How NTFS Works

120 out of 142 rated this helpful

Updated: March 28, 2003

Applies To: Windows Server 2003, Windows Server 2003 R2, Windows Server 2003 with SP1, Windows Server 2003 with SP2

How NTFS Works

In this section

A file system is a required part of the operating system that determines how files are named, stored, and organized on a volume. A file system manages files and folders, and the information needed to locate and access these items by local and remote users.

Microsoft Windows Server 2003 supports the NTFS file system on basic and dynamic disks. Basic disks and volumes are the storage types most often used with Windows operating systems. Dynamic disks offer greater flexibility for volume management because they use a database to track information about dynamic volumes on the disk and about other dynamic disks in the computer.

During the format of a volume you can choose the type of file system for the volume. When you choose the NTFS file system, the formatting process places the key NTFS file data structures on the volume, regardless of whether it is a basic or dynamic volume.

NTFS Architecture

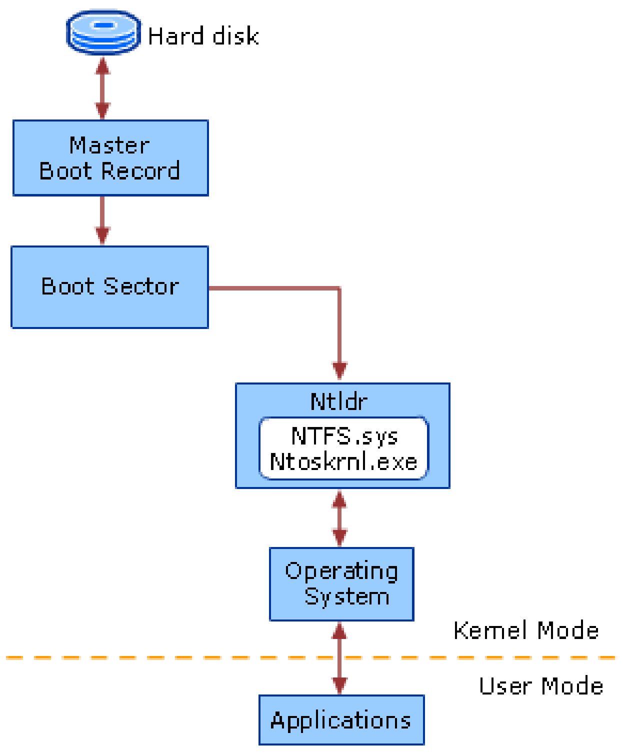
During format and setup of a volume file system on a hard disk, a master boot record (MBR) is created. The MBR contains a small amount of executable code called the master boot code as well as a partition table for the disk. When a volume is mounted, the MBR executes the master boot code and transfers control to the boot sector on the disk, allowing the server to boot the operating system on the file system of that specific volume.

Note

- The partition table contains a number of fields used to describe the partition. One of these fields is the System ID field, which defines the file system, such as NTFS, on the partition. For NTFS volumes, the system ID is 0x07.

The figure NTFS Architecture shows the architecture of this process.

NTFS Architecture



The following table describes the components of an NTFS file system.

NTFS Architecture Components on an x86-based System

Component	Component Description
Hard disk	Contains one or more partitions.
Boot sector	Bootable partition that stores information about the layout of the volume and the file system structures, as well as the boot code that loads Ntdlr.
Master Boot Record	Contains executable code that the system BIOS loads into memory. The code scans the MBR to find the partition table to

	determine which partition is the active, or bootable, partition.
Ntldlr.dll	Switches the CPU to protected mode, starts the file system, and then reads the contents of the Boot.ini file. This information determines the startup options and initial boot menu selections.
Ntfs.sys	System file driver for NTFS.
Ntoskrnl.exe	Extracts information about which system device drivers to load and the load order.
Kernel mode	The processing mode that allows code to have direct access to all hardware and memory in the system.
User mode	The processing mode in which applications run.

NTFS Physical Structure

The following information describes how clusters and sectors are organized on an NTFS volume, how the boot sector on the volume determines the file system, and how the Master File Table (MFT) organizes structures on the volume.

Clusters and Sectors on an NTFS Volume

A cluster (or allocation unit) is the smallest amount of disk space that can be allocated to hold a file. All file systems used by Windows Server 2003 organize hard disks based on cluster size, which is determined by the number of sectors (units of storage on a hard disk) that the cluster contains. For example, on a disk that uses 512-byte sectors, a 512-byte cluster contains one sector, whereas a 4-kilobyte (KB) cluster contains eight sectors.

Computers access certain sectors on a hard disk during startup to determine which operating system to start and where the partitions are located. The data stored on these sectors varies depending on the computer platform.

Sequence of Clusters on an NTFS Volume

Clusters on an NTFS volume are numbered sequentially from the beginning of the partition into logical cluster numbers. NTFS stores all objects in the file system using a record called the Master File Table (MFT), similar in structure to a database.

On NTFS volumes, clusters start at sector zero; therefore, every cluster is aligned on the cluster boundary. Contiguous clusters for file storage allow for faster processing of a file.

Note

- Floppy disks do not use NTFS and are always formatted as FAT.

Limitations of Cluster Sizes on an NTFS Volume

Because NTFS uses different cluster sizes depending on the size of the volume, each file system has a maximum number of clusters it can support. The smaller the cluster size, the more efficiently a disk potentially stores information because unused space within a cluster cannot be used by other files. And the more clusters a file system supports, the larger the volumes you can create and format by using a particular file system. NTFS uses smaller cluster sizes, which makes it a more efficient file organization structure.

The table Default NTFS Cluster Sizes lists NTFS volume and default cluster sizes.

Default NTFS Cluster Sizes

Volume Size	NTFS Cluster Size
7 megabytes (MB)–512 MB	512 bytes
513 MB–1,024 MB	1 KB
1,025 MB–2 GB	2 KB
2 GB–2 terabytes	4 KB

Maximum Sizes on an NTFS Volume

Before you format an NTFS volume, evaluate the types of files to be stored on the volume so that you can determine whether to use the default cluster size.

When formatting NTFS volumes, you can specify a cluster size of up to 64 KB using the Disk Management snap-in. If you format a volume, but do not specify a cluster size, default values are used. If you want to change the cluster size after the volume is formatted, you must reformat the volume.

Before you choose a cluster size other than the default, note the following important limitations:

- For Microsoft Windows NT, Windows 2000, Windows XP, and Windows Server 2003, the cluster size of FAT16 volumes ranging from 2 gigabytes (GB) through 4 GB is 64 KB, which can create compatibility issues with some applications. For example, setup programs do not compute free space properly on a volume with 64-KB clusters and cannot run because of a perceived lack of free space. For this reason, you can use either NTFS or FAT32 to format volumes larger than 2 GB.

- Because file compression is not supported on cluster sizes greater than 4 KB, the default NTFS cluster size for Windows Server 2003 never exceeds 4 KB.

In theory, the maximum NTFS volume size is 264 clusters minus 1 cluster. However, the maximum NTFS volume size as implemented in Windows Server 2003 is 2^{32} clusters minus 1 cluster. For example, using 64-KB clusters, the maximum NTFS volume size is 256 terabytes minus 64 KB. Using the default cluster size of 4 KB, the maximum NTFS volume size is 16 terabytes minus 4 KB.

Note

- If you use large numbers of files in an NTFS folder (300,000 or more), disable short-file name generation for better performance, and especially if the first six characters of the long file names are similar.

The table NTFS Size Limits lists NTFS size limits.

NTFS Size Limits

Description	Limit
Maximum file size	Architecturally: 16 exabytes minus 1 KB (2^{64} bytes minus 1 KB) Implementation: 16 terabytes minus 64 KB (2^{44} bytes minus 64 KB)
Maximum volume size	Architecturally: 2^{64} clusters minus 1 cluster Implementation: 256 terabytes minus 64 KB (2^{32} clusters minus 1 cluster)
Files per volume	4,294,967,295 (2^{32} minus 1 file)

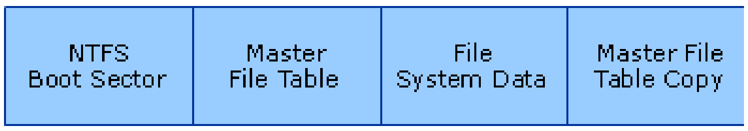
Partition Tables on MBR and GUID disks

Master boot record (MBR) disks use both basic and dynamic volumes. Because partition tables on MBR disks support partition sizes only up to 2 terabytes, you must use dynamic volumes to create NTFS volumes over 2 terabytes. Windows Server 2003 manages dynamic volumes in a special database instead of in the partition table; therefore dynamic volumes are not subject to the 2-terabyte physical limit imposed by the partition table. Dynamic NTFS volumes can be as large as the maximum volume size supported by NTFS. Itanium-based computers that use GUID partition table (GPT) disks also support NTFS volumes larger than 2 terabytes.

