



Operating System

Development Considerations for Storage Applications in Windows 2000

Abstract

The Microsoft® Windows® 2000 operating system contains new features and enhancements that affect storage applications. This paper gives an overview of these changes to the storage subsystem, and provides storage application developers with the information they need in order to support and exploit storage-related features in the applications they design.

Beta 3 Note: This paper is based on information available at the time of the Windows 2000 Beta 3 release. Information provided in this paper is subject to change before the final release of Windows 2000.

© 1999 Microsoft Corporation. All rights reserved.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Microsoft, ActiveX, Active Directory, Win32, Windows, and Windows NT are registered trademarks or trademarks of Microsoft Corporation.

Other product or company names mentioned herein may be the trademarks of their respective owners.

*Microsoft Corporation • One Microsoft Way • Redmond, WA 98052-6399 • USA
0399*

INTRODUCTION	1
Storage Features New to Windows 2000	1
UNDERSTANDING WINDOWS 2000 STORAGE FEATURES.....	5
File System Support	5
Volume Management	6
Reparse Points	9
NTFS Directory Junctions	12
Volume Mount Points	13
Storage Compression	15
Single Instance Store	15
Encrypting File System	16
Encrypting File System Implementation Details	17
NTFS Sparse File Support	18
Disk Quotas	21
Remote Storage	21
Removable Storage	23
File System Replication	29
System File Protection	29
Special Cases for Defragmentation	30
WINDOWS 2000 FEATURES THAT AFFECT STORAGE.....	31
The Microsoft Management Console	31
Unattended Installation or Automated Installation	31
Indexing Service	31
Installable File System Kit	32
Cluster Service	32
Microsoft Distributed File System	33
SUMMARY	36
FOR MORE INFORMATION	37

INTRODUCTION

The Microsoft® Windows® 2000 operating system introduces a number of new features and architectural enhancements that affect storage applications. Many of these features require direct changes to components that use the storage sub-system. In addition, other general operating system enhancements, although not directly related to storage, may require changes to storage applications as well.

This paper provides an overview of the new features and architectural changes in the Windows 2000 storage sub-system. The purpose of this paper is to help storage applications developers to support and exploit these storage-related features in their applications.

Although this paper discusses many application programming interfaces (APIs), please refer to the [Microsoft Platform SDK](#) for complete details regarding the Win32® API.

Storage Features New to Windows 2000

The following is a list and brief definition of the new storage features and enhancements in the Windows 2000 operating system. These features, their impact, and programming tips that take advantage of them are covered in more depth later in this document.

- **File Systems**—The Windows 2000 Server operating system supports FAT16, FAT32, Compact Disk File System (CDFS), Universal Disk Format (UDF), and the Windows NT File System (NTFS).
- **Volume Management**—The management of local disk volumes is enhanced in Windows 2000 with the introduction of a new volume manager, in addition to the legacy fault tolerant disk manager. Enhancements to volume management include remote administration using a Microsoft Management Console (MMC) snap-in and an expanded fault tolerance feature set. The new volume manager introduces a new disk partition layout.
- **NTFS**—The newest version of NTFS has a new on-disk file system structure that is included in the Windows 2000 operating system. This new structure is required to support many storage enhancements including volume mount points, remote storage, file system encryption, sparse files, and disk quotas. Windows 2000 setup handles the on-disk upgrade, and other system utilities are available for conversion of legacy file systems to NTFS. Windows NT® 4.0, Service Pack 4 and later includes an updated NTFS driver that may mount volumes with this updated NTFS format.
- **Reparse points**—These are new file system objects in NTFS. Reparse points are based on a definable attribute containing user-controlled data. When used in conjunction with installable file system filter drivers (IFS filter drivers), reparse points enable a wide variety of enhanced file and/or directory functionality. Reparse points are used to implement NTFS directory junctions, Remote Storage, and other features in the Windows 2000 operating system. Reparse points offer a mechanism for providing enhancements to storage applications when used in conjunction with ISV-provided file system filter drivers. At a

minimum, storage applications need to be aware of behaviors associated with reparse points.

- **NTFS directory junctions**—These are NTFS directories that can be resolved to any local namespace. Directory junctions provide a very powerful tool for system administrators, but are not generally deployed—they can only be created with the LINKD.EXE tool in the Windows 2000 Resource Kit. Because NTFS directory junctions can be used to make the storage namespace span volumes, they may present new subtleties for application developers.
- **Volume mount point**—Volume mount points allow a volume to be mounted on an existing folder rather than at the root of a new drive letter. Establishing a volume mount point for an empty NTFS directory allows an administrator to graft new volumes into the namespace without requiring additional drive letters. Volume mount points are robust when system changes occur, such as when devices are added or removed from a machine. Storage applications must be prepared to handle dynamic changes in a namespace that are a result of volume mount points.
- **Change Journal**—This log or change journal is a new, powerful feature in NTFS that tracks file additions, deletions, and changes. APIs allow changes to be viewed by an application without resorting to namespace traversal. The Change Journal provides storage applications with an efficient method for determining changes in a given namespace; for example, a backup application could consult the Change Journal to build its list of files prior to an incremental backup.
- **Single Instance Store**—This feature of the Windows 2000 operating system is automatically added to the server when Remote Installation Services are enabled. Single Instance Store (SIS) reduces duplicate data on volumes by replacing duplicate files by file links that point to a single instance of a file in a common store. Single Instance Store is used by internal services in the Windows 2000 Server operating system, such as the Remote Installation Service (RIS).
- **Encrypting File System**—Encryption capabilities are included in the Windows 2000 operating system as an Installable File System filter driver for NTFS. Raw import and export APIs are provided for processing raw, encrypted data for backup, replication, and so forth. Encryption and compression are mutually exclusive file attributes.
- **Sparse Files**—Support for these types of files is enhanced in NTFS. Files denoted with a new user-controlled file system attribute take advantage of this feature. NTFS deallocates sparse data streams and only maintains meaningful data as allocated. On file access, the file system yields allocated data as actual and deallocated data as zeros. APIs allow application developers to bypass file expansion and directly read the allocated ranges. This enables applications to avoid processing large streams of zeros yielded by the file system and to copy or move potentially huge files with sparse data streams in an efficient manner.
- **Disk Quotas**—Using disk quotas, a new feature in NTFS, provides administrators with more granular control of network-based storage. Quotas

allow implementation of both hard and soft storage limits on a per user basis for a given NTFS volume. Quotas affect the free disk space reported to applications.

- **Remote Storage**—Data storage in the computer running Remote Storage in a Windows 2000-based system is designed to lower storage costs by trading off latency against media cost. Based on established criteria, data is automatically migrated from local volumes to tape libraries. Reparse points specifically related to remote storage are used on the primary storage to denote migrated files. Because file recall does involve latency, it is the responsibility of storage applications programmers to refrain from inadvertently or unnecessarily recalling files. Examples of such applications are backup and anti-virus utilities.
- **Link Tracking and Object Identifiers**—These are new features in the version of NTFS used in the Windows 2000 operating system. The link tracking service is provided so that client applications can track link sources that have been moved either locally or within a domain. As a result, clients that subscribe to the link tracking service can maintain the integrity of their references because the objects referenced can be moved transparently. Shell shortcuts and ActiveX® document links are examples of such applications. NTFS files can be referenced by a unique object identifier (OID). Link tracking stores a file OID as part of its tracking information.
- **Removable Storage**—This is a core service in the Windows 2000 operating system that manages removable storage media. Removable Storage provides access to storage devices through a single set of APIs. Removable Storage eliminates the need for ISVs to support bulk media devices on a per device basis. More importantly, in the Windows 2000 operating system, multiple applications can share a bulk storage device. Because of the bulk media support in Removable Storage, storage applications can concentrate on customer-related features rather than hardware issues.
- **The Installable File System Kit**—This developer's kit provides sample kernel mode file system and file system filter drivers. It includes the necessary materials for developers to develop file systems and file system filters. For more detailed information, see the home page for the [Windows 2000 Installable File System \(IFS\) Kit](#).
- **File Replication Service**—This service is used in the Windows 2000 operating system to replicate system files in Active Directory™ directory service, as well as provide added value to services such as the Distributed File System (Dfs).
- **Backup** functionality is included in the Windows 2000 operating system. The Windows 2000 Server Backup utility provides support for Active Directory, Remote Storage, and other new functions in the Windows 2000 operating system. Backup is no longer limited to just tape drives as a backup media. Backup can now use tape drives and tape libraries. In addition, it can store a backup image as a conventional file on a hard disk, removable disk, or shared folder.
- **System File Protection**—This feature of the Windows 2000 operating system prevents the replacement of essential system files by protecting operating

system files. In the event that one of these files is deleted or over-written, System File Protection replaces the file with the original from a cache that it maintains, or from the install media if the cached version is unavailable.

