeBuffer	Frees memory allocated by the messaging system
MapiLogoff	Ends a session with the messaging system
MapiLogon	Begins a session with the messaging system
MapiReadMail	Reads a Mail message
MapiResolveName	Displays a dialog box to resolve an ambiguous recipient name
MapiSaveMail	Saves a Mail message
MapiSendDocuments	Sends a standard Mail message
MapiSendMail	Sends a Mail message

Integrating Mail and Other Applications

Some Windows provide macros and support functions to enable them to use the capabilities of Mail directly from within the application. For example, the latest releases of both Microsoft Excel for Windows and Microsoft Word for Windows provide macros to send worksheets and documents, respectively, directly from within these applications.

As an example of how Mail can be integrated with applications, Windows NT provides a File Manager extension to add a Send Mail option to the File Manager menu and toolbar.

The Send Mail command displays all the user interface necessary for the user to send the currently selected files or executable files as file attachments to a message. The user can add message text around the file attachment, change the attached files, and address the message as usual. This reflects Microsoft's belief that messaging should be tightly integrated into the operating system.

Tips for Using Mail

This section offers tips for running the Mail application that ships with Windows NT.

Recreating the Mail Initialization Procedure

When you first run the Mail application, it asks whether to use an existing postoffice on the workgroup or to create the postoffice on your computer. If you accidentally select the incorrect option, you can use the following procedure to reinitialize Mail so that these options are available again:

1. Open MSMAIL32.INI in a text editor such as NOTEPAD.EXE, then disable, or "comment out," the **ServerPath=** and **Login=** entries by typing a semicolon (;) at the beginning of each of these lines:

   ```
   ; ServerPath=<something>
   ; Login=<something>
   ```

2. Add or edit the **CustomInitHandler=** entry to read as follows:

   ```
   CustomInitHandler=WGPOM32.DLL, 10
   ```

3. Run Mail. The initialization process will begin and will once again prompt you for the location of the workgroup postoffice.

Changing the Name of the Postoffice Administrator

If you need to assign someone new as the postoffice administrator, use the following procedure. This procedure includes two parts—one for the future administrator and one for the former administrator. As noted below, each person must wait for the other before continuing to certain steps. These procedures must be done together.

On the future administrator's workstation:

1. Copy these lines from the **[Custom Commands]** section of the former administrator's MSMAIL32.INI file:

```
WGPOMgr1=3.0;Mail;;13
WGPOMgr2=3.0;Mail;&Postoffice Manager...;14;WGPOM32.DLL;0;;
   Administer Workgroup
```

2. Make a backup copy of your .CAL file from the postoffice. (This is the calendar file used for Schedule+.)
3. Use the Export Folder command from the File menu to move all Mail messages for the future administrator to a floppy disk.
4. After the former administrator is through with steps 1 through 3 described below, log on to Mail using the former administrator's account. This is the first account that was created on the postoffice.
5. From the Mail menu, choose Postoffice Manager. Then, change the name and password for this account to reflect the information for the future administrator.
6. If the former administrator is still part of the workgroup, create a new account for that person.
7. Replace the .CAL files for your account on the postoffice with the backup file from step 2.
8. Use the Import Folder command from the File menu to move your Mail messages from export folder files created in step 3 to the .MMF files on your workstation.

On the former administrator's workstation:

1. After the future administrator has copied them, remove these lines from the **[Custom Commands]** section of your MSMAIL32.INI file:

```
WGPOMgr1=3.0;Mail;;13
WGPOMgr2=3.0;Mail;&Postoffice Manager...;14;WGPOM32.DLL;0;;Administer
   Workgroup
```

2. Make a backup copy of your .CAL file from the postoffice.

3. Use the Export Folder command from the File menu to move all Mail messages for the future administrator to a floppy disk.
4. If you are remaining as part of this workgroup, log on to your new account (created by the future administrator in step 6 above). Change the password for the account.
5. Replace the .CAL files for your account on the postoffice with the backup file from step 2.
6. Use the Import Folder command from the File menu to move your Mail messages from export folder files created in step 3 to the .MMF files on your workstation.

Packaging Objects with UNC Pointers

Windows NT supports the use of the universal naming convention (UNC) inside packaged objects. This means that you can create an object with the Object Packager utility that includes a pointer to a file located on a network file share.

For example, instead of embedding a 1MB Word for Windows document into a mail message, you can insert a packaged object that contains a UNC pointer to the document on the network share. When the message is received, the recipient can double-click on the icon created by Object Packer to connect to the share and load the Word for Windows document.

Here's how to create a packaged object containing a UNC pointer to a Word document on a network share:

1. In Mail, choose the Insert Object command from the Edit menu.
2. Choose Package from the Insert Objects list. This starts Object Packager.
3. In Object Packager, choose the Command Line command from the Edit menu.
4. Type the UNC path and filename of the Word for Windows document file in the command box. For example, you can include something like the following:

   ```
   \\COMPUTER2\WORDDOCS\BUDGET.DOC
   ```

   (The network share should not have a password.)
5. Select the Insert Icon button and choose one of the available icons for this object. Choose OK.
6. From the Edit menu, choose Label, and then type a descriptive label for the icon.
7. From the Edit menu, choose Exit. When asked if you want to update the Mail message, choose Yes.
8. Send the Mail message.

Questions and Answers about Mail

This section answers some common questions about Mail.

How much disk space does Mail's postoffice require?

You should allow approximately 2MB of storage on the mail server to start. As the mail system is used, the amount of space it requires grows based on the number of users and the size of the messages and attachments being stored.

Is there a limit on the size or number of attachments in Mail?

There is a limit to the *number* of attachments, not the size. The constraint is the size of a single mail message (not including the size of any attachments). The sum (size of the message text in bytes) + (the number of attachments) must always be less than 32K—allowing for many more attachments than most users need to get their message across.

Will the message finder search attachments to my mail messages?

Not currently.

Can I retrieve deleted messages?

Mail doesn't actually purge deleted files; it moves them to a Deleted mail folder. Deleted messages are not deleted until you empty the Deleted mail folder or quit the program. You can also configure mail not to delete messages when you quit the program.

How do I change my Postoffice location for the Mail client?

In the MSMAIL32.INI file, use the **ServerPath=** entry to redirect where Mail looks for the postoffice.

CHAPTER 10

Schedule+

Schedule+ is an application that lets you plan and schedule meetings and appointments with others in your workgroup. As a mail-enabled application, Schedule+ works together with Mail to perform key functions such as sending meeting-request messages to other workgroup members.

This chapter describes the components that make up Schedule+, and shows how the Schedule+ application works with Mail. It also describes the custom message types used for Schedule+ and includes a section that answers common questions about Schedule+.

Overview of Schedule+ Architecture

Schedule+ is an example of a mail-enabled application. It relies on Mail for certain functions, including support for logging on, accessing Mail's address book, sending messages, and receiving messages.

Because Schedule+ relies on Mail for key functionality such as sending and receiving messages, there is no special Schedule+ server. User account information for Mail is automatically translated to Schedule+ accounts.

Figure 10.1 illustrates the key components that make up Schedule+:

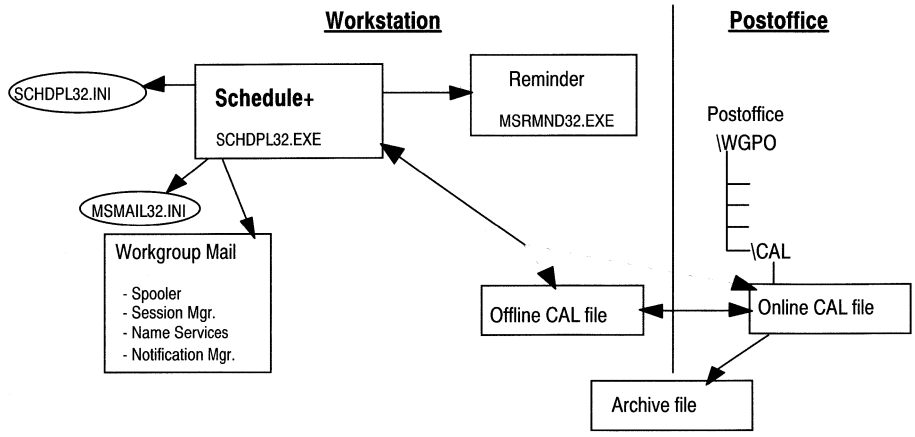


Figure 10.1 Overview of Schedule+

As shown in Figure 10.1, Schedule+ maintains both an online calendar and an offline calendar. This enables you to use Schedule+ as a stand-alone application or as a networked application. When you start Schedule+ offline, the local calendar file on your hard disk is read to the program. As you make changes to your calendar, those changes are written to disk immediately.