Organization of an NTFS Volume

The figure Organization of an NTFS Volume illustrates how NTFS organizes structures on a volume.

Organization of an NTFS Volume



The following table describes each of the organizational structures on the NTFS volume.

NTFS Volume Components

Component	Description
NTFS Boot Sector	Contains the BIOS parameter block that stores information about the layout of the volume and the file system structures, as well as the boot code that loads Windows Server 2003.
Master File Table	Contains the information necessary to retrieve files from the NTFS partition, such as the attributes of a file.
File System Data	Stores data that is not contained within the Master File Table.
Master File Table Copy	Includes copies of the records essential for the recovery of the file system if there is a problem with the original copy.

Boot Sectors

On MBR disks, the boot sector, which is located at the first logical sector of each partition, is a critical disk structure for starting your computer. It contains executable code and the data required by the code, including information that the file system uses to access the volume. The boot sector is created when you format a volume. At the end of the boot sector is a 2-byte structure called a signature word or end of sector marker, which is always set to 0x55AA. On computers running Windows Server 2003, the boot sector on the active partition loads into memory and starts Ntldr, which loads the boot menu if multiple versions of Windows are installed, or loads the operating system if only one operating system is installed.

GUID partition table (GPT) disks are similar to MBR disks, except they use primary and backup partition structures to provide redundancy. These structures are located at the beginning and the end of the disk. GPT identifies these structures by their logical block address (LBA) rather than by their relative sectors.

A boot sector consists of the following elements:

- An x86-based CPU jump instruction.
- The original equipment manufacturer identification (OEM ID).
- The BIOS parameter block (BPB), a data structure.
- The extended BPB.
- The executable boot code (or bootstrap code) that starts the operating system.

All Windows Server 2003 boot sectors contain the preceding elements regardless of the type of disk (basic disk or dynamic disk).

Components of a Boot Sector

The MBR transfers CPU execution to the boot sector, so the first three bytes of the boot sector must be valid, executable x86-based CPU instructions. This includes a jump instruction that skips the next several nonexecutable bytes.

Following the jump instruction is the 8-byte OEM ID, a string of characters that identifies the name and version number of the operating system that formatted the volume. To preserve compatibility with MS-DOS, Windows Server 2003 records “NTFS” in this field.

Note

- You might also see the OEM ID “MSWIN4.0” on disks formatted by Windows 95 and “MSWIN4.1” on disks formatted by Windows 95 OEM Service Release 2 (OSR2), Windows 98, and Windows Millennium

Edition. Windows Server 2003 does not use the OEM ID field in the boot sector except for verifying NTFS volumes.

Following the OEM ID is the BPB, which provides information that enables the executable boot code to locate Ntldr. The BPB always starts at the same offset, so standard parameters are in a known location. Disk size and geometry variables are encapsulated in the BPB. Because the first part of the boot sector is an x86 jump instruction, the BPB can be extended in the future by appending new information at the end. The jump instruction needs only a minor adjustment to accommodate this change. The BPB is stored in a packed (unaligned) format.

NTFS Boot Sector

The table Boot Sector Sections on an NTFS Volume describes the boot sector of a volume that is formatted with NTFS. When you format an NTFS volume, the format program allocates the first 16 sectors for the boot sector and the bootstrap code.

Boot Sector Sections on an NTFS Volume

Byte Offset	Field Length	Field Name
0x00	3 bytes	Jump instruction
0x03	8 bytes	OEM ID
0x0B	25 bytes	BPB
0x24	48 bytes	Extended BPB
0x54	426 bytes	Bootstrap code
0x01FE	2 bytes	End of sector marker

On NTFS volumes, the data fields that follow the BPB form an extended BPB. The data in these fields enables Ntldr to find the MFT during startup. On NTFS volumes, the MFT is not located in a predefined sector. For this reason, NTFS can move the MFT if there is a bad sector in the current location of the MFT. However, if the data is corrupted, the MFT cannot be located, and Windows Server 2003 assumes that the volume has not been formatted.

The following example illustrates the boot sector of an NTFS volume that is formatted by using Windows Server 2003. The printout is formatted in three sections:

- Bytes 0x00– 0x0A are the jump instruction and the OEM ID (shown in bold print).
- Bytes 0x0B–0x53 are the BPB and the extended BPB.
- The remaining code is the bootstrap code and the end of sector marker (shown in bold print).

```
Physical Sector: Cyl 0, Side 1, Sector 1
00000000: EB 52 90 4E 54 46 53 20 - 20 20 2
00000010: 00 00 00 00 00 F8 00 00 - 3F 00 F
00000020: 00 00 00 00 80 00 80 00 - 1C 91 1
00000030: 00 00 04 00 00 00 00 00 - 11 19 1
00000040: F6 00 00 00 01 00 00 00 - 3A B2 7
00000050: 00 00 00 00 FA 33 C0 8E - D0 BC 0
```

The table BPB and Extended BPB Fields on NTFS Volumes describes the fields in the BPB and the extended BPB on NTFS volumes. The fields starting at 0x0B, 0x0D, 0x15, 0x18, 0x1A, and 0x1C match those on FAT16 and FAT32 volumes. The sample values correspond to the data in this example.

BPB and Extended BPB Fields on NTFS Volumes

Byte Offset	Field Length	Sample Value	Field Name and Definition
0x0B	2 bytes	00 02	**Bytes Per Sector**. The size of a hardware sector. For most disks used in the United States, the value of this field is 512.
0x0D	1 byte	08	**Sectors Per Cluster**.The number of sectors in a cluster.
0x0E	2 bytes	00 00	**Reserved Sectors**. Always 0 because NTFS places the boot sector at the beginning of the partition. If the value is not 0, NTFS fails to mount the volume.
0x10	3 bytes	00 00 00	Value must be 0 or NTFS fails to mount the volume.
0x13	2 bytes	00 00	Value must be 0 or NTFS

			fails to mount the volume.
0x15	1 byte	F8	**Media Descriptor**. Provides information about the media being used. A value of F8 indicates a hard disk and F0 indicates a high-density 3.5-inch floppy disk. Media descriptor entries are a legacy of MS-DOS FAT16 disks and are not used in Windows Server 2003.
0x16	2 bytes	00 00	Value must be 0 or NTFS fails to mount the volume.
0x18	2 bytes	3F 00	Not used or checked by NTFS.
0x1A	2 bytes	FF 00	Not used or checked by NTFS.
0x1C	4 bytes	3F 00 00 00	Not used or checked by NTFS.
0x20	4 bytes	00 00 00 00	The value must be 0 or NTFS fails to mount the volume.
0x24	4 bytes	80 00 80 00	Not used or checked by NTFS.
0x28	8 bytes	1C 91 11 01 00 00 00 00	**Total Sectors**. The total number of sectors on the hard disk.
0x30	8 bytes	00 00 04 00 00 00 00 00	**Logical Cluster Number for the File $MFT**. Identifies the location of the MFT by using its logical cluster number.
0x38	8 bytes	11 19 11 00 00 00 00 00	**Logical Cluster Number for the File $MFTMirr**. Identifies the location of the mirrored copy of the MFT by using its logical cluster number.
0x40	1 byte	F6	**Clusters Per MFT Record**

0x40	1 byte	F6	**Clusters Per MFT Record**. The size of each record. NTFS creates a file record for each file and a folder record for each folder that is created on an NTFS volume. Files and folders smaller than this size are contained within the MFT. If this number is positive (up to 7F), then it represents clusters per MFT record. If the number is negative (80 to FF), then the size of the file record is 2 raised to the absolute value of this number.
0x41	3 bytes	00 00 00	Not used by NTFS.
0x44	1 byte	01	**Clusters Per Index Buffer**. The size of each index buffer, which is used to allocate space for directories. If this number is positive (up to 7F), then it represents clusters per MFT record. If the number is negative (80 to FF), then the size of the file record is 2 raised to the absolute value of this number.
0x45	3 bytes	00 00 00	Not used by NTFS.
0x48	8 bytes	3A B2 7B 82 CD 7B 82 14	**Volume Serial Number**. The volume's serial number.
0x50	4 bytes	00 00 00 00	Not used by NTFS.

Master File Table

When you format a volume with NTFS, Windows Server 2003 creates an MFT and metadata files on the partition. The MFT is a relational database that consists of rows of file records and columns of file attributes. It contains at least one entry for every file on an NTFS volume, including the MFT itself.