Other Windows 2000 features such as Indexing Service, the Distributed File System (Dfs), Active Directory, Windows Clustering, and Plug and Play have indirect effects on storage applications. Storage application vendors and network redirector providers should consider how their applications would co-exist with these new components.

UNDERSTANDING WINDOWS 2000 STORAGE FEATURES

Several of the new storage features of the Windows 2000 operating system are enhancements to existing technology; others are completely new. It is important to consider each feature from both an architectural and an implementation perspective. This section describes the specifics of each storage feature and the programming knowledge necessary to develop successful third-party storage applications for the Windows 2000 operating system. Although the examples below may refer to a specific type of storage application, the scope of a specific feature is not limited to this particular application type. A later section examines new features of Windows 2000 that independent software vendors (ISVs) should consider when designing and programming storage applications.

File System Support

This section describes the types of file system support provided by the Windows 2000 operating system.

FAT16 and FAT32

Support for FAT16 (or FAT) is provided to maintain an upgrade path for previous Windows-based products. The maximum volume size for a FAT16 partition in the Windows 2000 operating system is 4 gigabytes; which is unchanged from previous versions.

Windows 2000 introduces FAT32 file system support, which offers the same format and features as provided in the Windows 95 OSR2 and Windows 98 operating system. The on-disk FAT32 format enables smaller cluster sizes, and yet supports volumes up to 2 terabytes in size. Implementation limits in Windows 98 and Windows 95 OSR2 currently limit this to 127.53 GB in size. The reduced cluster size in FAT32 results in a 20 to 30 percent increase in disk space efficiency, as compared to FAT16 volumes. Since the Windows 2000 operating system supports the far more recoverable and scaleable NTFS file system, the Windows 2000 FAT32 implementation is limited to creating 32 GB volumes; however, any size FAT32 volume can be mounted and used.

Compact Disk File System

The Windows 2000 operating system continues to provide support for Compact Disk File System (CDFS) (an implementation of ISO 9660).

Universal Disk Format

The Universal Disk Format (UDF) file system is new in the Windows 2000 operating system. The primary intention of UDF in Windows 2000 is to provide read-only support for DVD media. UDF is a standards-based file system that is ISO 13346 compliant.

Windows NT File System

The new storage features in the Windows 2000 operating system require an update to the on-disk format for NTFS. Some features that require this format are reparse points, quotas, and file object identifiers. The upgrade to NTFS occurs automatically during installation of Windows 2000 for all existing NTFS volumes. FAT and FAT32

volumes can be converted to NTFS at any time.

Note that versions of Windows NT version 4.0 and earlier do not natively recognize this format. In situations where a Windows NT 4.0 installation needs to read an NTFS volume created or upgraded by Windows 2000, the Windows NT 4.0 installation must have support installed for the new on-disk format. This support is available in Windows NT 4.0 Service Pack 4. Without Service Pack 4, these volumes will be treated as unknown. (Features included in Service Pack 4 are also included in later Service Packs.) Configurations affected by this situation include:

- Volumes on removable media.
- Volumes used with multi-boot configurations.
- Volumes shared within clustered configurations.

Network clients using NTFS volumes on file/print servers are not affected. Proper planning is required to ensure that NTFS volumes are visible to all configurations.

NTFS Version Awareness

Applications that assume knowledge of a volume's on-disk format need to be updated. You can add this support by using the **NTQueryVolumeInformation** API. The file system version indicates the types of file objects that can be encountered on a volume. Later sections of this paper will explain the types of issues that can be encountered by applications that make assumptions. Details are provided in the Microsoft Platform SDK.

Volume Management

Volume management in the Windows 2000 operating system is responsible for the creation, deletion and maintenance of storage volumes in a system. Windows 2000 significantly enhances the volume management architecture. The goal of this updated architecture is to improve the manageability and recoverability of volumes in a Windows 2000 environment. This is achieved through the introduction of a logical disk manager (ldm) in addition to the current fault tolerant disk manager (ftdisk).

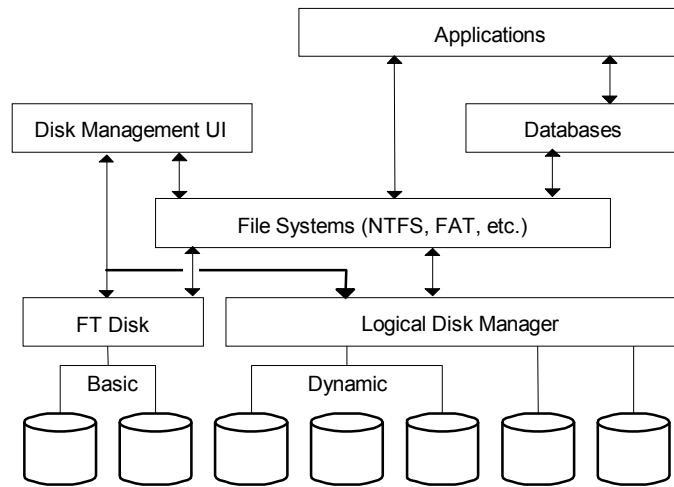


Figure 1. Volume management architecture

FT Disk

The FT Disk driver manages all partitions and fault-tolerant volumes originally created using the MS-DOS®, Windows, or Windows NT operating systems.

Disks managed by FT Disk are called basic disks. Basic disks may contain simple partitions, extended partitions, or fault-tolerant sets. All partitions on a basic disk are hard partitions—the underlying disk structure is statically allocated into contiguous disk cylinder aligned extents.

Simple partitions and extended partitions may be created with Windows 2000, but once created the size is fixed. A basic disk may be converted to a dynamic disk at any time.

Logical Disk Manager

Logical Disk Manager (LDM) is designed to reduce the total cost of ownership (TCO) and reduce system downtime by being highly flexible and available.

Table 1. Logical Disk Management

LDM Features	LDM Benefits
Better manageability and recoverability: Disk RAID objects (RAID 5, 0 and 1) can be created, broken and rebuilt online.	No downtime required to reconfigure the level of performance and redundancy you require for a file system. No additional hardware required to achieve online RAID management
Filesystems can be extended over new volumes while the filesystem is in use	No downtime required to extend the size of a disk volume
Microsoft Management Console (MMC) snap-in allows both local and remote management	The MMC architecture allows this function to be delegated to appropriate administration staff
LDM-managed disks are self describing: changes to SCSI Ids, LUNS, and host adapter ordering does not compromise	Ease of storage system reconfiguration

Inside Logical Disk Manager

Disks managed by LDM are called *dynamic disks*. Dynamic disks may contain simple volumes, concatenated volume sets, stripe sets, mirror sets, or RAID 5 (parity stripe) sets. Simple volumes may be extended dynamically at any time without reboot. A simple volume may be converted to a mirror set at any time or a mirror removed from a mirror set again without requiring system reboot.

LDM manages dynamic disks as a collection or group. Each disk in the group contains information on all other disks in the group in a small database. (Small is defined as one cylinder, typically between 1 and 8 MB in size.) Changes to the disk configuration cause a transactional update of the database on each disk so that LDM can robustly control the recovery of mirror or parity stripe sets. It also means that volume configuration information is no longer contained in the registry.

LDM creates volumes on dynamic disks using soft partitions. The disk contains only one type 0x42 hard partition or extent. The LDM database contains all actual volume configurations. When basic disks are converted to dynamic disks, the underlying hard partitions are conserved and these volumes cannot be extended. For disks other than system or boot disks, the recommended way to convert from basic disks to dynamic disks is to first backup or copy all basic disk data to another dynamic disk; delete all basic partitions on the basic disk; and then convert the basic disk to dynamic.