When you use Schedule+ online, it uses the online calendar file located in the WGPO\CAL directory of the workgroup postoffice. When run online, Schedule+ uses Mail to send meeting messages and to provide name service and logon support.

A separate program, Reminder (MSRMND32.EXE), is an application used to notify the user of Schedule+ appointments. This application is typically installed in the Windows Startup group and runs in the background. (It can alternately be launched by Schedule+.) When it is included in the Startup group, as soon as you start Windows you are prompted to supply your logon name and password. Reminder, like Schedule+, uses Mail's logon functionality. The same logon is shared by Reminder, Schedule+, and Mail. Once you log on to one of these, you are automatically logged on to the others. You can log off from all programs at once by choosing Exit and Sign Out from either Mail or Schedule+.

The next few sections discuss the components shown in Figure 10.1 that make up Schedule+.

Initialization Files

Schedule+ uses two initialization (.INI) files—SCHDPL32.INI and MSMAIL32.INI. The SCHDPL32.INI file stores the user's preferences, such as colors and the general options. It also stores the location of your calendar files and archives. MSMAIL32.INI is the initialization file for Mail. Schedule+ uses this file to identify logon information and custom message types used for scheduling.

The access privileges you set for Schedule+ are not stored in an .INI file. They are stored in your calendar (.CAL) file on the postoffice so that Schedule+ can check for the following information:

- When a person wants to look at or modify your calendar—what privileges have you assigned for that person to access your calendar?
- When someone is inviting you to a meeting—do you have an assistant, and what is the assistant's name?

Schedule+ also uses MSMAIL32.INI to identify your service provider and the path of your server. SCHMSG32.DLL refers to MSMAIL32.INI to find out how to launch Schedule+ when you choose the View Schedule button from within the Microsoft Mail client. (This button is displayed when you read received meeting requests.)

Calendar Files

The main file used by Schedule+ is a calendar file, which contains scheduling information.

Each user has an offline calendar file (named with the person's logon and a .CAL filename extension) and an online calendar file (also having a .CAL filename extension, but with a random filename). The offline file resides anywhere you want, usually on your local hard disk in the SCHDPLUS directory. The online file location is determined by the Mail transport, and resides in the CAL directory on the postoffice.

Whenever you want to access the schedules of other users on the postoffice, Schedule+ reads the data directly from the other user's online .CAL file. (Of course, you can prevent others from viewing, reading, or modifying your schedule by setting access privileges to the schedule.)

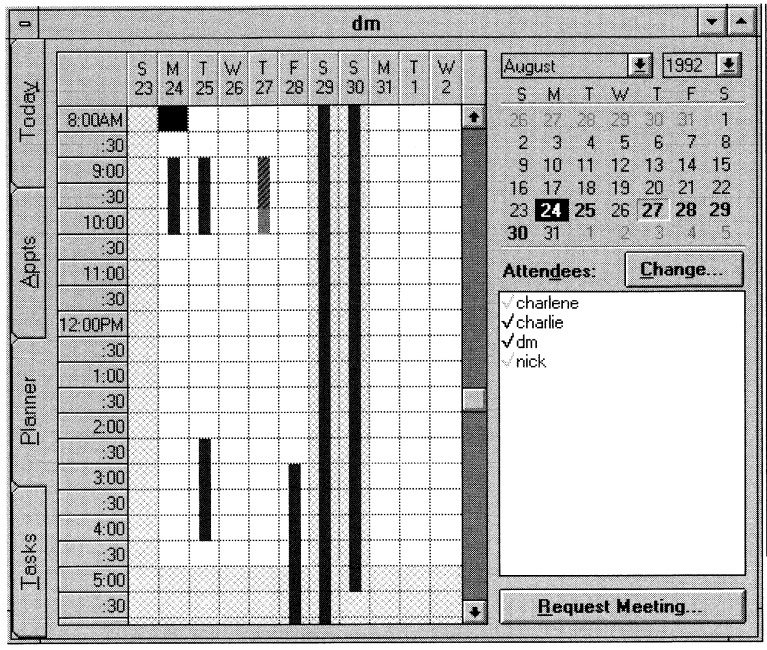


Figure 10.2 When you specify the attendees for a meeting, Schedule+ shows when everyone is available

A single postoffice file (.POF) in the CAL directory of the postoffice lists free/busy times for each user on the postoffice. When you want to schedule a meeting with another user, Schedule+ reads in the data in the .POF file to display an overlay of all users' busy times, as illustrated in Figure 10.2.

With every attendee's free time displayed, it's easy to choose a time for your meeting.

When you schedule an event with others, Schedule+ automatically creates a Request form that you can send to the others asking them to attend the meeting. The "Custom Message Types for Schedule+" section later in this chapter describes the automatic Request and Response forms.

Merging Online and Offline Calendars

As we've already mentioned, users can use Schedule+ both online and offline, and Schedule+ is designed so that users always have an online and offline calendar file. You can use your online calendar while your workstation is connected to the Mail server. If you have an assistant, that person can also make changes to your online calendar. You can also use the offline calendar when the Mail server is unavailable or when you are away from the office.

Schedule+ has a mechanism to keep online and offline calendar files synchronized. Your local (offline) file will be updated to match your server (online) file automatically each time you start and quit Schedule+ and each time you connect to or disconnect from the server. It will also be updated in the background while you are running Schedule+ if your server file changes.

The merging of calendar files happens as follows:

- Schedule+ adds all of the appointments that you have added offline to the postoffice file.
- Schedule+ deletes all of the appointments that you have deleted offline to the postoffice file.
- When you change an existing appointment, the changes made to the local file always take precedence over the changes made to the postoffice file. If you change the text of an appointment offline and another person changes the start time, both changes will be applied because these changes are not in conflict. If you change the start time, end time, start date, end date, reminder notification time, or reminder notification date for an appointment, all of these attributes will be set to the values stored in the local file.
- When an overlap occurs as a result of merging the online and offline files, Schedule+ doesn't notify you specifically; instead, both appointments are entered in the calendar. For example, suppose you schedule an appointment on Jan. 1 at noon while working offline and your Schedule+ assistant creates an appointment at the same time on the postoffice file. Because your assistant has authority to add to and modify your scheduled activities, it appears to Schedule+ that you made two changes to the schedule at the same time. Schedule+ enters both appointments in the merged calendar files.

Offsite Calendar Files

As with Microsoft Mail, you can work with Schedule+ offline at home or on a portable computer. You can take a copy of your Schedule+ files from your local computer to work with while you're away from the office.

To take your calendar home or on the road, choose the Move Local File command on the File menu to copy the calendar to a floppy disk. Then load Schedule+ on the destination computer and choose Move Local File command again. The command moves the calendar file to the location you specify and changes the SCHDPL32.INI so it searches for your calendar in the new location.

Archiving Old Calendar Information

As you might imagine, a calendar file full of scheduling information could soon get cumbersome. That is why Schedule+ includes an archiving feature.

Archiving enables users to remove past data from their calendars and store the data in a compressed format for later reference. This helps you to minimize disk space use on the server without having to completely discard past Schedule+ data.

User Accounts for People and Resources

Another key resource used by Schedule+ is the user account information maintained by Microsoft Mail. The list of user accounts for the workgroup is kept by Mail in the address book. It is also displayed in Schedule+ when you want to select attendees for a meeting.

With Schedule+, you can plan schedules not only for workgroup users but for resources such as conference rooms and AV equipment. This feature makes it easy for everyone in the workgroup to see when resources are available and to reserve the resources in advance.

To set up a resource on Schedule+, you first need to create an account for it by using Mail (just as you would for a workgroup member). Then you can log on to Schedule+ under that resource account and check "This account is for a Resource" in the General Options command. This automatically changes the default access privilege to "create appointments and tasks." That privilege means that people can reserve the resource by selecting it from the address book while they select attendees for a meeting. Users can also use the Open Other's Appt Book command to look at the calendar for the resource and book time in it directly. It is possible to assign an assistant to a resource and alter the default privilege to View Free/busy Times if you want somebody to be in control of allocating the resource.