The MFT stores the information required to retrieve files from the NTFS partition.

MFT and Metadata Files

Because the MFT stores information about itself, NTFS reserves the first 16 records of the MFT for metadata files (approximately 16 KB), which are used to describe the MFT. Metadata files that begin with a dollar sign ($) are described in the table Metadata Files Stored in the MFT. The remaining records of the MFT contain the file and folder records for each file and folder on the volume.

Metadata Files Stored in the MFT

System File	File Name	MFT Record	Purpose of the File
Master file table	$Mft	0	Contains one base file record for each file and folder on an NTFS volume. If the allocation information for a file or folder is too large to fit within a single record, other file records are allocated as well.
Master file table mirror	$MftMirr	1	Guarantees access to the MFT in case of a single-sector failure. It is a duplicate image of the first four records of the MFT.
Log file	$LogFile	2	Contains information used by NTFS for faster recoverability. The log file is used by Windows Server 2003 to restore metadata consistency to NTFS after a system failure. The size of the log file depends on the size of the volume, but you can increase the size of the log file by using the Chkdsk command.

Volume	$Volume	3	Contains information about the volume, such as the volume label and the volume version.
Attribute definitions	$AttrDef	4	Lists attribute names, numbers, and descriptions.
Root file name index	.	5	The root folder.
Cluster bitmap	$Bitmap	6	Represents the volume by showing free and unused clusters.
Boot sector	$Boot	7	Includes the BPB used to mount the volume and additional bootstrap loader code used if the volume is bootable.
Bad cluster file	$BadClus	8	Contains bad clusters for a volume.
Security file	$Secure	9	Contains unique security descriptors for all files within a volume.
Upcase table	$Upcase	10	Converts lowercase characters to matching Unicode uppercase characters.
NTFS extension file	$Extend	11	Used for various optional extensions such as quotas, reparse point data, and object identifiers.
		12–15	Reserved for future use.

The data segment locations for both the MFT and the backup MFT, $Mft and $MftMirr, respectively, are recorded in the boot sector. The $MftMirr is a duplicate image of either the first four records of the $Mft or the first cluster of the $Mft, whichever is larger. If any MFT records in the mirrored range are corrupted or unreadable, NTFS reads the boot sector to find the location of the $MftMirr. NTFS then reads the $MftMirr and uses the information in $MftMirr instead of the information in the MFT. If possible, the correct data from the $MftMirr is written back to the corresponding location in the $Mft.

MFT Zone

To prevent the MFT from becoming fragmented, NTFS reserves 12.5 percent of volume by default for exclusive use of the MFT. This space, known as the MFT zone, is not used to store data unless the remainder of the volume becomes full.

Depending on the average file size and other variables, as the volume fills to capacity, either the MFT zone or the unreserved space on the volume becomes full first.

- Volumes that have a small number of large files exhaust the unreserved space first.
- Volumes with a large number of small files exhaust the MFT zone space first.

In either case, fragmentation of the MFT occurs when one region or the other becomes full. You can change the size of the MFT zone for newly created volumes by to correspond to a percentage of the volume to be used as the MFT zone. The MFT zone sizes follow:

- Setting 1, the default, reserves approximately 12.5 percent of the volume.
- Setting 2 reserves approximately 25 percent.
- Setting 3 reserves approximately 37.5 percent.
- Setting 4 reserves approximately 50 percent.

In most computers, the default setting of 1 is adequate. The default setting accommodates volumes with an average file size of 8 KB. Storing a large number of smaller files might necessitate that you increase the size of the MFT zone for new volumes.

After you increase the size of the MFT zone, NTFS does not immediately allocate space to accommodate the size of the new MFT zone. Instead, NTFS exhausts the original reserved space before increasing the size of the MFT zone. When the original space is exhausted, NTFS looks for the next contiguous space large enough to hold the additional MFT zone, which can cause the MFT to become fragmented. You can adjust the zone size for the MFT if the defaults do not fit your needs.

NTFS File Record Attributes

Every allocated sector on an NTFS volume belongs to a file. Even the file system metadata is part of a file. NTFS views each file (or folder) as a set of file attributes. File elements such as its name, its security information, and even its data are file attributes. Each attribute is identified by an attribute type code and an optional attribute name.

File and folder records are 1 KB each and are stored in the MFT, the attributes of which are written to the allocated space in the MFT. Besides file attributes, each file record contains information about the position of the file record in the MFT.

When a file's attributes can fit within the MFT file record for that file, they are called resident attributes. Attributes such as file name and time stamp are always resident. When the amount of information for a file does not fit in its MFT file record, some file attributes become nonresident. Nonresident attributes are allocated one or more clusters of disk space. A portion of the nonresident attribute remains in the MFT and points to the external clusters. NTFS creates the Attribute List attribute to describe the location of all attribute records. The table NTFS File Attribute Types lists the file attributes currently defined by NTFS.

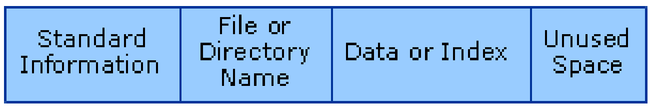
NTFS File Attribute Types

Attribute Type	Description
Standard Information	Information such as access mode (read-only, read/write, and so forth) timestamp, and link count.
Attribute List	Locations of all attribute records that do not fit in the MFT record.
File Name	A repeatable attribute for both long and short file names. The long name of the file can be up to 255 Unicode characters. The short name is the 8.3, case-insensitive name for the file. Additional names, or hard links, required by POSIX can be included as additional file name attributes.

	included as additional file name attributes.
Data	File data. NTFS supports multiple data attributes per file. Each file typically has one unnamed data attribute. A file can also have one or more named data attributes.
Object ID	A volume-unique file identifier. Used by the distributed link tracking service. Not all files have object identifiers.
Logged Tool Stream	Similar to a data stream, but operations are logged to the NTFS log file just like NTFS metadata changes. This attribute is used by EFS.
Reparse Point	Used for mounted drives. This is also used by Installable File System (IFS) filter drivers to mark certain files as special to that driver.
Index Root	Used to implement folders and other indexes.
Index Allocation	Used to implement the B-tree structure for large folders and other large indexes.
Bitmap	Used to implement the B-tree structure for large folders and other large indexes.
Volume Information	Used only in the $Volume system file. Contains the volume version.

NTFS creates a file record for each file and a folder record for each folder created on an NTFS volume. The MFT includes a separate file record for the MFT itself. These file and folder records are 1 KB each and are stored in the MFT. The attributes of the file are written to the allocated space in the MFT. Besides file attributes, each file record contains information about the position of the file record in the MFT. The figure MFT Entry with Resident Record shows the contents of an MFT record for a small file or folder. Small files and folders (typically, 900 bytes or smaller) are entirely contained within the file's MFT record.

MFT Entry with Resident Record



Typically, each file uses one file record. However, if a file has a large number of attributes or becomes highly fragmented, it might need more than one file record. If this is the case, the first record for the file, the base file record, is the case, the first record for the file, the base file record, stores the location of the other file records required by the file.

Folder records contain index information. Small folder records reside entirely within the MFT structure, while large folders are organized B-tree structures and have records with pointers to external clusters that contain folder entries that cannot be contained within the MFT structure.

The benefit of using B-tree structures is evident when NTFS enumerates files in a large folder. The B-tree structure allows NTFS to group, or index, similar file names and then search only the group that contains the file, minimizing the number of disk accesses needed to find a particular file, especially for large folders. Because of the B-tree structure, NTFS outperforms FAT for large folders because FAT must scan all file names in a large folder before listing all of the files.

Last Access Time

Each file and folder on an NTFS volume contains an attribute called Last Access Time. This attribute shows when the file or folder was last accessed, such as when a user performs a folder listing, adds files to a folder, reads a file, or makes changes to a file. The most up-to-date Last Access Time is always stored in memory and is eventually written to disk within two places:

- The file’s attribute, which is part of its MFT record.
- A directory entry for the file. The directory entry is stored in the folder that contains the file. Files with multiple hard links have multiple directory entries.

The Last Access Time on disk is not always current because NTFS looks for a one-hour interval before forcing the Last Access Time updates to disk. NTFS also delays writing the Last Access Time to disk when users or programs perform read-only operations on a file or folder, such as listing the folder’s contents or reading (but not changing) a file in the folder. If the Last Access Time is kept current on disk for read operations, all read operations become write operations, which impacts NTFS performance.

Note

- File-based queries of Last Access Time are accurate even if all on-disk values are not current. NTFS returns the correct value on queries because the accurate value is stored in memory.