Dynamic disks may be transported between systems. The disks are self-identifying, and the receiving system can determine all volume configuration information. Importing disks from one system to another causes the databases to be merged transactionally with no loss or corruption of the configuration data.

No comprehensive API exists to manage the creation and configuration of dynamic volumes. Many operations can currently be performed only using the GUI.

Programming with Windows 2000 Volumes

The following describes the types of operations that can be performed on volumes. (See Microsoft Platform SDK for more information).

A new Win32 API, **FindFirstVolume/FindNextVolume**, enables the enumeration of all volumes in a system. This API returns names in the form

```
<\\?\volume{<GUID>}\  
>
```

The API enumerates all of the devices of type MOUNTDEV_MOUNTED_DEVICE_GUID, and then queries the mount point manager for the sticky name. The name returned by this API ends in a backslash (\) so that creating a file with this name opens the root directory on the volume. To open the volume, you must remove the trailing backslash.

FindVolumeClose is used to close the open handle.

Volume IDs can be set using the **NtSetVolumeInformationFile** API and can be

retrieved using **NtQueryVolumeInformationFile**.

Reparse Points

Reparse points are NTFS objects that carry a specialized attribute containing user-defined data. Reparse points are used to extend functionality in the I/O subsystem. One reparse point is allowed for each file or directory. Remote Storage and NTFS directory junction points are examples of functionality based on reparse points.

How Does an I/O Reparse Work?

Microsoft assigns reparse tags to ISVs to differentiate among reparse points. When a file system object with a reparse point attribute is encountered during path name resolution, it is passed back up the driver stack for an I/O reparse. The file system filter driver handles the I/O reparse, which includes identifying the ISV reparse tag. Vendor-specific file system filter drivers are responsible for executing specific I/O functionality. These drivers use the reparse tag and a GUID to identify the I/O calls for which they are responsible. Although the reparse tag itself is unique, the GUID provides an additional (and required) identification.

The following steps and the diagram in Figure 2 illustrate one example of how a reparse point works.

1. The user mode application code issues a **CreateFile** call to open a directory on an NTFS volume.
2. The call reaches the file system object and encounters the reparse point attribute containing the reparse tag.
3. Each installable file system filter driver in the Windows 2000 I/O stack examines the tag associated with the reparse point. If there is a match, the associated file system filter driver intercepts the call. (File system filter drivers can examine both inbound and outbound calls. In this example, there is no functionality associated with the inbound call so it is not referenced in the diagram.)
4. The filter driver executes the enhanced functionality associated with the reparse point. In the case of the Windows 2000 Remote Storage Service, the file system filter driver calls its service to process a request directed to data stored on remote media.
5. The file system driver returns the call to the calling application. Because this example involves remote data, the system retrieves it before returning a handle to the calling function.

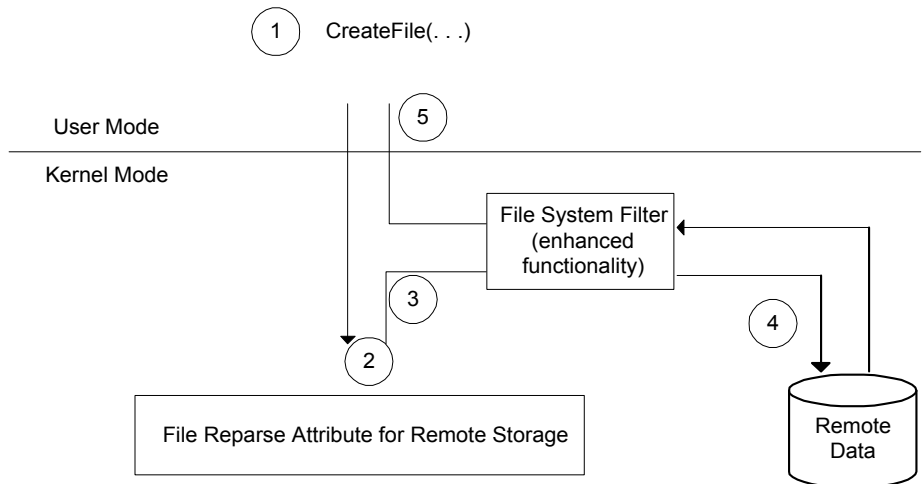


Figure 2. Reparse points

There are two points to consider in the example above:

- If the reparse point were not present, the **CreateFile** function would result in normal behavior.
- If the flag `FILE_OPEN_REPARSE_POINT` were passed to the **CreateFile** function, the call would bypass the enhanced functionality and simply return a handle to the reparse point data, enabling the call to process it directly. A backup application, for example, might access this data directly. (Not all file system filter drivers support this flag.)

Reparse Point Awareness

Applications that manipulate files or directories must be aware of the following issues introduced by reparse points in NTFS:

- Files and directories have additional behavior as a result of reparse points. Reparse point processing is transparent. Reparse point functionality can be disabled by setting the flag `FILE_OPEN_REPARSE_POINT` during **CreateFile** calls. In doing so, a handle to the underlying NTFS object containing the reparse point is retrieved. This is generally not desired unless the application issuing the **CreateFile** call is specifically manipulating the underlying object itself without using the enhanced functionality.
- There is no guarantee that a reparse point will successfully resolve. The open call will fail if the corresponding Installable File System Filter driver (IFS Filter) is not installed.
- When volume mount points and directory junction points present, the file system namespace is no longer guaranteed to be a directed acyclic graph, as it is possible to define reparse points that create cycles in the namespace. This is not a problem on any individual **CreateFile** call because new functionality in the Windows 2000 kernel will stop the recursion after a finite number of loops, and return an error code. However, any application that opens directories and files in a namespace in a recursive manner must deal with this possibility, as the

kernel code cannot provide assistance here. This is because **CreateFile** is being called more than once in an environment that has potential loops. The recommended strategy for dealing with this possibility is to detect name grafting reparse points while traversing through the namespace opening files. This approach guarantees the application will always be acting upon the local namespace, and that entrance into other namespaces is by choice. If entering the new namespace, code must be implemented to prevent infinite loops caused by circular references of reparse points.

- Reparse points established at directories require the directory to be empty. Attempting to create a reparse point at a directory that is not empty will result in failure.
- **CreateFile** with FILE_OPEN_REPARSE_POINT attribute, **DeleteFile**, and **RemoveDir** calls will succeed even if the appropriate file system filter driver is not present. This is also true of reparse points that are unresolvable. By default, these Win32 calls first attempt to operate without bypassing the file system filter driver. Still, if the reparse point cannot be resolved, they operate by bypassing the file system filter driver's functionality.

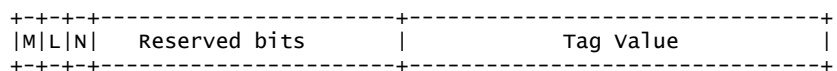
Tracking Reparse Points on a Volume

If an application is specifically using Windows 2000-based APIs that support an object identifier as a parameter, it is possible to track reparse points for an entire volume without having to enumerate all files. \$Reparse is an index that has been added to NTFS to allow applications to maintain efficient lists of files and directories on a volume that contain a reparse point attribute. This index, used in conjunction with the Change Journal, provides the means for very efficient tracking of changes to file system objects that have a reparse point attribute. Access is the same as it is for all indices using the \\$.Extend directory. See the [Microsoft Platform SDK](#) for details.

Reparse Point Tag Structure

I/O calls use tags to differentiate types of reparse points. These tags are examined by installable file system filter drivers that determine the owner of the reparse point and take appropriate action. ISVs must apply directly to Microsoft for a reparse point tag value.

Reparse tags are ULONG with minimal structure built into them. The low 16 bits are used to determine the type of reparse point; the high bits are composed of 3 attribute bits and 13 reserved bits. The tag is as follows:



where

- **M** is the Microsoft bit. When set to 1, it denotes a tag owned by Microsoft. All ISVs tags must have this bit set to 0.
- **L** is the high-latency bit. When set to 1, a reparse point with this tag is expected to have a long latency in retrieving the first byte of data.

-
- **N** is the Name surrogate bit. When set to 1, the file represents another named entity in the system.