Setting Up Resources in Schedule+

Creating a Schedule+ resource account is a two-step process for the administrator. First you must create a Mail account for each resource. Then you need to sign in to each resource account to set Schedule+ options for that account.

As the administrator, when you log on to each resource account, you are also logging into Mail. When Microsoft Mail is used with Windows NT, special steps are required to make it possible to log on to multiple accounts on the same workstation. The following information explains how.

Logging into multiple accounts from the same computer requires special steps because Microsoft Mail creates and uses an MSMAIL.MMF file that is specific to a particular Mail account. Normally, a separate .MMF file is required for each user. However, it isn't possible to create multiple MSMAIL.MMF files on the same computer. Because the MSMAIL.MMF file is protected by the Mail account password, you can work around this difficulty by simply giving each resource account the same password. As the administrator, when you set up the resource accounts from your computer, the password must be your Microsoft Mail password. This enables you to log on to Schedule+ under various accounts, while using the same .MMF file.

Note Be sure not to delete the MSMAIL.MMF file at any point.

When you sign in the first time, Mail will create the MSMAIL.MMF file. When you sign in again with the second resource account, Mail enables the existing MSMAIL.MMF file to be used because the password for the second account is the same as for the first account. You can continue signing in with each resource account and Mail will continue to use the same MSMAIL.MMF file.

When you sign in to the resource account from your computer, Mail will be able to use your MSMAIL.MMF file because the passwords for both the resource account and your account are now the same.

This process won't alter the contents of the MSMAIL.MMF file in any way, and you can continue using Mail as usual after the resource accounts are set up.

Custom Message Types for Schedule+

When you invite individuals to attend a meeting or when you reserve a resource, Schedule+ automatically prepares a Request form. When you respond to someone else's meeting request, Schedule+ generates a Response form. This section describes the forms Schedule+ automatically generates and the custom message types associated with each.

As described in Chapter 9, "Mail," Microsoft Mail enables you to create and use custom message types to send specific kinds of messages. Schedule+ uses this feature to define meeting requests, meeting responses, and meeting cancellations. In all, Schedule+ defines five message types in the **[Custom Messages]** section of your MSMAIL32.INI file:

- Meeting Request
  (defined as message type "IPM.Microsoft Schedule.MtgReq")
- Positive Meeting Response
  (message type "IPM.Microsoft Schedule.MtgRespP")
- Negative Meeting Response
  (message type "IPM.Microsoft Schedule.MtgRespN")
- Tentative Meeting Response
  (message type "IPM.Microsoft Schedule.MtgRespA")
- Meeting Cancellation
  (message type "IPM.Microsoft Schedule.MtgCncl")

These message types create special Request and Response forms that you can use to schedule meetings with others in your workgroup. For example, when you add a new appointment, choose a time, and specify attendees, the Send Request form shown in Figure 10.3 is displayed automatically:

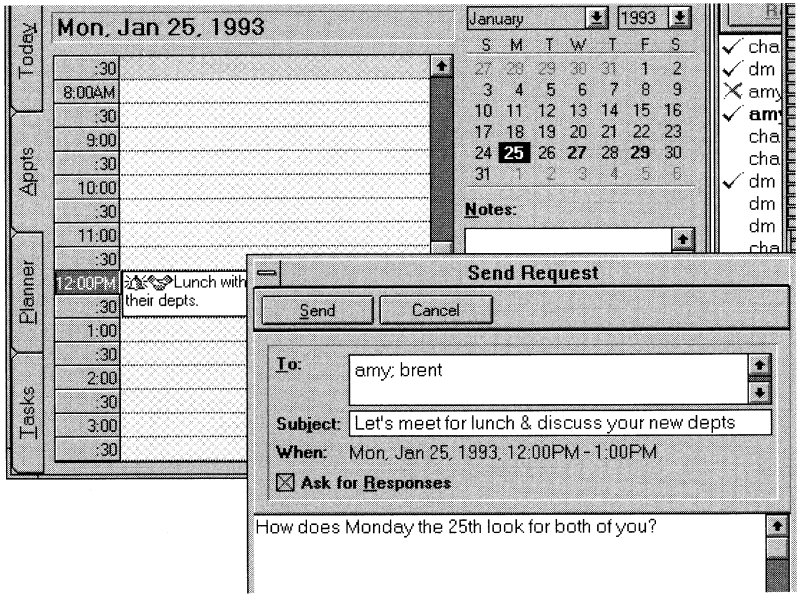


Figure 10.3 Schedule+ includes a Send Request form associated with the custom message type IPM.Microsoft Schedule.MtgReq

This message is deposited in the outbox and sent to users just as any mail message would be sent.

When a meeting attendee accepts the meeting, another automatic form, called the Response form, is generated and sent as a mail message. Three variations of the Response form (positive, negative, and tentative) are defined by custom message types defined by Schedule+. These three message types define the Response forms and a portion of the response message (such as "I might attend," shown in the Response form in Figure 10.4). They also append "Yes:", "No:", or "Tentative:" to the front of the original request title to create a response-message title. When the messages are displayed in the recipient's Messages window, these custom message types also display symbols (✓, **X**, or **?**) to the left of each message to make it easy to see responses at a glance.

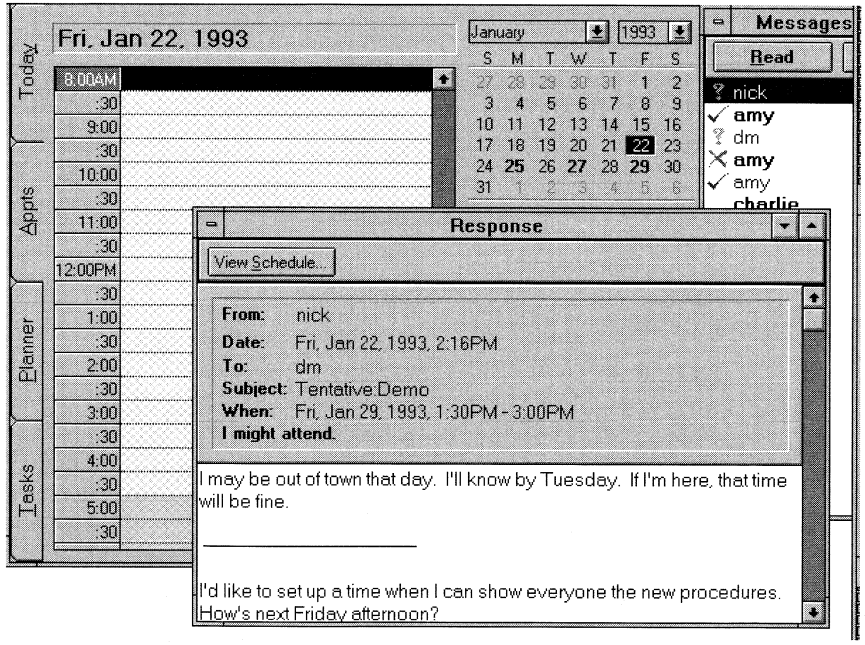


Figure 10.4 Schedule+ also defines three Response forms for negative, positive, and tentative responses to meeting requests

Because meeting messages are sent using Mail facilities, they can be viewed from within Schedule+ or from within Microsoft Mail. Only one copy of each message is maintained, however, because what appear to be duplicates are really only pointers to the original. So when you delete the message in one view, it is also deleted in the other.

Schedule+ Interoperability

Schedule+ has a flexible architecture that makes communication with other schedule and calendar systems possible. Developers can modify their schedule and calendar software to share information with Schedule+ by using the *Schedule+ Interoperability Specification*, a document which details the Schedule+ Interchange format conventions.

The Schedule+ Interchange format is a text file with a .SCH extension. An interchange file can be imported to Schedule+ by choosing File/Import Appointments, then selecting Schedule+ from the Show Files Of Type list box. Likewise, a Schedule+ calendar can be exported by choosing File/Export Appointments, then selecting Schedule+ as the file format. The file format supports importing and exporting projects, tasks, appointments, meetings, and notes.

Interchange Format Syntax

The Interchange format file begins with a header describing the owner of the file (who the schedule belongs to), and when it was exported or created. It is followed by descriptors for projects, tasks, notes, standard appointments, and recurring appointments. Although meetings are supported in the interchange format, they are not documented here. It is assumed that meetings can be imported as standard appointments by the scheduling or calendar software that is exchanging data with Schedule+. The sequence of descriptors is not significant, except that projects must precede tasks.