NTFS eventually writes the in-memory Last Access Time to disk as follows disk as follows:

Within the file’s attribute
NTFS typically updates a file’s attribute on disk if the current Last Access Time in memory differs by more than an hour from the Last Access Time stored on disk, or when all in-memory references to that file are gone, whichever is more recent. For example, if a file’s current Last Access Time is 1:00 P.M., and you read the file at 1:30 P.M., NTFS does not update the Last Access Time. If you read the file again at 2:00 P.M., NTFS updates the Last Access Time in the file’s attribute to reflect 2:00 P.M. because the file’s attribute shows 1:00 P.M. and the in-memory Last Access Time shows 2:00 P.M.

Within a directory entry for a file
NTFS updates the directory entry for a file during the following events:

- When NTFS updates the file’s Last Access Time and detects that the Last Access Time for the file differs by more than an hour from the Last Access Time stored in the file’s directory entry. This update typically occurs after a program closes the handle used to access a file within the directory. If the program holds the handle open for an extended time, a lag occurs before the change appears in the directory entry.

- When NTFS updates other file attributes such as Last Modify Time, and a Last Access Time update is pending. In this case, NTFS updates the Last Access Time along with the other updates without additional performance impact.

Note

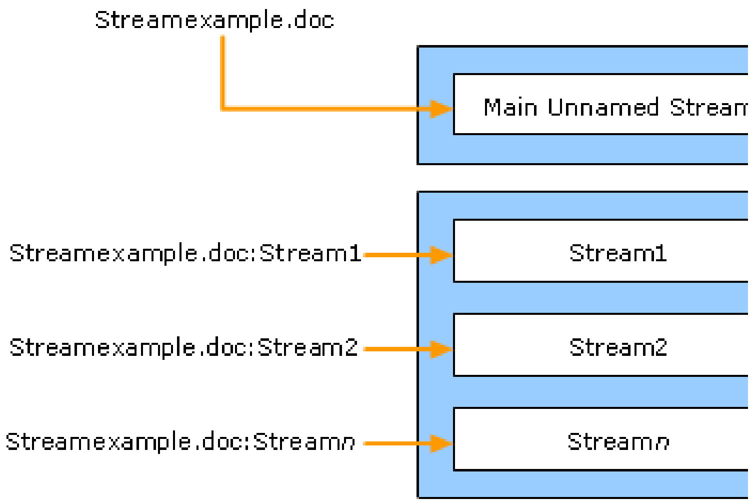
- NTFS does not update a file’s directory entry when all in-memory references to that file are gone.

If you have an NTFS volume with a high number of folders or files, and a program is running that briefly accesses each of these in turn, the I/O bandwidth used to generate the Last Access Time updates can be a significant percentage of the overall I/O bandwidth.

Multiple Data Streams
A data stream is a sequence of bytes. An application populates the stream by writing data at specific offsets within the stream. The application can then read the data by reading the same offsets in the read path. Every file has a main, unnamed stream associated with it, regardless of the file system used.

However, NTFS supports additional named data streams in which each data stream is an alternate sequence of bytes as illustrated in the figure Unnamed and Named Streams. Applications can create additional named streams and access the streams by referring to their names. This feature permits related data to be managed as a single unit. For example, a graphics program can store a thumbnail image of bitmap in a named data stream within the NTFS file containing the image.

Unnamed and Named Streams


Streamexample.doc
Main Unnamed Stream
Streamexample.doc:Stream1
Stream1
Streamexample.doc:Stream2
Stream2
Streamexample.doc:Stream*n*
Stream*n*

FAT volumes support only the main, unnamed stream, so if you try to copy or move Streamexample.doc to a FAT volume or floppy disk, you receive an error message.

NTFS Processes and Interactions

The following sections describe NTFS processes and interactions.

Mounting an NTFS Volume

When mounting an NTFS volume, the MBR executes code to start up the boot sector. The boot sector then executes additional code to mount the volume.

Master Boot Code Startup Process
The MBR contains a small amount of executable code called the master boot code, the disk signature, and the partition table for the disk. During startup, the master boot code performs the following activities:

1. Scans the partition table for the active partition.
2. Finds the starting sector of the active partition.
3. Loads a copy of the boot sector from the active partition into memory.
4. Transfers control to the executable code in the boot sector.

Boot Sector Startup Process

Computers use the boot sector to run instructions during startup. The initial startup process is summarized in the following steps:

1. The system BIOS and the CPU initiate the power-on self test (POST).

2. The BIOS finds the boot device, which is typically the first disk the BIOS finds, unless the controller is configured to boot from a different disk.

3. The BIOS loads the first physical sector of the boot device into memory and transfers CPU execution to that memory address.

If the boot device is on a hard disk, the BIOS loads the MBR. The master boot code in the MBR loads the boot sector of the active partition, and transfers CPU execution to that memory address. On computers that are running Windows Server 2003, the executable boot code in the boot sector finds Ntldr, loads it into memory, and transfers execution to that file.

Note

- Windows Server 2003 cannot start up from a spanned, striped, or RAID-5 volume on dynamic disks. These disk structures cannot be registered into the MBR partition table; therefore, a system volume that uses these structures cannot start.

If drive A contains a floppy disk, the system BIOS loads the first sector (the boot sector) of the disk into memory. If the disk is a startup disk (formatted by MS-DOS with core operating system files applied), the boot sector loads into memory and uses the executable boot code to transfer CPU execution to Io.sys, a core MS-DOS operating system file. If the floppy disk is not a startup disk, the executable boot code displays an error message.

Note

- These messages do not appear on normally functioning systems that are configured to look for the startup files on drive C first. On many computers, an option in the CMOS setup program allows the user to set the sequence of installed disks that the system searches to find the startup files.

If you get similar errors when trying to start the computer from the hard disk, the boot sector might be corrupted.

Initially, the startup process is independent of disk format and operating system. The unique characteristics of operating and file systems become important when the boot sector's executable boot code starts.

Formatting Volumes

During volume format, Windows Server 2003 places key NTFS file system structures on the volume, including the boot sector and the MFT as well as replacing Ntldr. Formatting also aligns clusters at the cluster size boundary.

Formatting a volume will check the integrity of all sectors on the volume during the process, as well as allow you to change the cluster size used on the volume. If a volume is formatted using Quick format, the file system structure on the volume is created, but the integrity of every sector in the volume is not checked.

Converting Volumes

Windows Server 2003 can convert previous versions of NTFS to the new version of NTFS used in Windows Server 2003.

Converting NTFS Volumes Formatted By Using Windows 2000

When Windows Server 2003 first mounts an NTFS volume that was formatted by using Windows 2000, Windows Server 2003 converts the NTFS volume to NTFS 3.1. The conversion consists of changing the NTFS version from 3.0 to 3.1. No other changes are made to existing metadata or files on the volume. However, Windows Server 2003 uses a different header style for new files created on NTFS 3.1 volumes. As a result of this change, some non-Microsoft imaging programs cannot create images of NTFS 3.1 volumes. Contact the manufacturer of your imaging program to find out if a version is available that supports NTFS 3.1 volumes in Windows Server 2003.

Computers running Windows NT 4.0 with Service Pack 4 or later or Windows 2000 can access NTFS 3.1 volumes without any conversion or additional service packs. Also, note that NTFS 3.1 is identical in Windows XP and Windows Server 2003.

Converting NTFS Volumes Formatted By Using Windows NT 4.0 and Earlier

When you upgrade the operating system from Windows NT 4.0 to Windows Server 2003, all local volumes formatted by using the version of NTFS used in Windows NT 4.0 and earlier are upgraded to NTFS 3.1. The upgrade occurs when Windows Server 2003 mounts the volume for the first time after Windows Server 2003 Setup is completed. (The upgrade does not take place during Setup.) Any NTFS volumes that are removed or turned off during Setup, or added after Setup, are converted when Windows Server 2003 mounts the volumes.

The Ntfs.sys driver performs the conversion by determining which version of NTFS is used on the volume and converting the volume if necessary. The conversion takes only a few seconds on any size volume and consists of the following new records in the master file table:

- $Secure, which contains unique security descriptors for all files within a volume.
- $Extend, which is used for extensions such as quotas, reparse point data, and object identifiers. The conversion process also adds three new files the to $Extend directory:
    - $Quota, used for disk quotas.
    - $Reparse, used for reparse points.
    - $ObjID, used for distributed link tracking.

Both $Secure and $Extend take the place of previously unused master file table (MFT) records, so sufficient space always exists in the volume for these two records. However, $Quota, $Reparse, and $ObjID are new additions to the MFT, and you must have enough free space in the volume to contain these files, or the conversion fails.