The Windows 2000 operating system I/O also has a size constant that limits the amount of data that can be placed in a reparse point. A set operation fails if the amount of data to be placed in the reparse point exceeds this limit. The maximum size of the reparse point data is (16 * 1024) bytes.

Reparse Point Tag Values

Microsoft requires ISVs to use registered reparse point in commercial Windows 2000-based software. ISV reparse point tags consist of three parts:

- Identification bits
- Reserved data
- Vendor tags

The reparse point GUID, although used in combination with the tag value to uniquely identify the reparse point, is stored in the reparse point data itself. See the [Microsoft Platform SDK](#) or the IFS Kit for information regarding implementation of reparse points and associated tag and GUID data.

Programming with Reparse Points

Although the Microsoft Platform SDK covers the API in depth, the following are some changes that directly apply to reparse points.

- **NtQueryVolumeInformation** indicates whether or not a volume supports reparse points via a new flag, `FILE_SUPPORTS_REPARSE_POINTS` (0x00000080).
- Any application that wants to identify reparse points will have to do the additional work before processing file system objects.
- **FindFirstFile** and **FindNextFile** return the flag `FILE_ATTRIBUTE_REPARSE_POINT`, as well as the tag of a reparse point.
- `FILE_FLAG_OPEN_REPARSE_POINT` is a flag that can be used during an open operation to inhibit the enhanced behavior of a reparse point.

NTFS Directory Junctions

An NTFS directory junction is an NTFS directory with a special type of reparse point associated with it. An NTFS directory junction can be mapped to any local or remote target directory.

Awareness of NTFS Directory Junctions

The following notes apply specifically to NTFS directory junctions:

- NTFS directory junctions are identified by a special reparse point tag.
- On individual open operations, the Windows 2000-based computer detects name cycles. Cycles created by multiple, consecutive opens require detection routines in the application itself because the Windows 2000 operating system provides no assistance in detecting these potentially problematic situations.
- All information about a volume where a directory junction was encountered (file

system type, free disk space, and so forth) becomes invalid, as the junction will possibly enter a new namespace. Attributes that might be affected are free disk space, compression, and file system version.

NTFS Directory Junctions vs. Distributed File System Shares

Although the Distributed File System (Dfs) shares and NTFS directory junctions have some features in common, there are important differences to note. The following list clarifies some key distinctions:

- NTFS directory junctions are intended to be tools for grafting local storage volumes or network shares into a single local namespace. DFS is designed to graft network shares into a single DFS namespace.
- DFS functionality is integrated with the Active Directory directory service, while NTFS directory junctions are not.
- NTFS directory junctions rely on NTFS. Dfs is file system agnostic and is compatible with Windows NT 4.0.
- NTFS directory junctions do not require a client component. Dfs requires a client component.
- Dfs hierarchies offer load balancing and fault tolerance via replication across systems; NTFS directory junctions have no inherent load balancing or fault tolerance features as they only resolve to one possible target volume. An NTFS directory junction can, however, resolve to a Dfs share.

Volume Mount Points

Volume mount points are file system objects that resolve to internal Windows 2000 volume device names in a persistent manner. Placing a volume mount point on an NTFS directory causes the storage subsystem to resolve the directory to a specified local volume. This mounting is done transparently and does not require a drive letter to represent the volume. In the Windows 2000 operating system, a mount point always resolves to the root directory of the desired volume. Volume mount points require NTFS because they are based on NTFS reparse points.

An example of mount point usage (shown in Figure 3) is when a portable computer user has two physical drives: Drive 1 is for the operating system and personal use and Drive 2 is for storing work-related data. Because most personal productivity tools are set to open and save work at a common directory, such as C:\My Documents, it would be disruptive to have to change drives depending on whether or not personal or work-related data was being used. An NTFS volume mount point, therefore, could be placed in a C:\My Documents\Work directory so that this directory and all subdirectories would use physical disk space on Drive 2. Changing directories to C:\My Documents\Personal, however, would access Drive 1.

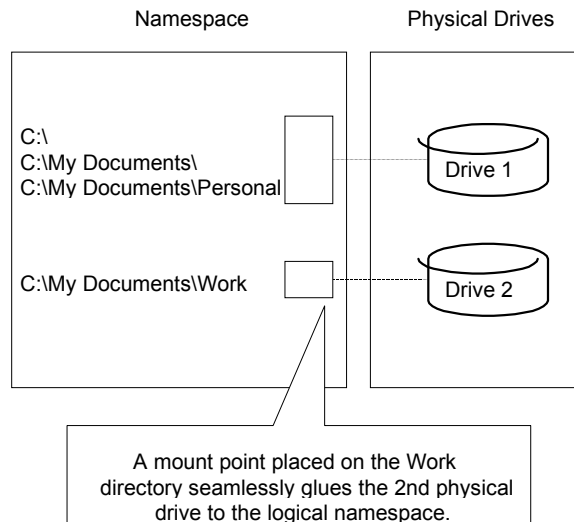


Figure 3. Mount point example

Multiple volume mount points can target any given volume.

Volume mount points are sticky in that the Windows 2000 operating system automatically prevents resolution problems due to changes in the internal device name of the target volume name; for example, due to hardware device reconfiguration activities. This means that for all practical purposes, a mount point is the target volume in the same way that a drive letter is the target volume. (For more information, see the [“Volume Management”](#) section in this paper.)

No additional precautions are required for volume mount points beyond those already discussed for NTFS directory junctions.

Programming with Volume Mount Points

Volume mount points are tracked by the Volume Mount Point Manager, which maintains a database of all volume mount points in a Windows 2000 system. A new set of Win32 APIs has been developed to enumerate and administer volume mount points maintained by the Volume Mount Point Manager. This allows volume mount points to be tracked on a given volume without having to traverse the file system and check returned attributes.

The volume enumeration APIs (see the “Volume Management” section of this white paper) can be used to obtain a volume GUID. **GetVolumeInformation** determines whether the volume supports mount points. **FindFirstVolumeMountPoint**, **FindNextVolumeMountPoint**, and **FindVolumeMountPointClose** return the most recent list of volume mount points for a given volume.

GetVolumeNameForVolumeMountPoint can be used to retrieve the volume name to which a mount point resolves. In Windows 2000, this volume name does not have to be a drive letter.

Storage Compression

Storage compression has not changed in the Windows 2000 operating system.

Compressed files on an NTFS volume are non-encrypted files that carry the FILE_ATTRIBUTE_COMPRESSED attribute. Files that carry the attribute FILE_ATTRIBUTE_ENCRYPTED cannot carry the attribute FILE_ATTRIBUTE_COMPRESSED at the same time. The two attributes are mutually exclusive.

Change Journal

For each volume, NTFS tracks information about added, deleted, and modified files using a new feature in NTFS called the Change Journal. The Change Journal identifies the file and the modification. This is useful for applications that need to know what happened on a particular volume. File system indexing, replication managers, remote storage, and incremental backup applications are a few examples of applications that can benefit from the Change Journal.

How does the Change Journal Work?

The Change Journal is a sparse stream where only a small active range of the file uses any disk allocation. The active range initially begins at offset 0 in the stream and moves monotonically forward through the file. The Unique Sequence Number (USN) of a particular record represents its virtual offset in the stream. As the active range moves forward through the stream, earlier records are deallocated and these records become unavailable. The size of the active range in a sparse file can be adjusted.

Change Journal Awareness

The Change Journal does not affect a storage application unless it is specifically used. Thus, the following only applies to applications that read the change journal:

- The Change Journal operates in a bounded space. It is also based on a sparse data stream that allows for deallocation from the front of a file. Thus, change entries can be removed and any application that depends on these entries must be prepared to deal with this event.
- The Change Journal records data on a per volume basis.
- The Change Journal is only applicable to NTFS volumes.

For more information about the Change Journal, see the [Microsoft Platform SDK](#).

Single Instance Store

Single Instance Store (SIS) is a base operating system component of the Windows 2000 operating system that is tasked with managing disk space by exploiting the existence of duplicate files on a volume. The result is that duplicate files are replaced with links to a single common store file that contains the actual file data. The Single Instance Store is a service used by the Remote Installation Service in Windows 2000.

How is the Single Instance Store Implemented?