The following section describes the syntax for individual descriptors.

If you export a Schedule+ calendar to the interchange format, you will find that each descriptor begins with a line defining a field called "aid." This is an appointment ID defined and maintained by Schedule+. It is not required input for an import file and shouldn't be included in the import file.

Table 10.1 Key to Interchange Format Syntax

→ ? tab	m ? month or minute
italic ? required input	d ? day
non-italic = fixed field	h ? hour
definitions	y ? year

Header

SCHEDULE+ EXPORT BY *mailbox name* ON *m/dd/yy* AT *hh:mm AM/PM*

Non-Private Project

FixedAppt:
→szText
project description
→fTask → → → → →T
→aidProject → → → → → →*project id # (integer, number in sequence)*
End

Private Project

FixedAppt:
→szText
project description
→aaplWorld → → → → →Read
→fTask → → → → →T
→aidProject → → → → → →*project id # (integer, number in sequence)*
End

Note The "fTask" line is present for all projects; other lines, like the "aaplWorld" line above, are present only for private projects.

Standard Task

FixedAppt:
→szText
task description
→fTask → → → → →T
→bpri → → → → → →*priority # (1–35, where 1–9 = priority 1–9 and 10–35 = priority A–Z)*
→aidParent → → → → → →*project id # of associated project*
End

Standard Private Task

FixedAppt:
➔szText
task description
➔aaplWorld ➔ ➔ ➔ ➔ ➔Read
➔fTask ➔ ➔ ➔ ➔ ➔T
➔bpri ➔ ➔ ➔ ➔ ➔ ➔*priority # (1–35, where 1–9 = priority 1–9 and 10–35 = priority A–Z)*
➔aidParent ➔ ➔ ➔ ➔ ➔ ➔*project id # of associated project*
End

Task with Due Date and Start Work Date

FixedAppt:
➔dateStart ➔ ➔ ➔ ➔ ➔ ➔*due date in format m-d-yyyy hh:mm*
➔dateEnd ➔ ➔ ➔ ➔ ➔ ➔*due date in format m-d-yyyy hh:mm*
➔szText
task description
➔fTask ➔ ➔ ➔ ➔ ➔T
➔nAmtBeforeDeadline ➔ ➔ ➔ ➔ ➔ ➔*integer # (with unit below, indicates start work before due date)*
➔tunitBeforeDeadline ➔ ➔ ➔ ➔ ➔*unit: Day, Week, or Month (indicates start work before due date)*
➔bpri ➔ ➔ ➔ ➔ ➔ ➔*priority # (1–35, where 1–9 = priority 1–9 and 10–35 = priority A–Z)*
➔aidParent ➔ ➔ ➔ ➔ ➔ ➔*project id # of associated project*
End

Note The dateStart and dateEnd lines ask for *hh:mm*. This must be supplied but won't be used for these date-oriented (rather than time-oriented) tasks.

Standard Appointment, Marked Private

FixedAppt:
➔dateStart ➔ ➔ ➔ ➔ ➔ ➔*start date and time in format m-d-yyyy hh:mm (time is 24 hour)*
➔dateEnd ➔ ➔ ➔ ➔ ➔ ➔*end date and time in format m-d-yyyy hh:mm (time is 24 hour)*
➔szText
appointment description
➔aaplWorld ➔ ➔ ➔ ➔ ➔Read
End

Standard Appointment with Alarm

FixedAppt:
➔dateStart ➔ ➔ ➔ ➔ ➔ ➔*start date and time in format m-d-yyyy hh:mm (time is 24 hour)*
➔dateEnd ➔ ➔ ➔ ➔ ➔ ➔*end date and time in format m-d-yyyy hh:mm (time is 24 hour)*
➔szText
appointment description
➔dateNotify ➔ ➔ ➔ ➔ ➔ ➔*specification of alarm in format m-d-yyyy hh:mm (24 hour time)*
➔nAmt ➔ ➔ ➔ ➔ ➔ ➔*integer defining how long before an appointment the alarm should go off*
➔tunit ➔ ➔ ➔ ➔ ➔*the unit (Hour, Day, Week, Month) defining the alarm*
End

Note The dateNotify line asks for *hh:mm*. Place any time here; a value is required. The actual time used for the alarm is set in the nAmt and tunit lines. If the alarm is in minutes, the tunit line is not required.

Recurring Appointment

RecurAppt:
➔ymdStart ➔ ➔ ➔ ➔ ➔*start date of recurrence pattern in format m-d-yyyy*
➔ymdEnd ➔ ➔ ➔ ➔ ➔*end date of recurrence pattern in format m-d-yyyy*
➔wgrfValidMonths ➔ ➔ ➔ ➔ ➔*used to define recurrence pattern (see next section for detail)*
➔bgrfValidDows ➔ ➔ ➔ ➔ ➔➔*used to define recurrence pattern (see next section for detail)*
➔trecur ➔ ➔ ➔ ➔ ➔ ➔*used to define recurrence pattern (see next section for detail)*
➔timeStart ➔ ➔ ➔ ➔ ➔*start time in format hh:mm*
➔timeEnd ➔ ➔ ➔ ➔ ➔*end time in format hh:mm*
➔szText
Recurring appointment description
End

Note The ymdEnd line is required only if the recurring appointment has an end date. If it does not, omit the ymdEnd line. See the following section, "Recurrence Patterns," for more information about this type of entry.

Notes for One Month

MonthNotes:
➔*date of Note in format m-d-yyyy*
note #1 text
➔*date of Note in format m-d-yyyy*
note #2 text
End

Note All notes for the month are listed together in sequence.

Recurrence Patterns

The recurrence pattern is defined by three fields. In combination, these fields define the following recurrence patterns: daily, weekly, biweekly, monthly defined by a pattern, monthly defined by a date, yearly defined by a pattern, and yearly defined by a date.

The three fields are used to define the recurrence types and patterns as follows:

Table 10.2 Recurrence Pattern Fields

Field	Description
wgrfValidMonths *bbbbbbbbbbbb*	Enumerates valid months of the year, where *bbbbbbbbbbbb* is 12 bits, each being either T or F and representing January through December. All bits are T in every recurrence type except yearly. In the yearly case, all but one bit will be F. The T bit indicates which month the yearly appointment falls into in either of the yearly recurrence patterns.
bgrfValidDows *bbbbbbb*	Enumerates valid days of the week where *bbbbbbb* is seven bits, T or F, representing Sunday through Saturday. (Note that in this syntax the week always starts on Sunday regardless of the user's start of week designation in Schedule+ Options.) All bits are T in the case of a monthly or yearly recurrence defined by the specific date. In the daily-every weekday, recurrence or the monthly or yearly recurrences defined as the *x* "weekday," the first and seventh bits are F and all others are T. In the monthly or yearly recurrence where field 2 is weekend day (see description of fields below), the first and seventh bits are T and all others are F. In all other recurrence types, one or more bits are T to indicate the days of the week on which the appointment falls.

Table 10.2 Recurrence Pattern Fields *(continued)*

Field	Description
trecur [**Week** *bb* 0 | **IWeek** *bbbbb* 0 | **Date** *x*]	**trecur** is followed by one of these terms to describe the recurrence type: **Week** *bb* **0**—daily or weekly; **trecur** and **bgrfValidDows** are used in conjunction to define the daily or weekly recurrence. The days of the week are indicated as described above. **Week** *bb* **0**—biweekly; **trecur** and **bgrfValidDows** are used in conjunction to define the biweekly recurrence. **IWeek** *bbbbb* **0**—monthly or yearly, defined by a pattern instead of the date. If you look at the choices for the monthly recurrence in Schedule+, you'll see that the pattern is defined by two fields. For example, a monthly appointment can be on the first (field 1) weekday (field 2) of the month. The yearly pattern is defined by three fields. The first two are the same as the monthly fields and field 3 is the month of the year. **IWeek** is followed by five bits with T or F; only one of these bits can be T. Field 2 of the monthly or yearly recurrence pattern is defined by **bgrfValidDows**. For example, if field 2 is a weekday, then **bgrfValidDows** would be FTTTTTF. Field 3 in the yearly pattern is defined by **wgrfValidMonths**. **Date** *x*—monthly or yearly, defined by the date *x* (the day of the month, 1–31). If it is a monthly occurrence, then all months are T in **wgrfValidMonths**. If it is a yearly occurrence, then one month of the year is indicated by T in **wgrfValidMonths**, and the rest are F.