If the conversion fails, the volume is still available, but you can only perform NTFS-related tasks that were available in Windows NT 4.0 or earlier. To convert the volume to NTFS 3.1, you must free disk space by deleting or moving files and then dismount the volume.

Note

- Removable media that is formatted by using the previous version of NTFS is upgraded after the installation or upgrade process, or when you insert the media and Windows Server 2003 mounts it.

Limitations in Converting Volumes

The conversion is a one-way process. After you convert a volume to NTFS, you cannot reconvert the volume to FAT without backing up your data, reformatting the volume as FAT, and then restoring your data. There should also be a certain amount of free space on the volume and sufficient memory to update the cache.

The following limitations also apply to conversion of a volume from FAT to NTFS:

- In multiple-boot configurations, NTFS volumes are accessible only by using Windows NT 4.0 with Service Pack 4 or later, Windows 2000, Windows XP, or Windows Server 2003.
- When you install Recovery Console onto a volume that is formatted for either the FAT16 or FAT32 file systems, and then convert the volume to NTFS, the Recovery Console no longer runs. This problem occurs because the file-system-specific boot files (in the cmdcons folder of the system volume) that are used to run Recovery Console are not valid for a volume that has been converted to NTFS. You can re-install Recovery Console from the Windows Server 2003 operating system disk after the conversion. You can also use the Windows Server 2003 operating system disk to start Recovery Console.
- Because formatting in Windows Server 2003 aligns FAT data clusters at the cluster size boundary, conversion can preserve the cluster size for the size of the volume (up to 4 KB) instead of using the 512-byte cluster size used in Windows 2000 for converted volumes. The table Cluster Sizes for Volumes Converted to NTFS lists cluster sizes for volumes converted to NTFS.

  Cluster Sizes for Volumes Converted to NTFS

Original FAT Cluster Size	Converted NTFS Cluster Size
512 bytes	512 bytes
1 KB	1 KB
2 KB	2 KB
4 KB and larger	4 KB

File Naming

Windows Server 2003 supports both long and short file names on NTFS volumes.

File Names in Windows Server 2003

Every time you create a file with a long file name, NTFS creates a second file entry that has a similar 8.3 short file name. A file with an 8.3 short file name has a file name containing 1 to 8 characters and a file name extension containing 1 to 3 characters. The file name and file name extension are separated by a period.

File names in Windows Server 2003 can be up to 255 characters and can contain spaces, multiple periods, and special characters that are not allowed in MS-DOS file names. Windows Server 2003 makes it possible for other operating systems to access files that have long names by generating an MS-DOS-readable (8.3) name for each file. These MS-DOS-readable names also enable MS-DOS-based and Windows 3.*x*–based applications to recognize and load files that have long file names. When a program saves a file on a computer running Windows Server 2003, both the 8.3 file name and long file name are retained.

Note

- The 8.3 format means that files can have between 1 and 8 characters in the file name. The name must start with a letter or a number and can contain any characters except the following:
- . " / \ [ ] : ; | = , * ? (space)
- An 8.3 file name typically has a file name extension that is from one to three characters long and has the same character restrictions. A period separates the file name from the file name extension.
- Several special file names are reserved by the system and cannot be used for files or folders: CON, AUX, COM1, COM2, COM3, COM4, LPT1, LPT2, LPT3, PRN, NUL

How NTFS Generates Short File Names

In Windows Server 2003, both FAT and NTFS use the Unicode character set, which contains several prohibited characters that MS-DOS cannot read, for their names. To generate a short MS-DOS-readable file name, Windows Server 2003 deletes all of these characters from the long file name and removes any spaces. Because an MS-DOS-readable file name can have only one period, Windows Server 2003 also removes extra periods from the file name. If necessary, Windows Server 2003 truncates the file name to six characters and appends a tilde (~) and a number. For example, each non-duplicate file name is appended with **~1**. Duplicate file names end with **~2**, then **~3**, and so on. After the file names are truncated, the file name extensions are truncated to three or fewer characters. Finally, when displaying file names at the command line, Windows Server 2003 translates all characters in the file name and extension to uppercase.

Note

- You can permit extended characters by using the **fsutil behavior set** command. You must restart the computer before this setting takes effect. For more information about using the **fsutil behavior set** command, see the topic Fsutil: behavior in Help and Support Center in Windows Server 2003.

When five or more files exist that can result in duplicate short file names, Windows Server 2003 uses a slightly different method for creating short file names. For the fifth and subsequent files, Windows Server 2003:

- Uses only the first two letters of the long file name.
- Generates the next four letters of the short file name by mathematically manipulating the remaining letters of the long file name.
- Appends **~1** (or another number, if necessary, to avoid a duplicate file name) to the result.

This method substantially improves performance when Windows Server 2003 must create short file names for a large number of files with similar long file names. Windows Server 2003 uses this method to create short names for files on both FAT and NTFS volumes.

The following table shows the short file names for files created by six tests.

Short File Names Created by Windows Server 2003 — Example One

Long File Name	Short File Name
This is test 1.txt	THISIS~1.TXT
This is test 2.txt	THISIS~2.TXT
This is test 3.txt	THISIS~3.TXT
This is test 4.txt	THISIS~4.TXT
This is test 5.txt	THA1CA~1.TXT
This is test 6.txt	THA1CE~1.TXT

If the long file names in the preceding table are created in

If the long file names in the preceding table are created in a different order, their short file names are different, as shown in the following table.

Short File Names Created by Windows Server 2003 — Example Two

Long File Name	Short File Name
This is test 2.txt	THISIS~1.TXT
This is test 3.txt	THISIS~2.TXT
This is test 1.txt	THISIS~3.TXT
This is test 4.txt	THISIS~4.TXT
This is test 5.txt	THA1CA~1.TXT
This is test 6.txt	THA1CE~1.TXT

When you delete a file, its short file name is also deleted. When you create new files in the same folder, Windows Server 2003 might re-use short file names that have been deleted. For instance, in Example 1, if you delete the file "This is test 1.txt," and then create a new file called "This is test 7.txt," its short file name becomes THISIS~1.TXT.

If you have a large number of files (300,000 or more) in a folder, and the files have long file names with the same initial characters, the time required to create the files increases. The increase occurs because NTFS bases the short file name on the first six characters of the long file name. In folders with more than 300,000 files, the short file names start to conflict after NTFS uses all of the 8.3 names that are similar to the long file names. Repeated conflicts between a generated short file name and existing short file names cause NTFS to regenerate the short file name from 6 to 8 times.

Compression of Files and Folders

NTFS supports compression on individual files, all files within a folder, and all files within NTFS volumes. Because compression is implemented within NTFS, any Windows-based program can read and write compressed files without determining the compression state of the file. Compression is set in a bit within the file header, while information about compression is stored in the Data file attribute.

Files and directories are compressed and decompressed by passing FSCTL_SET_COMPRESSION code to

DeviceIoControl. A compressed file or directory then has the flag FILE_ATTRIBUTE_COMPRESSED associated with it. Applications can determine a file or directory's compression state using GetFileAttributes.

When a program opens a compressed file, NTFS decompresses only the portion of the file being read and then copies that data to memory. By leaving data in memory uncompressed, NTFS performance is not impacted when it reads or modifies data in memory. NTFS compresses the modified or new data in the file when the data is later written to disk.

The compression algorithms in NTFS support cluster sizes of up to 4 KB. When the cluster size is greater than 4 KB on an NTFS volume, none of the NTFS compression features are available.

Moving and Copying Files or Folders

Moving and copying files and folders can change their compression state. The resulting compression state depends on whether you move or copy the files and whether you move the files between NTFS volumes or to FAT volumes.

NTFS supports compression on one file, all files in a directory, or all files on a volume. Compression is set in a bit within the file header, while information about compression is stored in the Data file attribute. When the bit is set, the system compresses the file when it is saved and decompresses it as needed.

Compression adds overhead to the system because a compressed NTFS file is decompressed, copied, and then recompressed as a new file even when the file is copied in the same computer. Any change to the compression attribute is applied to the files you specify for moving or copying. If you compress all files in the volume, the process might take a few minutes to finish, depending on the size of the volume, the number of files to compress, and the speed of the computer. The delay occurs because Windows Server 2003 must change the compression state of every folder on the volume and compress or uncompress every file on the volume.