The Single Instance Store consists of a base tracking service, a kernel mode file system filter driver, SIS-specific reparse points, and sparse files. In single instance storage, duplicate files are replaced with a sparse file and a reparse point specific to SIS. All backing files are stored in a protected directory referred to as the *common store*. The actual file is renamed with a 128 bit globally unique identifier (GUID) when migrated to the common store. File migration is accomplished via an initial scanning process and is maintained by a monitoring process performed by the Single Instance Store.

Duplicate files processed by SIS are replaced by both a sparse file with a reparse point. The sparse file is a file that logically represents the amount of data contained in the backing file, however, it has no allocated regions. Thus, the file appears to be the size of the backing file, yet, it occupies very little space on the volume. The reparse point contains an SIS specific tag and SIS data that is recognized by the SIS file system filter driver in kernel mode. The reparse point also contains data including the GUID that represents the actual file in the common store.

When a SIS file link is opened, the SIS file system filter driver intercepts incoming calls, parses data that indicates the location of the backing file, performs the I/O action on the backing file, and passes the handle of the file to the original user mode call. (See sections on sparse files and reparse points in this paper for more details on these components of the Single Instance Store solution).

Encrypting File System

File/directory level encryption is implemented in NTFS for enhanced security in NTFS volumes. Today, NTFS provides C2 compliant security for files and directories on NTFS volumes. However, as with any security scheme enforced purely by the operating system software, there is no security guarantee if you lose physical possession of the storage media. It is possible for an unauthorized user to remove a physical storage volume, and mount it on another Windows 2000-based system. Once mounted by another system, the administrator of this system can take ownership of all data, bypassing NTFS security. Portable computers are particularly vulnerable in this scenario.

Encrypting File System (EFS) solves this problem by storing actual data in encrypted form, thus providing security in cases where the storage media is removed from a Windows 2000-based system. EFS implements encryption keys on a per domain user basis. In addition, EFS generates a recovery key so that administrators have the ability to recover encrypted data if a user forgets a password or his or her employment is terminated. A 128-bit version of DESX is the current cryptography scheme that is implemented (a 40-bit scheme is implemented for international configurations). A flexible architecture, however, allows for the current DESX based implementation to change in future versions of the Windows 2000 operating system.

Encrypting File System Implementation Details

EFS encryption is implemented using Windows 2000-based security.

As a general rule, it is difficult to implement encryption in a file system because of the significant number of techniques used to increase file system performance and data availability. Data caching is an example of one of these techniques. Caching presents a challenge when encryption is present because it is possible to have both clear text data and ciphertext data simultaneously present in cache.

The Windows 2000 operating system decrypts all data and leaves it as clear text in the Windows 2000 protected cache. Thus, all file access attempts require the file to be decrypted. If decryption fails, attempts to read the file data will fail as well. Still, there are operations that need to access encrypted data in its ciphertext form. For example, backup and restore utilities and remote storage applications need access to encrypted data in an EFS environment. Therefore, Windows 2000 provides raw import/export APIs to allow the processing of data in ciphertext format. Several Windows 2000 file utilities have been enhanced to take advantage of these APIs.

EFS Architecture

The following describes the Encrypting File System architecture and components.

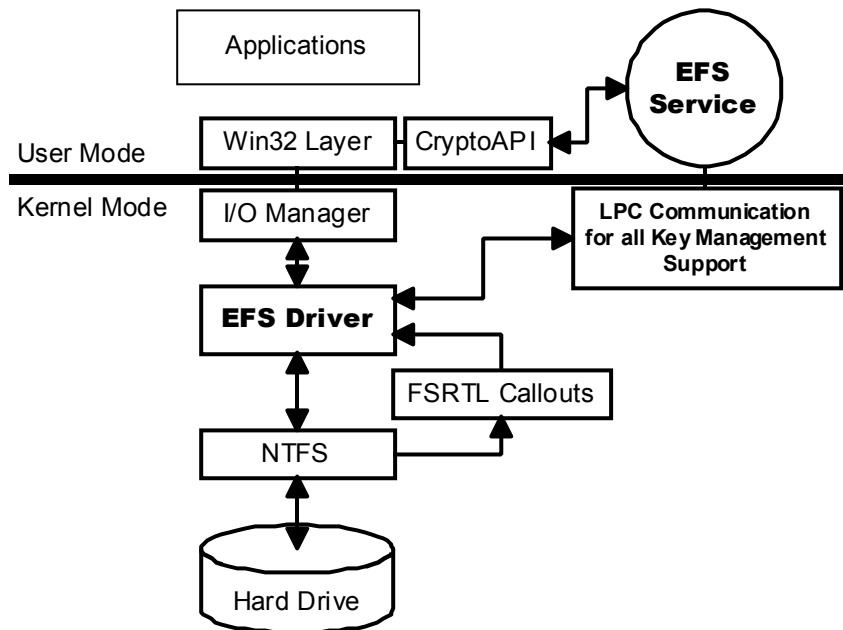


Figure 4. EFS architecture and components

EFS consists of the following components in the Windows 2000 operating system:

- **EFS driver**—The EFS driver works effectively as a slave driver to NTFS—it is not a filter file system. NTFS communicates with EFS service to request file encryption keys, data decryption fields (DDFs), data recovery fields (DRFs), and other key management services. It passes this information to the EFS file

system run-time library (FSRTL) to perform various file system operations (open, read, write, and append) transparently.

- **EFS FSRTL**—FSRTL is a module within the EFS driver that implements NTFS callouts to handle various file system operations, including reads, writes, and opens on encrypted files and directories, as well as operations to encrypt, decrypt, and recover file data when it is written to or read from disk. Even though the EFS driver and FSRTL are implemented as a single component, they never communicate directly. They use the NTFS file control callout mechanism to pass messages to each other. This ensures that NTFS participates in all file operations. The operations implemented using the file control mechanisms include writing the EFS attribute data (DDF and DRF) as file attributes and communicating the file encryption key (FEK) computed in the EFS service to FSRTL such that it can be set up in the open file context. This file context is then used for transparent encryption and decryption on writes and reads of file data from disk.
- **EFS service**—The EFS service is part of the security subsystem. It uses the existing LPC communication port between the Local Security Authority (LSA) and the kernel-mode security reference monitor to communicate with the EFS driver. In user mode it interfaces with CryptoAPI to provide file encryption keys and generate DDFs and DRFs. The EFS service also provides support for Win32 APIs, which are programming interfaces for encryption, decryption, recovery, importing, and exporting.
- **Win32 APIs**—These provide programming interfaces for encrypting plaintext files, decrypting or recovering ciphertext files, and importing and exporting encrypted files (without decrypting them first). These APIs are supported in a standard system dynamic-link library (DLL), advapi32.dll.

For detailed information regard EFS, see the white paper, "[Encrypting File System for Windows 2000](#)" located with other Windows 2000 white papers on the Windows 2000 Server Web site.

Programming Awareness with EFS

EFS is implemented in the Windows 2000 operating system as an installable file system filter driver that interoperates with NTFS. Windows 2000 encryption can be replaced with third-party filters (see IFS Kit for details). The new file system attribute `FILE_ATTRIBUTE_ENCRYPTED` (0x00000040) can be set on a file or a directory.

Access to raw encrypted data (as described above) is allowed using the **OpenRaw** and **WriteRaw** APIs (winbase.h). **OpenRaw** and **WriteRaw** allow the file to be opened and written in its encrypted state. Security is maintained as the data streams are encrypted, yet the file can be opened, which provides a solution to the challenges facing applications that need to perform non-intrusive operations, such as backup/restore.

NTFS Sparse File Support

A *sparse* file is a file with an attribute that causes the I/O subsystem to interpret the

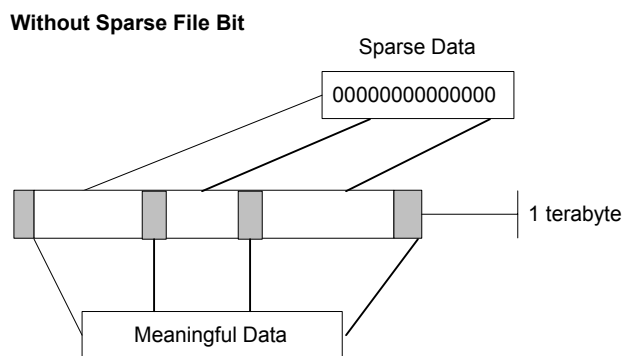
file's data based on allocated ranges. All meaningful or non-zero data is allocated, whereas all non-meaningful data (large strings of data composed of zeros) is simply not allocated. When a sparse file is read, allocated data is returned as stored and non-allocated data is returned, by default, as zeros in accordance with the C2 security requirement specification. Without sparse file support, the file system would actually have to allocate space for all zero bit data, which would be inefficient.