Sample Schedule+ Interchange Format File

SCHEDULE+ EXPORT BY user1 ON 9/16/1992 AT 10:44 amFixedAppt:
➔ szText
Personal Projects
➔ aaplWorld➔ ➔ ➔ ➔ ➔ Read
➔ fTask➔ ➔ ➔ ➔ ➔ T
➔ aidProject➔ ➔ ➔ ➔ ➔ ➔ 1
End
FixedAppt:
➔ szText
pay bills (belongs to Personal Project)
➔ aaplWorld➔ ➔ ➔ ➔ ➔ Read
➔ fTask➔ ➔ ➔ ➔ ➔ T
➔ bpri➔ ➔ ➔ ➔ ➔ ➔ 4
➔ aidParent➔ ➔ ➔ ➔ ➔ ➔ 1

End
FixedAppt:
➔ dateStart➔ ➔ ➔ ➔ ➔ ➔ 9-27-1992 12:00
➔ dateEnd➔ ➔ ➔ ➔ ➔ ➔ 9-27-1992 12:00
➔ szText
confirm presenters (belongs to Trade Show project)
➔ fTask➔ ➔ ➔ ➔ ➔ T
➔ nAmtBeforeDeadline➔ ➔ ➔ ➔ ➔ ➔ 2
➔ tunitBeforeDeadline➔ ➔ ➔ ➔ ➔ Day
➔ bpri➔ ➔ ➔ ➔ ➔ ➔ 1
➔ aidParent➔ ➔ ➔ ➔ ➔ ➔ 2
End
FixedAppt:
➔ dateStart➔ ➔ ➔ ➔ ➔ ➔ 10-17-1992 08:00
➔ dateEnd➔ ➔ ➔ ➔ ➔ ➔ 10-17-1992 09:00
szText
meeting with John
➔ dateNotify➔ ➔ ➔ ➔ ➔ ➔ 10-17-1992 08:00
➔ nAmt➔ ➔ ➔ ➔ ➔ ➔ 1
➔ tUnit➔ ➔ ➔ ➔ ➔ Month
End
FixedAppt:
➔ dateStart➔ ➔ ➔ ➔ ➔ ➔ 9-17-1992 14:00
➔ dateEnd➔ ➔ ➔ ➔ ➔ ➔ 9-17-1992 15:30
➔ szText
phone call with ABC Corp
➔ aaplWorld➔ ➔ ➔ ➔ ➔ Read
End
RecurAppt:
➔ ymdStart➔ ➔ ➔ ➔ ➔ 9-16-1992
➔ wgrfValidMonths➔ ➔ ➔ ➔ ➔ TTTTTTTTTTTT
➔ bgrfValidDows➔ ➔ ➔ ➔ ➔ ➔ FFFTFFF
➔ trecur➔ ➔ ➔ ➔ ➔ ➔ Week FF 0
➔ timeStart➔ ➔ ➔ ➔ ➔ 16:30
➔ timeEnd➔ ➔ ➔ ➔ ➔ 17:30
szText
Weekly appointment without end date
End
RecurAppt:
➔ ymdStart➔ ➔ ➔ ➔ ➔ 9-01-1992
➔ ymdEnd➔ ➔ ➔ ➔ ➔ 6-01-1993
➔ wgrfValidMonths➔ ➔ ➔ ➔ ➔ TTTTTTTTTTTT
➔ bgrfValidDows➔ ➔ ➔ ➔ ➔ ➔ TTTTTTT
➔ trecur➔ ➔ ➔ ➔ ➔ ➔ Date 1
➔ timeStart➔ ➔ ➔ ➔ ➔ 16:30

➔ timeEnd➔ ➔ ➔ ➔ ➔ 17:30
szText
Monthly Appointment (the first of each month) with end date
End
MonthNotes:
➔ 9-17-1992
call dentist
➔ 10-31-1992
Halloween
End

Questions and Answers about Schedule+

This section answers some common questions about Schedule+:

Does Schedule+ support OLE or attachments (such as a meeting schedule)?

No, but Mail does. You can create a message to invite attendees to a meeting and refer them to a separate mail message that includes OLE objects and/or attachments.

Does Schedule+ include all hours of each day as valid meeting times when the auto-pick feature is used to search for a meeting time?

Auto-pick only looks at weekdays (Monday–Friday) within the time defined by your start-work and end-work settings in the General Options command to determine a meeting time. Schedule+ looks only at the meeting initiator's settings to determine valid times. Schedule+ doesn't recognize the non-working hours of other users and the user isn't able to define days other than Saturday or Sunday as non-working days.

Does Schedule+ support recurring meetings? For example, I'd like to schedule a weekly meeting with my group.

Schedule+ supports recurring *appointments*, but not recurring *meetings*. Schedule+ doesn't support recurring meeting requests. Instead, you must request the meeting for a single occurrence and then remind the attendees to use the Recurring Appointment command to enter it for each week. Alternately, you can ask everyone attending your weekly meeting to set up their own recurring appointment for that time each week.

Are there options for altering the time and date displays (for example, using a 24-hour clock or a European date format)?

Yes. From the Windows NT Control Panel, select the International icon. In the dialog box, you can specify the format used for the date and the time which will affect the displays in Schedule+ also.

Does Schedule+ support customized reminder sounds?

Yes. You can modify the sound that is assigned to the Schedule+ Reminders in the Sound option in Control Panel. (You must first install the Sound Driver for PC-Speaker using the Drivers option in Control Panel.)

Because each resource requires a user account, does this mean that each resource requires a user license?

No. Microsoft's licensing is per workstation running the software, not per user account used.

What are the dimensions of the print sizes in Schedule+?

The formats are designed to match the most popular appointment book sizes. They are the following:

- Standard = 8 1/2 inches by 11 1/2 inches
- Junior = 5 1/2 inches by 8 1/2 inches
- Pocket = 3 3/4 inches by 6 3/4 inches

Are meetings scheduled by somebody else distinguished in any way?

Yes. If you double-click on an appointment created by someone else, the name of the person who created it will be indicated at the bottom of the appointment details. However, the appointment is not identified differently by an icon or special color in the appointment book.

How can I view or modify someone else's calendar?

Schedule+ enables you to grant access privileges to other users with the Set Access Privileges command from the Options menu. Once you have assigned privileges to another user, that user can view your calendar while being signed in under his/her own account by choosing Open Other's Appt. Book from the File menu.

What is the difference between the "Modify" and the "Assistant" privileges?

The modify privilege is part of the set of privileges granted to your assistant. In addition to modifying privileges, the assistant can send meeting messages on your behalf by requesting a meeting while viewing your calendar. The assistant also receives meeting messages designated for you and can respond to them on your behalf.

When I specify someone as my Schedule+ assistant, do we both receive meeting messages for my meetings or does only my assistant receive them?

You have the ability to specify either of the above. If you choose General Options from the Options menu, you will see a check box that specifies Send Meeting Messages Only To My Assistant.

Because I can view the meeting request messages from Mail or Schedule+, does this mean that there are actually two messages?

No. The Schedule+ Messages box and the Mail inbox just provide two views of the same message. The message is only stored in one place. When you delete it from one view, it is deleted in the other.

Does Schedule+ require Microsoft Mail? What happens if I want to upgrade to a multi-server network?

Schedule+ requires only the software components that make up the client portion of Microsoft Mail. Schedule+ is transport-independent in the same way that the Mail client is transport-independent. To run Schedule+ against a different back-end, you need the following:

- Microsoft Mail.
- A messaging driver that enables the client to run against another transport.
- A scheduling driver that tells Schedule+ where to find the server calendars and how to access calendars of users on other servers. (This driver is very simple—about 6K of code.)

Microsoft includes a schedule driver for each messaging driver, so you really need nothing extra to run Schedule+ against a different back-end.

CHAPTER 11

Troubleshooting

This chapter describes ways of troubleshooting various problems that can arise while installing, booting, and running Windows NT. The chapter is organized to discuss these areas:

- Hardware compatibility problems
- Installation problems
- Trouble while booting Windows NT or an alternate operating system
- Trouble while logging on to Windows NT
- Network problems
- Problems starting services or subsystems
- Problems removing Windows NT or an NTFS partition

Hardware Configuration

The following sections discuss common hardware problems that can prevent Windows NT from installing, starting, or operating properly. See also "Network Problems" later in this chapter.