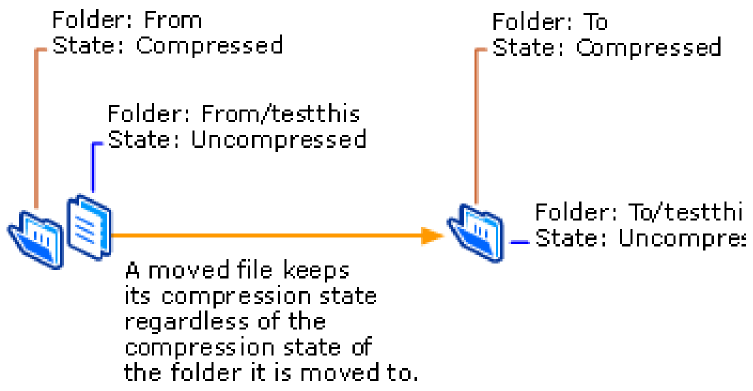
Changing the compression state of folders is relatively fast because for each folder Windows Server 2003 changes only the compression attribute. However, compressing or uncompressing every file on the volume takes longer because NTFS must read data in its current form (compressed or uncompressed) from the disk, convert the data to its new form in memory, and then write the data back to disk.

Moving Files or Folders within an NTFS Volume

When you move an uncompressed file or folder to another

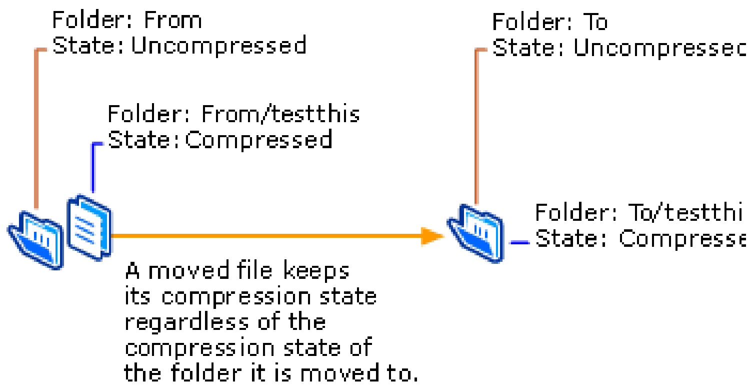
When you move an uncompressed file or folder to another folder on the NTFS volume, the file remains uncompressed. The figure Moving an Uncompressed File to a Compressed Folder illustrates the result of moving an uncompressed file to a compressed folder.

Moving an Uncompressed File to a Compressed Folder



When you move a compressed file to an uncompressed folder, the file remains uncompressed after the move. The figure Moving a Compressed File to an Uncompressed Folder illustrates the result of moving a compressed file or folder to a compressed folder.

Moving a Compressed File to an Uncompressed Folder

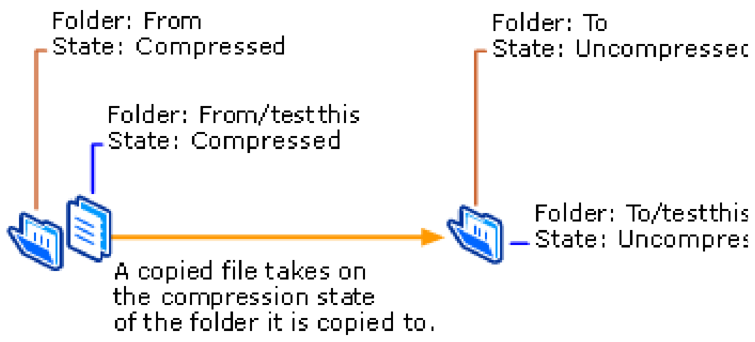


Copying Files or Folders within an NTFS Volume

Copying a file to a folder takes on the compression attribute of the target folder.

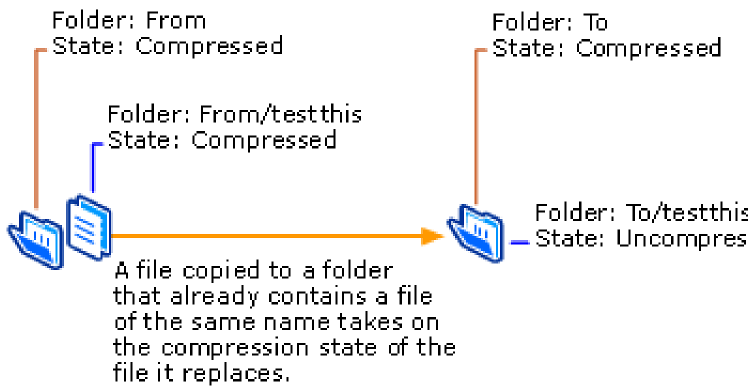
If you copy a compressed file to an uncompressed folder, the file is uncompressed when it is copied to the folder, as shown in the figure Copying a Compressed File to an Uncompressed Folder.

Copying a Compressed File to an Uncompressed Folder



When you copy a file to a folder that already contains a file of the same name, the copied file takes on the compression attribute of the target file, as shown in the figure Copying a File to a Folder that Contains a File of the Same Name.

Copying a File to a Folder that Contains a File of the Same Name



Copying Files between FAT and NTFS Volumes
Files copied from a FAT folder to an NTFS folder take on the compression attribute of the target folder. Compressed files copied from an NTFS volume to a FAT volume or floppy disk are uncompressed.

Mounted Drives on NTFS Volumes

Mounted drives, also known as volume mount points or drive paths, are volumes attached to an empty folder on an NTFS volume. Mounted drives function the same way as any other volume, but are assigned a label or name instead of a drive letter. Mounted drives are robust against system changes that occur when devices are added or removed from a computer. They are not subject to the 26-volume limit imposed by drive letters, so you can use them for access to more than 26 volumes on your computer.

The version of NTFS included with Windows Server 2003 must be used on the host volume. However, the volume to be mounted can be formatted in any file system supported by Windows Server 2003.

One volume can host multiple mounted drives, providing a way for you to easily extend the storage capacity of any particular volume on a Windows Server 2003 system. Users on the local computer or users who connect to it over a network can continue to use the same drive letter for access to the volume, but multiple volumes can be in use simultaneously from that drive letter.

Only NTFS volumes can hold a mounted drive, although any local drive can be mounted on one.

Implementing Mounted Drives

NTFS mounted drives are implemented by using reparse points and are subject to their restrictions. Reparse points are a collection of user-defined data. The format of this are a collection of user-defined data. The format of this data is understood by the application which stores the data, and a file system filter, which you install to interpret the data and process the file. When an application sets a reparse point, it stores this data, plus a reparse tag, which uniquely identifies the data it is storing. When the file system opens a file with a reparse point, it attempts to find the file system filter associated with the data format identified by the reparse tag. If a file system filter is found, the filter processes the file as directed by the reparse data. If a file system filter is not found, the file open operation fails.

The following restrictions apply to reparse points:

- Reparse points can be established for a directory, but the directory must be empty. Otherwise, NTFS fails to establish the reparse point. In addition, you cannot create directories or files in a directory that contains a reparse point.

- Reparse points and extended attributes are mutually exclusive. NTFS cannot create a reparse point when the file contains extended attributes, and it cannot create extended attributes on a file that contains a reparse point.

- Reparse point data cannot exceed 16 kilobytes. Setting a reparse point fails if the amount of data to be placed in the reparse point exceeds this limit.

Hard Links

A hard link is an NTFS-only based link to a given file. When you create a hard link to a file on an NTFS volume, NTFS adds a directory entry for the hard link without duplicating the original file. By creating hard links you can:

- Use the same file name as the original file but appear in different folders.

- Use different file names from the original file but appear in the same folder.

- Use different file names from the original file and appear in different folders.

Because a hard link is a directory entry for a file, an application can modify a file by using any of its hard links. Applications that use any other hard link can detect the changes. However, directory entries for hard links are updated only when a user accesses a file by using the hard link. For example, if a user opens and modifies a file by using its hard link, and the size of the original file changes, the hard link that is used to access the file also shows the new size.

Hard links do not have security descriptors; instead, the security descriptor belongs to the original file to which the hard link points. Thus, if you change the security descriptor of any hard link, you actually change the underlying file’s security descriptor. All hard links that point to the file allow the newly specified access. You cannot give a file different security descriptors on a per-hard-link basis.

Hard links use the Win32 function CreateHardLink to create hard links between files.

Distributed Link Tracking

Distributed link tracking ensures that shell shortcuts and OLE links continue to work after the target file is renamed or moved. When you create a shortcut to a file on an NTFS volume, distributed link tracking stamps a unique object identifier (ID) into the target file, known as the link source. Information about the object ID is also stored within the referring file, known as the link client. Distributed link tracking uses this object ID to locate the link source in any combination of the following events that occur on NTFS volumes within a Windows Server 2003-based domain:

- The link source is renamed.
- The link source is moved to another folder on the same volume or to a different volume on the same computer.
- The link source is moved from one shared network folder to another shared network folder on different computers within the same domain.
- The computer containing the link source is renamed.
- The name of the shared network folder containing the link source has changed.
- The volume containing the link source is moved to another computer within the same domain.