What Challenges Do Sparse Files Present?

NTFS includes full sparse file support for both compressed and uncompressed files. Disk allocation is only required for specified ranges. NTFS handles read operations on sparse files by returning allocated data and sparse data defined by file map ranges. It is possible to read a sparse file as allocated data and range data without having to retrieve the entire data set. This is desirable for applications that need to efficiently handle sparse files in their operations. By default, NTFS returns the entire data set.

Data streams with an NTFS sparse attribute set have two allocation definitions. The first is the virtual **AllocatedLength**, which is rounded up to a cluster boundary greater than or equal to the size of the stream. The second is **TotalAllocatedLength**, which represents the actual disk clusters allocated to the stream and is always less than or equal to the **AllocatedLength**.

An example of sparse file use is a scientific application that may require 1 terabyte of storage for data used in a matrix. Actual meaningful data in the matrix may only account for 1 megabyte. With the sparse file attribute set, the file system can de-allocate from anywhere in the file and yield the zero data to calling applications by range, instead of storing and returning the actual data. The result is that file access requests are satisfied with the correct bits and disk space is managed efficiently. File system APIs allow for the file to be copied or backed up as actual bits and sparse stream ranges. The net result is efficiency in file system storage and access.



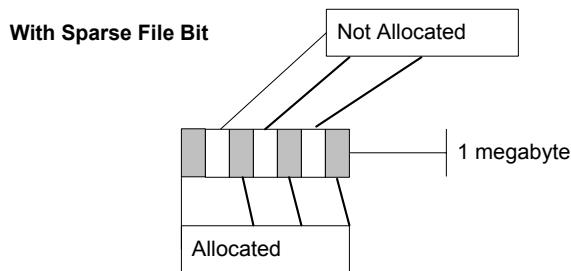


Figure 5. Sparse file support

Tips When Programming with Sparse Files

Keep the following in mind when programming with sparse files:

- Performance of file management applications benefit if sparse files are processed using range data and not accessed as normal operating system files. Normal file access returns the entire data contents of a file, which results in longer transfer times and higher storage space requirements.
- NTFS uses the file attribute bit `FILE_ATTRIBUTE_SPARSE_FILE` to indicate that a stream is potentially sparse. This attribute is returned with file attribute flags on both file and directory enumeration. A directory with this bit set merely indicates the possibility of a sparse file within the sub-tree.
- The Win32 APIs **BackupRead** and **BackupWrite** have been updated to deal with sparse files.
- The sparse file flag can be removed from a file without loss of data.
- **NtQueryVolumeInformationFile** returns a sparse file supported bit, `FILE_SUPPORTS_SPARSE_FILES`, when querying an NTFS volume.
- The FSCTL code for setting a sparse file bit on a stream is `FSCTL_SET_SPARSE`.
- Since quotas are a function of EOF and **AllocateLength**, sparse files are subject to quota restrictions: the entire file size, including zero-filled ranges, counts towards the quota.
- It is possible to deallocate a range of data in a stream with the FsCtrl `FSCTL_SET_ZERO_DATA`.
- NTFS will support a new FSCTL call, **QueryAllocatedRanges**, which returns the allocated ranges of a stream. Thus, storage applications will want to combine the **QueryAllocated** call with appropriate read calls to avoid wasted resources involved with processing and storing large streams of zeros. This FSCTL works correctly on both sparse and non-sparse files.
- Use the **GetCompressedFileSize** function to obtain the size of a sparse file. If the file is sparse, its size will not include the size of the sparse streams. Use the **GetFileSize** function to determine the allocated size of a file. The same API is used for obtaining the size of a compressed file.

Disk Quotas

Disk quotas provide system administrators with a powerful tool for managing storage growth in distributed environments. Disk quotas are implemented in NTFS on a per volume basis. This is accomplished by reporting available disk space based on user context.

Free Disk Space Awareness

Free disk space is reported by the quota management system as disk space available to a user context. Any application that wishes to know the actual free disk space outside of the quota mechanism will have to specifically call for it.

The following are some guidelines relating to free disk space calls:

- Use the function **GetDiskFreeSpaceEx**. For applications that also run on versions of Windows that don't support this function, include the logic that implements **GetDiskFreeSpace**.
- Disk space queries made in user context should be accomplished via the *FreeBytesAvailableToCaller* parameter in **GetDiskFreeSpaceEx**. This accurately reports disk space, taking quotas into consideration, to applications that run in user mode. Use the parameter *TotalNumberOfFreeBytes* for system-wide queries, such as those used by administrative applications.
- The introduction of Dfs, NTFS junctions, and volume mount points creates situations where logical directories do not have to correspond to the same physical volume. Thus, disk space should not be assumed based on space queries made in directories other than the current one.

Remote Storage

Remote Storage is a storage management feature based on NTFS reparse points. The purpose of Remote Storage is to automatically maintain a desired level of free space on disk volumes. Remote Storage does this by selectively removing data from primary storage after it has been safely copied to a high-capacity, low-cost secondary storage medium, such as magnetic tape. Remote Storage operations are transparent to users and conventional applications. Files are migrated from primary to secondary storage based on established rules. File attributes and other placeholder information remain on the primary storage so that migrated files can be viewed in directories, evaluated, and recalled as needed by users and applications.

Primary storage refers to NTFS disk volumes that are local to the server running Remote Storage, or mapped locally to that server by Dfs. Secondary storage refers to a pool of bulk storage devices, such as magnetic tape libraries.

Remote Storage is beneficial in managing the cost associated with large quantities of data that must be accessible. It is one of several features introduced in the Windows 2000 operating system dedicated to lowering the total cost of ownership associated with distributed computing.

What Does Remote Storage Do?

Remote Storage maintains an approximate level of desired free space on managed

NTFS volumes. When Remote Storage is set up, the administrator specifies the desired free space level for the volume, and some parameters to control which files can be migrated to secondary storage. For example, it might be desirable to migrate only files that have not been accessed in at least three days, and to exclude executable (.exe) files from consideration for migration.

When a volume is managed, Remote Storage scans the volume periodically for eligible data, copies the eligible files (in their entirety) to secondary storage, and attaches the Remote Storage reparse point to each file copied. The file is otherwise not modified at the time.

Periodically (more frequently than the above scans) Remote Storage checks to ensure that free space on the volume is at or above the desired free space threshold. If it is not, then Remote Storage creates free space by truncating the unnamed data stream from files that have already been copied to secondary storage. Before each file is truncated, Remote Storage uses the Change Journal to determine whether the data on secondary storage still represents the data on primary storage. If the primary data has changed, then this file is not truncated. Because the slower secondary storage devices are not involved in the truncation process, free space can be created very quickly when it is needed. A file which has had its primary data stream truncated is referred to as a *placeholder*.

When a file that has been truncated is needed, Remote Storage recalls the removed data from secondary storage and reconstructs the primary data stream. This operation is transparent to the application requesting the data, except that I/O operations are blocked until the requested data has been restored. This delay experienced during data retrieval is referred to as *latency*.