Non-SCSI Hard Drives

Windows NT currently recognizes only two non-SCSI hard drives attached to a given controller. If you need more disk space, use SCSI drives.

SCSI

Make sure the SCSI bus is set up properly (refer to your hardware documentation for specific details):

- The ends of the SCSI bus must have terminating resistor packs (also called "terminators") installed. If you have only internal or only external SCSI devices, the ends of the bus are probably the SCSI adapter and the last device on the cable. If you have both internal and external SCSI devices, then the adapter probably is in the middle of the bus and should not have terminators installed.
- If you disconnect a device that has terminators installed (such as an external CD-ROM drive), be sure to install terminators on whatever device then becomes the last one on the bus.
- One of the devices on the SCSI bus, usually the adapter, should be configured to provide termination power to the bus.

Make sure the SCSI ID of the CD-ROM drive is not 0 or 1. Some SCSI adapters reserve IDs 0 and 1 for hard drives. If your CD-ROM drive has an ID of 0 or 1, you may see an extra partition that does not exist. The SCSI ID is usually set by jumpers on the device.

Adapters

Devices cannot share interrupt request (IRQ) numbers, memory buffer addresses, or ROM addresses. Make sure there are no conflicts among adapters or between the system board and adapters. Sometimes these are set with jumpers or switches on the hardware (refer to your hardware documentation for specific details), sometimes with software. If the network card can be configured through software, you can do so with Windows NT Setup. For more information on network adapter card settings, see "Network Adapter Card Settings" later in this chapter.

If the video controller is not one currently supported, have the manufacturer's disk ready for selection of resolution parameters, if other than standard VGA (640x480x16).

Network Cabling

Make sure your network adapter is configured for the network cable you are using.

Disk Space

You should have at least 66MB of space available on the hard drive for installing Windows NT.

Installing Windows NT

If you are installing Windows NT onto an MS-DOS-based system, make sure you have a bootable MS-DOS disk for your drive A that contains copies of your current CONFIG.SYS and AUTOEXEC.BAT. If major errors occur during installation, you may not be able to boot from the hard drive and will need the disk to recover your system.

Make sure your disk has enough space for the paging file (at least as much as you specified during Setup, and a minimum of 20MB).

Booting Windows NT

Windows NT boots in the following sequence:

1. When Windows NT is installed, it alters the system's boot record to look for and run a program called NTLDR.
2. NTLDR runs NTDETECT.COM, which builds a list of the system's hardware components.
3. NTLDR reads BOOT.INI and builds a menu of the operating systems that you can boot.

   BOOT.INI is a text file that has two sections; the first specifies the default operating system that will boot after a timeout value has expired and the second specifies the operating systems that you can boot. The directory specifications for Windows NT are in the ARC syntax for addressing drives. The content of a typical BOOT.INI looks like the following:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\winnt