Note

- Distributed link tracking works only on NTFS volumes in computers running Windows 2000, Windows XP, or Windows Server 2003. The NTFS volumes cannot be on removable media.

Distributed link tracking attempts to maintain even those links that do not occur within a domain: cross-domain, within a workgroup, or on a single computer that is not connected to a network. Links can always be maintained in these events when a link source is moved within a computer, or when the network shared folder on the link source computer is changed. Typically, links can be maintained when the link source is moved to another computer; however, this form of tracking is less reliable over time.

Distributed link tracking uses different services for client and server:

- The Distributed Link Tracking Client service runs on all Windows 2000-based and Windows Server 2003-based computers. In computers that are not part of a network, the Client service performs all activities related to link tracking.
- The Distributed Link Tracking Server service runs on Windows 2000 and Windows Server 2003 domain controllers. The Server service maintains information relating to the movement of link sources. Because of this service and the information it maintains, links within a domain are more reliable than those outside a domain. For computers that run in a domain, the Distributed Link Tracking Client service takes advantage of this information by communicating with the Distributed Link Tracking Server service.

The Distributed Link Tracking Client service monitors activity on NTFS volumes and stores maintenance information in a file called Tracking.log, which is located at the root of each volume in a hidden folder called System Volume Information. This folder is protected by permissions that allow only the system to have access to it. The System Volume Information folder is also used by other Windows Server 2003 services such as Indexing Service.

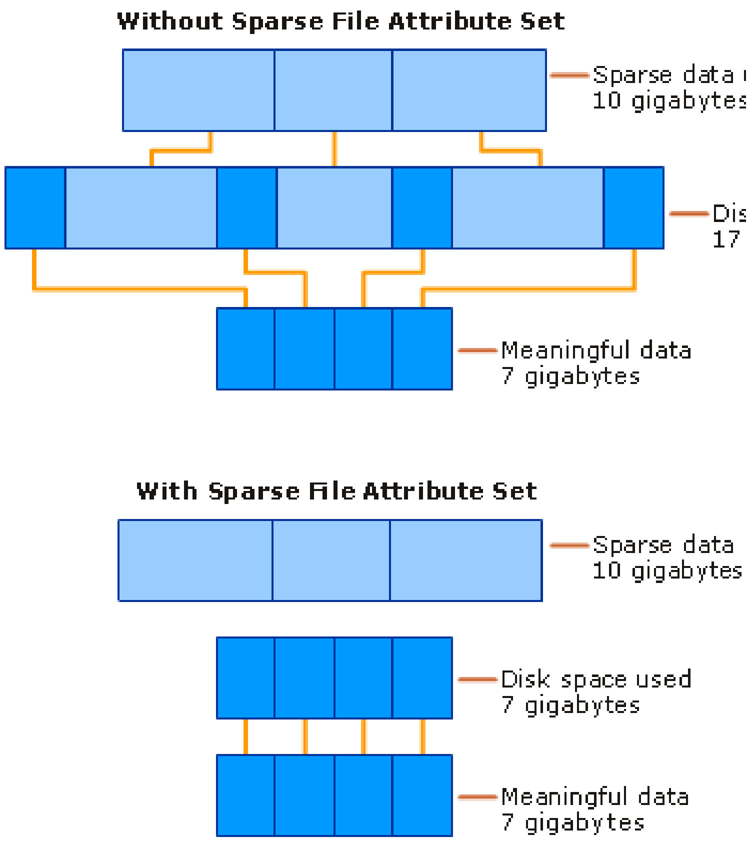
Sparse Files

Sparse files provide a method of saving disk space for files that contain meaningful data, as well as large sections of data composed of zeros. If an NTFS file is marked as sparse, then NTFS allocates disk clusters only for the data explicitly specified by the application. Non-specified ranges of the file are represented by non-allocated space on the disk. When a sparse file is read from allocated ranges, the data is returned as it was stored. Data read from non-allocated ranges is returned as zeros.

File system application programming interfaces (APIs) allow for the file to be copied or backed as actual bits and sparse stream ranges.File system APIs also allow for querying allocated ranges. Programs that implement these APIs then need only to read allocated ranges to recover all

AI is then need only to read allocated ranges to recover all data in the file. The result is efficient file system storage and access. The figure Sparse Data Storage shows how data is stored with and without the sparse file attribute set.

Sparse Data Storage



For example, the properties of a file might show that the file is a 1-GB sparse file. Although the file is 1 GB, it occupies only 64 KB of disk space.

Note

- Only NTFS volumes mounted by Windows 2000, Windows XP, or the Windows Server 2003 family support sparse files. If you copy or move a sparse file to a FAT volume or an NTFS volume mounted by an operating system other than those listed previously, the file is built to its originally specified size. If the required space is not available, the operation fails.

Disk Quotas

You can enable disk quotas to restrict the amount of volume space users take up on remote or local computers with NTFS file systems. Disk quotas uses names from the domain in which the server resides. An administrator can then set disk quotas against those users in the domain.

For additional information about Disk Quotas, see the Disk Quotas Technical Reference.

NTFS Change Journal

As files, folders, and other NTFS objects are added, deleted, and modified, NTFS enters change journal records in streams, one for each volume on the computer.

The total size of all the records currently in the journal varies, but there is a configurable maximum size. The change journal can exceed the maximum size until the size reaches an outer threshold, at which point a portion of the oldest records are deleted until the change journal is restored to its maximum size. The maximum size of the change journal is configurable but cannot be reduced, only increased.

The change journal conveys significant scalability benefits to applications that might otherwise need to scan an entire volume for changes. File system indexing, replication managers, virus scanners, and incremental backup applications can benefit from using the change journal.

The change journal is much more efficient than time stamps or file notifications for determining changes in a particular namespace. Applications that must rescan an entire volume to determine changes can now scan once and subsequently refer to the change journal. The I/O cost depends on how many files have changed, not on how many files exist on the volume.

The APIs are fully documented and can be leveraged by independent software vendors (ISVs). Microsoft uses the change journal in Windows Server 2003 components such as the Indexing Service and File Replication Service. ISVs can use this feature to enhance the scalability and robustness of a range of products including backup, antivirus, and auditing tools.

NTFS File System Recoverability

NTFS is a recoverable file system that guarantees the consistency of the volume by using standard transaction logging and recovery techniques. In the event of a system failure, NTFS runs a recovery procedure that accesses information stored in a transaction log file. The NTFS recovery procedure guarantees that the volume is restored to a consistent state. Transaction logging requires very little overhead.

Recovering NTFS File Structures

NTFS views each operation that modifies a file on a volume as a transaction and manages each one as an integral unit. NTFS might also break a single complex operation into multiple transactions. After a transaction is started, it is either completed, or if an event occurs that causes the operation to fail, it is rolled back, and the NTFS volume returns to its state before the transaction began. Events that can cause an operation to fail include bad sectors, transient low-memory conditions, and disconnected devices.

To ensure that a transaction can either be completed or rolled back, NTFS performs the following steps for each transaction:

1. Records the metadata operations of a transaction in a log file cached in memory.
2. Records the actual metadata operations in memory.
3. Marks the transaction in the cached log file as committed.
4. Flushes the log file to disk.
5. Flushes the actual metadata operations to disk.

Steps 4 and 5 occur in a lazy fashion after the transaction is completed, meaning that the flush operations are not tied to the transaction itself. Instead, NTFS modifies the log and metadata quickly in memory, and then flushes later at a convenient time to boost performance.

NTFS guarantees that the log records containing the metadata operations of the transaction are written to disk before the metadata that is modified in the transaction is written to disk. After NTFS updates the cache, NTFS commits the transaction by recording in the cached log file that the transaction is complete. After the cached log file is flushed to disk, all committed transactions are guaranteed to be completed, even if the system fails before the changes are written to disk.

Note

- Applications can specify the FILE_FLAG_WRITE_THROUGH Win32 flag to instruct the system to write through any intermediate cache and go directly to disk. The system can still cache write operations, but cannot lazily flush them.

If a system failure occurs, NTFS has enough information in the log to complete or abort any partial NTFS transaction. During recovery operations, NTFS redoes each committed transaction found in the log file. Then NTFS locates in the log file the transactions that were not committed at the time of the system failure and undoes each metadata operation recorded in the log file. Because NTFS flushes the log to disk before any metadata changes are written to disk, NTFS has complete information available about any metadata changes that need to be rolled back during recovery.