Storage Application Considerations

Since file recall involves latency, storage applications such as content indexing and anti-virus agents need to be aware of possible interactions when running Remote Storage. Storage applications that operate in the Windows 2000 remote storage environment should consider the following:

- Remote Storage depends on *last access time* attributes stored with each file. It is imperative that storage applications do not disturb these attributes. Be careful at the same time to avoid making spurious Change Journal entries (when the file accessed has not actually been modified). For example, when using **NTSetInformationFile** to prevent modification of last access time, be sure that *FileAttributes* are set to 0.
- The FILE_OPEN_READ_NO_RECALL flag can be used to satisfy a storage application's need to read data without recalling files from secondary storage to local media. When this flag is used, data is read from secondary storage directly into the application buffer, without affecting the status of the file on primary storage in any way.
- Because Remote Storage depends on the use of the reparse point, files managed by Remote Storage cannot have another reparse point attached.
- Volumes managed by Remote Storage can appear to contain far more data

than their actual physical capacity permits. To avoid allowing wildcard operations to fill managed volumes to capacity by recalling many files, most applications are subject to a runaway recall limit. This means that Remote Storage detects when an application requests more than a specified number of recalls in a fixed period of time. Applications that run with administrative privilege can be made exempt from the runaway recall limitation. It is imperative that such applications are aware of this and use `FILE_FLAG_OPEN_NO_RECALL` if their activity would otherwise cause runaway recalls.

- Remote Storage recalls cannot preempt other operations that tie up the secondary storage devices needed to access migrated data. Applications that share secondary storage devices with Remote Storage must be aware of potential deadlock or timeout situations that can result. Generally, such problems can be avoided by making two or more compatible drives available in the libraries used.
- Applications that take action after a file is modified should not confuse Remote Storage events (adding the reparse point after migration, or truncating the primary data stream) with substantive file modifications. For example, Indexing Service should not re-index a file after one of these events. Remote Storage events are tagged in the Change Journal with the source indicator `USN_SOURCE_DATA_MANAGEMENT`.

Preliminary Considerations for Remote Storage Applications

Although the Windows 2000 operating system includes a two-tiered remote storage application, ISVs can write multi-tiered remote storage solutions. However, ISVs should take some preliminary steps and consider some architectural issues before attempting to write a remote storage solution for Windows 2000. The following is a preliminary list:

- Obtain a registered reparse point tag value with the Microsoft Corporation.
- Take the Removable Storage architecture into consideration when designing a remote storage architecture.
- Take the Microsoft Management Console (MMC) into consideration when designing user interface (UI) components.

Removable Storage

The following section describes the Removable Storage architecture and components in some detail.

Removable Storage Architecture Overview

Removable Storage provides a single set of APIs that allow applications to catalog all removable media (except floppies and similar small capacity media), whether housed on shelves (offline) or in robotic libraries (online). In disguising the complexities of underlying robotic library systems and bulk media libraries, Removable Storage both lowers development costs for storage applications and provides consistency to customers who purchase these applications.

The Removable Storage architecture introduces several new storage components. The following figure, Figure 6, illustrates these components and their relationships to each other.

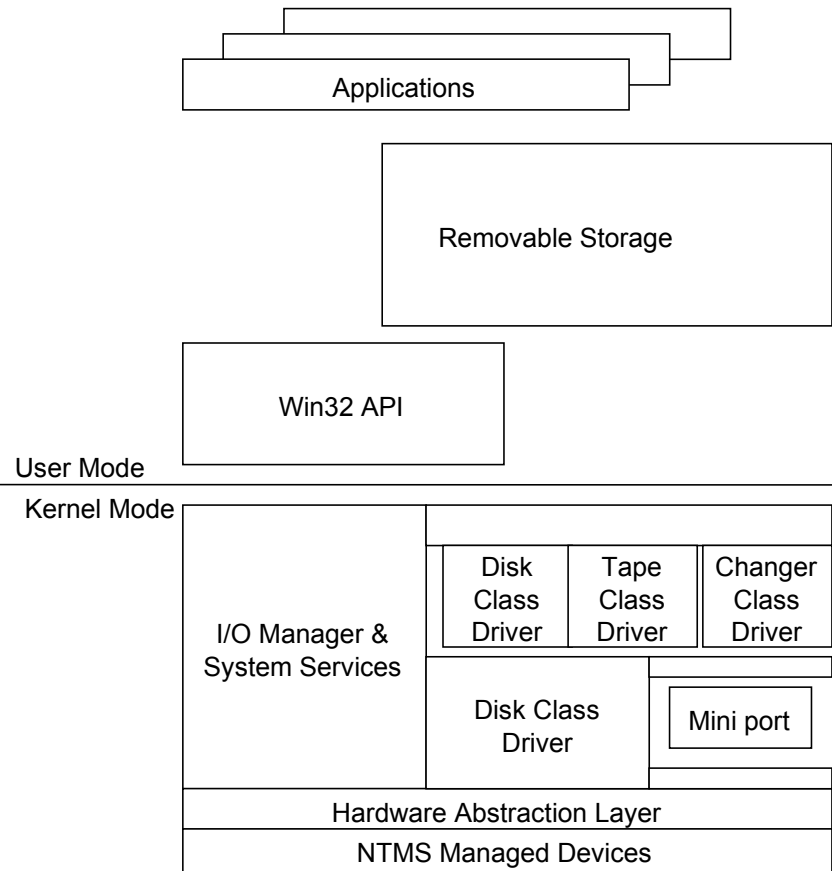


Figure 6. Removable Storage architecture

The components of Removable Storage include:

- **Applications**—Traditional data mover applications such as backup/recovery, hierarchical storage, and document management.
- **Win32 API**—A service that processes traditional I/O calls such as open, close, read, write, write seek, filemark, and so forth. Applications that use Removable Storage for removable media will still use these calls to perform I/O to the devices. The RSM API is an extension of the Win32 API to handle removable media operations such as cataloging, mounting and dismounting, etc.
- **Device Drivers**—Standard Windows 2000-based device drivers that represent the lowest level of interface with changer/library hardware. Device drivers are provided by hardware manufacturers.

Removable Storage API Overview

The Removable Storage API provides a layer of abstraction that simplifies the handling of underlying media hardware and drivers.

This discussion is intended to provide a high-level definition and explanation of the Removable Storage API. For API details, such as parameters, structures, and returns, refer to the Platform SDK.

The major classifications of the Removable Storage API functions are:

- Session Management
- Media Services
- Library Control
- Cleaner Management
- On-Media Management
- Object Management
- Operator Request
- Database Notification

Note: The Removable Storage API uses the acronym NTMS to refer to Windows NT Media Services. This feature has been renamed Removable Storage. To minimize the impact on existing code that references these APIs, the NTMS acronym will not be changed to reflect the current marketing name for the feature.

Session Management Functions

Session Management refers to the opening and closing of Removable Storage sessions. Sessions are required to access the majority of functionality in the Removable Storage API. Sessions can be opened and closed using **OpenNTMSSession** and **CloseNTMSSession** functions.

Opening a Removable Storage session returns a handle that is passed to other Removable Storage functions. When an application opens a session, a connection is established between that application and Removable Storage. Sessions must be closed using **CloseNTMSSession**. Sessions are thread safe; however, they should not be passed among processes.

Media Services Functions

Media services functions are used to mount/dismount media, allocate/deallocate media, and manage pools in an open session with a server running Removable Storage. To prepare for I/O, a sequence of calls would be:

1. Open a session.
2. Allocate media.
3. Mount media
4. Invoke I/O by using a Win32 API.
5. Deallocate media.
6. Dismount media.
7. Close the Removable Storage session.

The following is a list of media services functions:

- **AllocateNtmsMedia**
- **CreateNtmsMediaPool**
- **DeallocateNtmsMedia**

-
- **DecommissionNtmsMedia**
 - **DeleteNtmsMedia**
 - **DeleteNtmsMediaPool**
 - **DismountNtmsMedia**
 - **MountNtmsMedia**
 - **MoveToNtmsMediaPool**
 - **SetNtmsMediaComplete**
 - **SwapNtmsMedia**
 - **WaitForNtmsMount**

Media is mounted in a drive with **MountNtmsMedia** and dismounted with **DismountNtmsMedia**. **MountNtmsMedia** requests that a specific piece of media be mounted for an open Removable Storage session. A timeout is specified in milliseconds using the variable `ERROR_IO_PENDING`. If the mount fails, an error is returned. If the mount does not fail, but is not completed within the timeout period (specified by the value of `ERROR_IO_PENDING`), application code is required to issue a **WaitForNTSMount** call. Alternatively, the code can issue a **DismountNtmsMedia** call to cancel the request.