[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\winnt="Windows NT" /NODEBUG
c:\="MS-DOS"
```

   The choices that NTLDR displays in the menu is the text that appears between quotes in the operating systems section. You can edit this to customize the menu.

   The line for Windows NT in the operating systems section should always contain \NODEBUG. The debugging mode (default or \DEBUG) is generally useful only for developing low-level device drivers. The system will run noticeably slower in debug mode.

4. Either you highlight and select an operating system to boot, or the time counts down to 0 and NTLDR boots the default operating system.

   If you don't see the menu and the default operating system automatically boots, the timeout value has been set to 0.

5. The low-level components of Windows NT load, and then Windows NT initializes the drivers and starts the services in accordance with information stored in the Registry. For more information, see Chapter 4, "Windows NT Configuration Registry."
6. The high-level components of Windows NT load, and then the Welcome screen is displayed, ready for you to log on.

Once Windows NT boots successfully, back up your configuration (<*sysroot*>\SYSTEM32\CONFIG). As you change your configuration and accounts, maintain current backups. If you have to use the repair disk that you made when you installed Windows NT to restore the system, the configuration may revert to what it was when the system was first installed. Rather than having to reconfigure the system and recreate all the accounts, you can just restore the configuration from your backup.

If Windows NT does not boot, make sure that the statements in BOOT.INI (found in the root directory of your system partition) refer to the correct path for the \WINNT directory.

For x86-based systems, do not delete BOOT.INI, NTLDR, BOOTSECT.DOS, NTDETECT.COM, or NTBOOTDD.SYS (if Windows NT is installed on a SCSI disk) in the root directory of the system partition. For RISC systems, do not delete HAL.DLL or OSLOADER.EXE in \OS\NT. If these files are deleted, the system will not boot. Use the repair disk to recover these files.

If you made changes to a system that used to boot Windows NT successfully, and now it doesn't boot, you can return to your previous configuration ("undo" the changes you made) and boot Windows NT by doing the following:

1. If your system boots both Windows NT and another operating system, press the space bar immediately after selecting Windows NT from the Boot Loader menu. If your system boots directly into Windows NT, press the space bar when the words "OS Loader" appear.

   A menu appears that lets you select one of the following choices:

   - Use Current Boot Configuration
   - Use Last Known Good Configuration
   - Reboot

2. The Last Known Good Configuration is the configuration that last successfully booted Windows NT. Select it to boot Windows NT as it was before you made the changes that prevented it from booting.

If Windows NT still won't boot, use the repair disk to recover the system. If the Repair Disk doesn't recover the system, reinstall Windows NT.

If a device driver fails to properly initialize, Windows NT may attempt to automatically boot the LastKnownGood configuration. This depends upon the Error Control value that is recorded in the Registry for that device driver.

For more information on boot configurations and Last Known Good, see Chapter 4, "Windows NT Configuration Registry."

Booting an Alternate Operating System

For an MS-DOS system, you must have an AUTOEXEC.BAT and CONFIG.SYS file in the root directory of your system partition. If these files did not exist before you ran Windows NT Setup, you can create them in Windows NT using a non-formatting text editor such as Notepad. Save the files in the root directory of drive C, then reboot your computer.

If the alternate operating system does not boot, make sure that the statements in BOOT.INI (found in the root directory of your system partition) specify the correct path for that operating system.

The file BOOTSECT.DOS contains the boot record for the alternate operating system (whether or not the alternate is MS-DOS). The boot fails if the system cannot find BOOTSECT.DOS in the root directory of the system partition. Use the repair disk to recover BOOTSECT.DOS.

At this time, UNIX cannot be an alternate operating system. Only MS-DOS and OS/2 can be used as alternates.

Logging On

There are a variety of reasons why you may not be able to log on to Windows NT. An administrator can use Event Viewer in Administrative Tools to look at the audit trail of security events to determine why you were not able to log on, and can use User Manager to resolve problems with your account or password. The Audit policies for your account must include Failure for Logon and Logoff events in order for such problems to be displayed in the security log. When viewing the security log, the administrator should filter out all events except Failure Audit.

Some of the possible problems with logging on include the following:

- Passwords are case-sensitive and you used the wrong case. For example, if your password is "Test", you cannot use a password of "test" to log on.
- You forgot your password. There is no way to determine your old password. An administrator can give you a new password;
- Your account or your password may be expired.
- You were required to enter a new password, and you may have entered one that is less than the minimum length set for the account. Also, there may be a restriction on your account concerning the reuse of an old password.
- Your account may be disabled.
- The workstation may be locked by a previous user. If that is the case, only that person can unlock the workstation.
- You may be trying to log on at a time during which you are not allowed.
- If you are logging on remotely from a workstation:
  - You may be allowed to log on only at certain computers.
  - You may be trying to log onto the wrong domain for your account.
  - The network may be down.

Network Problems

Common network problems include hardware problems with the network itself, incompatible protocols running on different devices in the network, duplicate computernames, and interrupt request (IRQ) number conflicts.

If network problems persist, use Event Viewer from the Administrative Tools group to review the error log information generated during startup. Details in the system error log will reveal possible interrupt conflicts or other driver problems.

Network Hardware Problems

There may be physical problems with the cabling and other hardware that prevents the network from operating properly. Standard tools are available for troubleshooting such problems, for example PING, PVIEW, and WNBSTAT. Refer to the documentation provided with these tools for information on how to use them.

Duplicate Computernames

Each computer on a network must have a unique name. If you specify a computername that is the same as another computer on the network or the same as a workgroup or a domain, the network will not start when you run Windows NT.

Network Adapter Card Settings

If your network adapter card can accept more than one type of cable, make sure that the card is configured for the cable that you are plugging into it. Refer to the documentation that came with the card for details.

When you configure the network adapter card, you must select the correct interrupt request (IRQ) number, I/O port base address, and memory buffer address. On older network cards, you set jumpers or switches for each of these items; with newer cards, you can program them with the driver software using only the I/O port address.

If you are not sure about which network card is installed in your computer or what its settings are, accept the defaults proposed by Windows NT Setup. After Setup is complete, you can use the Networks utility in Control Panel to install and configure network settings. For information about completing any options in the dialog boxes that appear during the network portion of Setup or when you run the Networks tool in Control Panel, use the Help button.

For the correct settings for your particular hardware, see the documentation for your network card and other devices such as SCSI adapters, or contact your hardware manufacturer.

You do not have to specify settings for built-in Ethernet capabilities on RISC-based computers from Acer, MIPS, and Olivetti®.

The default settings for network adapter cards in Windows NT are listed in the following table.

Default Network Adapter Card Settings

Card name	Default setting
3Com	
EtherLink II®, EtherLink II / TP, EtherLink II/ 16, or EtherLink II/ 16 TP	IRQ = 3 IoBaseAddress = 0x300 Transceiver = External MemoryMapped = Off
EtherLink® / MC	Not required
EtherLink 16/16 TP	IRQ = 5 IoBaseAddress = 0x200 MemoryMappedAddress = 0xD0000 MemorySize = 16 Transceiver = External Zero Wait State = Off

Default Network Adapter Card Settings *(continued)*

Card name	Default setting
DEC	
EtherWORKS LC	IRQ = 5 MemoryAddress = 0xD0000 I/O Port Address = Primary
EtherWORKS Turbo or EtherWORKS Turbo / TP	IRQ = 10 MemoryBaseAddress = 0xD0000 I/O Port Address = Primary
DEC PC	Not required
DECstation®	Slot Number = 1
IBM	
Token Ring 16/4	I/O Port Address = Primary
Token Ring 16/4A	Not required
Novell	
NE3200	Not required
NE2000	IRQ = 3 I/O Port Address = 0x300
Proteon	
P1390	IRQ = 5 I/O Port Address = 0xa20 DMA Channel = 5 Cable Type = STP Card Speed = 16
P1990	Not required
SMC®/Western Digital ™	
8003EP	IRQ = 2 MemoryBaseAddress = 0x0 I/O Port Address = 0x200
8013EWC or 8013WB	IRQ = 2 MemoryBaseAddress = 0x0 I/O Port Address = 0x200
8013EA	Not required
Ungermann-Bass ®	
Ethernet NIUps (MC) or Ethernet NIUps/EOTP (short MC)	Not required
Ethernet NIUpc (long) or Ethernet NIUpc/EOTP (short)	IRQ = 5 IoBaseAddress =0x368 MemoryMappedAddress = 0xD8000

Network Adapter Card Interrupts

The interrupt (IRQ) that you assign to a network adapter card should be unique; that is, it should not be used by any other device in the system. The standard assignments for IRQs in *x*86-based computers include the following:

IRQ	Used for	IRQ	Used for
0	Timer	8	Clock
1	Keyboard	9	—
2	(cascade)	10	—
3	COM2	11	—
4	COM1	12	—
5	LPT2	13	Math coprocessor
6	Floppy controller	14	Hard drive
7	LPT1	15	—

A network card should not be assigned an IRQ that is used by an active serial or parallel port, even if no device is currently attached to the port. Most newer *x*86-based computers let you disable the built-in serial or parallel ports. After you disable a port, you can assign its associated IRQ to another device, such as a network card.

For example, if you use only a network printer you can usually disable the built-in parallel printer ports for both LPT1 and LPT2. Network software does not use these interfaces when the underlying devices are redirected.

For information on disabling serial or parallel ports, see the documentation for your computer.

If you do not disable the serial ports, then COM1 (IRQ 4) and COM2 (IRQ 3) are usually poor choices because most *x*86-based computers come with two active serial ports. For example, a typical computer with a mouse on COM1 and a modem on COM2 cannot use IRQ 3 or 4 for a network adapter card. IRQ 5 is often a safe choice, because *x*86-based computers usually do not have two parallel printer ports.

If you have two or more COM ports on your computer, you might find that a network adapter card (especially an EtherLink II card) will conflict with one port. Two common symptoms are that the workstation fails to start and that an error attributed to the network adapter card is logged in Event Viewer.

▶ **To change the interrupt of a network adapter card**

1. Choose the Networks option in Control Panel.
2. Double-click the correct entry in the list of Adapter Cards.
3. In the configuration dialog box, change the interrupt number from 3 to an available interrupt, such as 5 or 10.

   Make sure that the interrupt you choose is not being used by another device.

Assigning I/O Port Base Addresses

Most devices have unique default I/O port base addresses. In the rare case that an I/O port appears to be in conflict, it can usually be moved to another setting without harm. The following table shows some common I/O port addresses:

I/O address	Used for	I/O address	Used for
3F8	COM1	300	—
3BC	—	2F8	COM2
378	LPT1	278	LPT2

Refer to the documentation for your network adapter card and other hardware devices to find what I/O addresses are required or settable for your system.

Assigning Memory Buffer Addresses

No two devices can share memory buffers. Make sure that the network adapter card buffer address is not already used by another device, such as a SCSI adapter card or hard disk controller. Check the installation guide for your computer or peripherals to verify the setting of the memory buffer address.

Some SCSI and network adapters use conflicting memory addresses, such as an Adaptec™ or Future Domain SCSI adapter and a DEC EtherWORKS Turbo TP network adapter. This requires reconfiguring the hardware by changing jumpers.

You can use the Microsoft Diagnostics program (MSD.EXE) installed with either Windows NT or MS-DOS to check how memory buffers are being used.

Use the following list to track which buffers are used by various devices in your system.

1MB	FC00	_____
	F800	_____
	F400	_____
	F000	_____
960K	EC00	_____
	E800	_____
	E400	_____
	E000	_____
896K	DC00	_____
	D800	_____
	D400	_____
	D000	_____
	CC00	_____
	C800	_____
	C000	_____
768K	BC00	_____
	B800	_____
	B400	_____
	B000	_____
704K	AC00	_____
	A800	_____
	A400	_____
	A000	_____
640K		

Services or Subsystems Not Starting

If services or subsystems do not start properly, use the Services and Devices applets in Control Panel to check their status. You can try to start them with those applets. Check the system log in Event Viewer for entries relating to the problem.

Removing Windows NT

There are two ways to remove Windows NT from your computer:

1. To return to your original MS-DOS-based configuration, boot MS-DOS and type the following:

   ```
   sys c:
   ```

   This will replace the Windows NT boot sector (which runs NTLDR) with the MS-DOS boot sector (which boots your computer straight into MS-DOS).

2. Delete the following files, which are left in the root directory:
   - BOOT.INI
   - NTBOOTDD.SYS (if this is a SCSI disk)
   - NTDETECT.COM
   - NTLDR
   - PAGEFILE.SYS

Note If you want to leave NTLDR and the other files on the disk, you can boot MS-DOS without prompting by editing BOOT.INI.

BOOT.INI is a read-only file. To edit it, change the attributes of the file by typing the following:

```
attrib -r BOOT.INI
```

Alternately, you can use the System applet in Control Panel to change BOOT.INI. Be sure to remove the read-only attribute from BOOT.INI first.

Change the timeout parameter to 0 and the default operating system to C:\. After you are done, the contents of BOOT.INI should be similar to the following:

```
[boot loader]
timeout=0
default=c:\

[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\winnt="Windows NT" /NODEBUG
c:\="MS-DOS"
```

Removing an NTFS Partition

If Windows NT is not installed on the NTFS partition, you can use the FORMAT command from a Windows NT command prompt.

Windows NT will not let you format the drive that it is installed on. If Windows NT is installed on an NTFS partition and you want to reformat the drive, do the following:

1. Run the Windows NT setup program.
2. When the screen appears that asks whether you want to install Windows NT on a particular partition, or want to create or delete a partition, highlight the NTFS partition and type "P" to delete it.
3. Either continue installing Windows NT or press F3 to exit the setup program.

APPENDIXES

Troubleshooting Flowcharts

The following pages include tables and flowcharts to help you with the following procedures:

- Identifying system requirements before you install Windows NT
- Checking interrupts for your computer
- Changing a password for a user
- Converting a FAT or HPFS partition to NTFS
- Repairing a corrupted FAT, HPFS, or NTFS partition

System Requirements for Windows NT

The following are system requirements for bothe RISC-based and Intel *x*86 based system

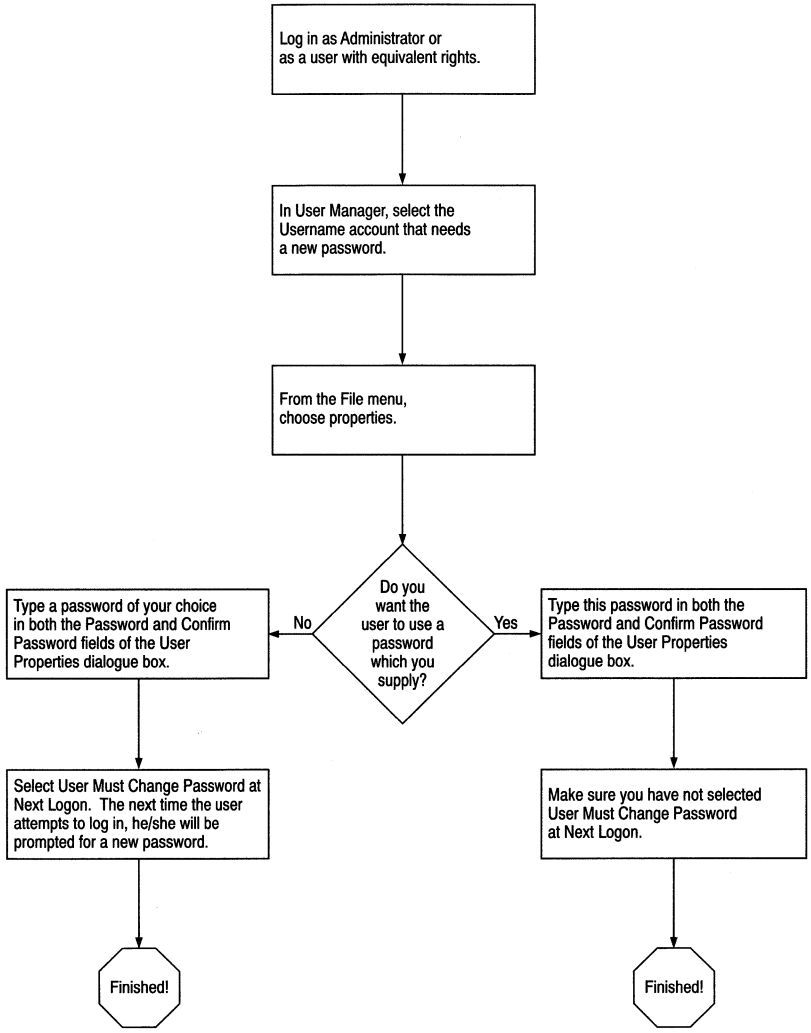
Intel x86-based	RISC-based
Required Components:	
32-bit *x*86-based microprocessor (386dx/25 or higher)	Support RISC-based Microprocessor
VGA, or higher resolution, video display adapter	VGA, or higher resolution, video display adapter
High density floppy drive	SCSI CD-ROM drive
One or more hard disks, with 50MB minimum free hard disk space on the partition that will contain the Windows NT systems files	One or more hard disks, with 50MB minimum free hard disk space on the partition that will contain the Windows NT systems files
8MB RAM	16MB RAM
Optional Components:	
Mouse or other pointing device	Mouse or other pointing device
One or more SCSI CD-ROM drives	
One or more network adapter cards, if you want to use Windows NT with a network	One or more network adapter cards, if you want to use Windows NT with a Network

Common Hardware Interrupt Usage for Intel 80x86 Computers

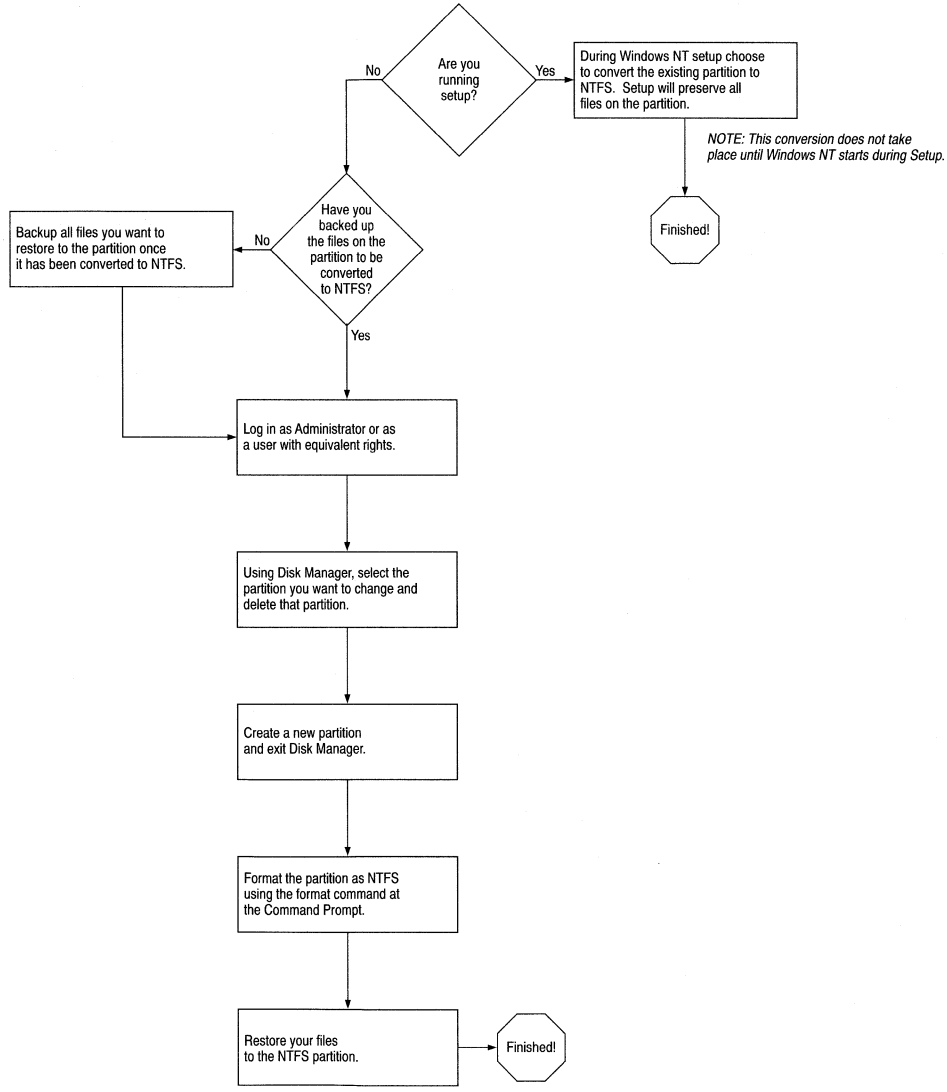
IRQ Usage for Intel 80x86 Computers

IRQ	Use
0	System Timer
1	Keyboard
2	Progrmmable Interrupt Controller Cascade(to IRQ 9)
3	COM2 and COM4 (Serial Ports 2 and 4)
4	COM1 and COM3 (Serial Ports 1 and 3)
5	LPT 2 (Parallel Printer Port 2)
6	Floppy Disk Controller
7	LPT 1 (Parellel Printer Port 1)
8	Real Time Clock
9	Cascade to IRQ 2
10	Reserved for Future Expansion (Open)
11	Reserved for Future Expansion (Open)
12	Reserved for Future Expansion (Open) Mouse Port on PS/2
13	80*x*87 Math Coprocessor
14	Hard Disk Controller
15	Reserved for Future Expansion (Open)

How the Administrator Can Recover from a User Forgetting His/Her Login Password



Converting from the FAT or HPFS File System to the NTFS File System



Repairing Corruption on an NTFS, HPFS, or FAT Volume