Note

- NTFS uses transaction logging and recovery to guarantee that the volume structure is not corrupted. For this reason, all file system data is accessible after a system failure. NTFS guarantees user data only if the program used to create the data uses the FILE_FLAG_WRITE_THROUGH Win32 flag. If the program does not use this flag, user data can be lost due to a system failure. If a system failure does occur, NTFS shows either the previous data, the new data, or zeros. Users do not see random data on the volume as the result of a crash.

Caching and Data Recovery

The cache is the area of RAM that contains the most recently used data. When you write data to disk, the lazy-writetechnique in Windows Server 2003 indicates that the data is written when it is still in the cache. Cache memory can also be on the disk controller, such as cache memory available on SCSI controllers, or on the disk unit, such as cache memory available on Advanced Technology Attachment (ATA) disks. The following information can help you decide whether to enable the disk or the controller cache:

- Write caching improves disk performance, particularly if large amounts of data are being written to the disk.

- Control of the write-back cache is a firmware function provided by the disk manufacturer. See the documentation supplied with the disk or disk controller. You cannot configure the write-back cache from Windows Server 2003.

- Write caching does not impact the reliability of the file system's own metadata as long as the firmware provided by the disk manufacturer honors write-through requests issued by the NTFS driver. NTFS instructs the disk device driver to ensure that metadata is written whether or not write caching is enabled. Non-metadata is typically written to disk and can be cached.

- Read caching in the disk does not affect the reliability of a file system.

Cluster Remapping

When NTFS detects a bad sector, NTFS dynamically remaps the cluster containing the bad sector — a recovery technique called cluster remapping — and allocates a new technique called cluster remapping — and allocates a new cluster for the data. If the error occurred during a read, NTFS returns a read error to the calling program, and the data is lost. If the error occurs during a write, NTFS writes the data to the new cluster, and no data is lost.

NTFS puts the address of the cluster containing the bad sector in the bad cluster file, $BadClus, in the MFT so that the bad sector is not reused.

Disk Recovery Operations

NTFS ensures the integrity of all NTFS volumes by performing disk recovery operations whenever a volume is mounted after the computer is restarted or after the volume is dismounted. NTFS also uses a technique called cluster remapping to minimize the effects of a bad sector on an NTFS volume.

NTFS views each operation that modifies a file on a volume as a transaction and manages each one as an integral unit. NTFS might also break a single complex operation into multiple transactions. After a transaction is started, it is either completed, or if an event occurs that causes the operation to fail, it is rolled back, and the NTFS volume returns to its state before the transaction began. Events that can cause an operation to fail include bad sectors, transient low-memory conditions, and disconnected devices.

NTFS guarantees that the log records containing the metadata operations of the transaction are written to disk before the metadata that is modified in the transaction is written to disk. After NTFS updates the cache, NTFS commits the transaction by recording in the cached log file that the transaction is complete. After the cached log file is flushed to disk, all committed transactions are guaranteed to be completed, even if the system crashes before the changes are written to disk.

If a system failure occurs, NTFS has enough information in the log to complete or abort any partial NTFS transaction. During recovery operations, NTFS redoes each committed transaction found in the log file. Then NTFS locates in the log file the transactions that were not committed at the time of the system failure and undoes each metadata operation recorded in the log file. Because NTFS flushes the log to disk before any metadata changes are written to disk, NTFS has complete information available about any metadata changes that need to be rolled back during recovery.

Cleanup Operations on Windows NT–Based Volumes

Because files on volumes formatted by using the version of NTFS included with Windows Server 2003 can be read and written to by Windows NT 4.0 Service Pack 4 or later, Windows Server 2003 might need to perform cleanup operations to ensure the consistency of the data structures of a volume after it is mounted on a computer running Windows NT.

Windows Server 2003 does not perform cleanup operations on volumes previously mounted by using Windows 2000 or Windows XP.

Cleanup operations affect the following features:

Reparse points
Computers running Windows NT 4.0 or earlier cannot access files that have reparse points, so no cleanup operations are necessary. Reparse points are files or directories that have blocks of data called reparse data associated with them.

Disk quotas
If disk quotas are turned off, Windows Server 2003 performs no cleanup operations. If disk quotas are turned on, Windows Server 2003 cleans up the quota information by rebuilding the index. If a user exceeds the disk quota while the NTFS volume is mounted by a Windows NT 4.0 SP4 or later system, and disk quotas are strictly enforced, all further disk allocations of data by that user using Windows Server 2003 fail. The user can still read and write data to any existing file but cannot increase the size of a file. However, the user can delete and shrink files. When usage falls below the assigned disk quota, disk allocations of data can resume.

Encryption
Encrypted files cannot be accessed by computers that are running Windows NT 4.0 or earlier, so no cleanup operations are necessary.

Sparse files
Computers running Windows NT 4.0 or earlier cannot access sparse files, so no cleanup operations are necessary.

Change journal
Computers that are running Windows NT 4.0 or earlier do not log file changes in the change journal. When Windows Server 2003 starts, the change journals on volumes accessed by Windows NT are reset to indicate that the journal history is incomplete. Applications that use the change journal must be able to accept incomplete journals.

Object identifiers
Windows Server 2003 maintains two references to the object identifier: one on the file and one in the volume-wide object identifier index. If you delete a file that has an object identifier, Windows Server 2003 must scan and clean up the entry in the index.

POSIX Compliance

NTFS provides a several features to support the Portable Operating System Interface (POSIX) standard, which is defined by the Institute of Electrical and Electronic Engineers (IEEE) standard 1003.1-1990 (also known as ISO/IEC 9945-1:1990).

NTFS includes the following POSIX-compliant features.

Case-sensitive naming
For example, POSIX interprets README.TXT, Readme.txt, and readme.txt as separate files.

Hard links
A file can have more than one name. This allows two different file names, which can be in different folders on the same volume, to point to the same data.

Additional time stamps
These show when the file was last accessed or modified.

The POSIX subsystem included with Windows NT and Windows 2000 is not included with Windows Server 2003. A new subsystem supporting the broad functionality found on most UNIX systems beyond the POSIX.1 standard is shipped as part of Interix 2.2. The Interix subsystem can be certified to the NIST FIPS 151-2 POSIX Conformance Test Suite.

Note

- You must use Interix-based programs to manage file names that differ only in case. You cannot use standard Windows Server 2003 command-line tools (such as **copy**, **del**, and **move**, or their equivalents in Windows Explorer or My Computer) to manage file names that differ only in case.

Related Information

- Basic Disks and Volumes Technical Reference
- Disk Quotas Technical Reference
- Dynamic Disks and Volumes Technical Reference

Community Additions

Calculation for "Clusters Per MFT Record" and "Clusters Per Index Buffer"

These 2 fields of EBPB can be negative (0x80 to 0xFF).
The explanation says "If the number is negative (80 to FF), then the size of the file record is 2 raised to the absolute value of this number.".
Example #1) This byte field is 0xFF. The absolute value of -1 is +1.
 2^1 is 2. OK
Example #2) This byte field is 0x80. The absolute value of -128 is +128. 2^128 is 3.4e+38. WOW

This last number is greater than NTFS Size Limits -> Maximum file size -> Architecturally: 16 exabytes minus 1 KB (2^64 bytes minus 1 KB)

JeanF J
3/24/2013

Typo

This line: "In theory, the maximum NTFS volume size is 264 clusters minus 1 cluster." should be "In theory, the maximum NTFS volume size is 2^64 clusters minus 1 cluster."

Made me do a double take when I read it. HA

Trevor Arjeski
10/18/2011

Calculating the used disk space exactly

Calculating the used disk space accurately on an NTFS volume is not very easy. The only graphical tool I know that is capable of doing this is TreeSize Professional from http://www.jam-software.com. It is aware of NTFS compression, sparse files, hardlinks and alternate data streams (ADS), for the last two an option needs to be activated as it slows down a scan.
In case of hardlinks there is a special problem: If you have two hardlinks in two different folders pointing to the same file, for which of the folders do you count the file if you don't want to count it twice? It is not possible to find the other names that points to some data, all you know is that it will be in the same filesystem. That means if we see a file having more than one hardlink, we do not know if we already counted it elsewhere. TreeSize solves this problem by counting 1/n of the file's size which has n hardlinks.

The above claim applies only to early versions of Windows. Beginning in Vista, the FindFirstFileName API (http://msdn.microsoft.com/en-us/library/aa364421.aspx) allows finding the various filenames hardlinking to a single file.

Ben Voigt
5/11/2011