For media to be used, it must first be allocated using **AllocateNtmsMedia**. Media is allocated from a media pool. If there is no media available in that pool, then media can be allocated from a free pool, depending upon media policies and rules. The variable *DwTimeout* specifies both the time to search for available media and the time for an operator to insert media upon Removable Storage media insert request. Allocation can be cancelled. Once media is allocated, standard Win32 I/O APIs can be invoked to read and write data streams to media.

After media usage is complete for a given transaction, it can be deallocated using the function **DeallocateNtmsMedia**. An application does this only after all data on that media is no longer needed. Deallocated media is either marked available or decommissioned using **DecommissionNtmsMedia**.

Deallocated media can be in one of several states. These states are:

- **Available**—Available media can be allocated and used for I/O.
- **Decommissioned**—Decommissioned media is recognized by the system, but does not contain data and is never used by the system again.
- **Deleted**—Deleted media has been removed from the Removable Storage database. To be deleted, the media must be processed using the media service function **DeleteNtmsMedia**.

Library Control Functions

Library control functions are used to manage library systems. Removable Storage library control functions include:

- **AccessNtmsLibraryDoor**
- **AllocateNtmsCleanerSlot**
- **CancelNtmsLibraryRequest**
- **CleanNtmsDrive**

-
- **DeallocateNtmsCleanerSlot**
 - **DismountNtmsDrive**
 - **EjectNtmsMedia**
 - **InjectNtmsMedia**
 - **InventoryNtmsLibrary**
 - **UpdateNtmsOmidInfo**

Object Management Functions

Removable Storage object management functions provide management hooks for each object specified in Removable Storage. These functions allow the calling application to enable and disable Removable Storage objects, and to get and set object security and attributes.

The following is a list of Removable Storage object management functions:

- **DisableNtmsObject**
- **EnableNtmsObject**
- **EnumerateNtmsObject**
- **GetNtmsObjectAttribute**
- **GetNtmsObjectInformation**
- **GetNtmsObjectSecurity**
- **SetNtmsObjectAttribute**
- **SetNtmsObjectInformation**
- **SetNtmsObjectSecurity**

Operator Request Functions

Operator request functions allow management of operator intervention with the library unit. Operator request functions include:

- **CancelNtmsOperatorRequest**
- **SubmitNtmsOperatorRequest**
- **WaitForNtmsOperatorRequest**

Database Notification Functions

Removable Storage has a database that catalogs media and handles configurations and events. The database notification functions provide a way for an application to receive notification when events (state change of an object) occur. The Removable Storage database notification functions include:

- **CloseNtmsNotification**
- **OpenNtmsNotification**
- **WaitNtmsNotification**

Direct access to the Removable Storage database (beyond these functions) provides no value to applications developers and is not supported.

Media Pool Functions

Media pools have several functions in the management of a media server. Media pools control the selection of media and media type. Media pools allow media to be shared across applications and they allow such sharing to be tracked.

There are two classes of media pools: *system* and *application*. The system media pools are *free*, *import*, and *unrecognized*. The free pool holds media that contains no useful data and is freely available to any application. The import and unrecognized pools are holding places for media newly added to the system.

If Removable Storage recognizes the format of a new piece of media, it adds it to the import pool. If Removable Storage does not recognize the format, the media is added to the unrecognized pool. Applications then move media from these pools into either the free pool or to the application pools. Application pools are pools created by applications to hold media for their own use. For example, Backup, a backup utility that is included with Windows 2000 Server, creates its own application pool for backup media. Figure 7 illustrates the classes of media pools.

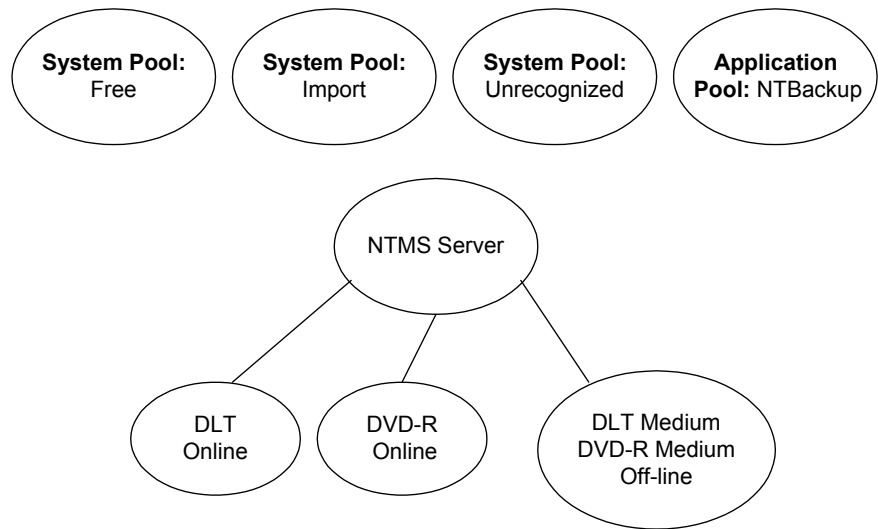


Figure 7. Media Pools

The Removable Storage media calls provide the functionality required to manage media pools. Both application code and media management interact with media pools.

Application media pools are created with the **CreateNtmsMediaPool** function. Note that because Removable Storage provides free, unrecognized, and import media pools by default, these pools cannot be created with **CreateNtmsMediaPool** (they already exist). Media pools are managed in a hierarchy separated by the backslash (\) character. The default pools created by Removable Storage are located at the top of the hierarchy.

Application pools have the application as the major node, the pool as the secondary node, and the media and policy being the leaf or end nodes. Thus Application 1 could have two pools: \App1\Pool1 and \App1\Pool2. This is very similar to file system hierarchies.

MoveToNtmsMediaPool is used to move media from its current media pool to

another media pool. Both media pools must be of the same media type, and security is a factor.

Media pools can be deleted with **DeleteNtmsMediaPool**. The media pool must be empty, and the default media pools created by Removable Storage (free, unrecognized, and import) are not affected by this function.

In summary, Removable Storage media functions provide interfaces required to control and manage media in a Windows 2000-based server running Removable Storage. The concept of media pools is essential to media applications and media management programs.

File System Replication

The Windows 2000 operating system provides file system replication that has two intended uses. First, there is a collection of system data, not stored in the Windows 2000 directory information tree (DIT), which must be replicated in an Active Directory environment. These files are stored in a logical, shared directory called SYSVOL that is replicated amongst other domain controllers. Second, services such as the distributed file system (Dfs) can provide added value by replicating content to duplicate share points in a storage hierarchy. For example, some companies keep read-only shares of site-licensed software on installation points for employees to install when it is needed. These binaries can be replicated to Dfs share points in multiple sites so that users can always install from local and up-to-date binaries.

System File Protection

When shared system files are overwritten by application installations, unpredictable performance results occur, ranging from application errors to operating system failure. The types of files most commonly affected when system files are overwritten are dynamic link libraries (DLLs) and executable files (.exe).

In the Windows 2000 operating system, a new feature called System File Protection (SFP) prevents the replacement of certain monitored system files. By preventing the replacement of essential system files, file version mismatches can be avoided.

How does System File Protection Work?

System File Protection provides protection for system files by using a background mechanism that runs inside WINLOGON.EXE on a Windows 2000-based system. At the end of GUI-mode setup, SFP runs a scan of all protected files to ensure they have not been modified by applications installed by unattended installation. After successful startup, the System File Protection service performs a check of all catalog (.cat) files used to track correct file versions. If any catalog files are missing or corrupted, System File Protection renames the affected catalog file and retrieves a cached version of that file from the dllcache directory. If a cached copy of the catalog file is not available in dllcache, SFP requests the appropriate media (i.e., Windows 2000 Service Pack, Windows 2000 Hotfix, etc.) to retrieve a clean copy of the catalog file.

System File Protection is triggered when it receives a directory change notification on a file in a protected directory. Once this notification is received, SFP will determine which file was changed. If the file is protected, SFP will look up the file signature in a catalog file to determine if the new file is the correct Microsoft version. If it is not, then the system either replaces the file from a cache directory or asks for the original installation media.

The only installation mechanisms that will not trigger the System File Protection functionality are:

- Windows 2000 Service Pack installation (UPDATE.EXE).
- Hotfix distributions installed using HOTFIX.EXE.
- Operating system upgrade (WINNT32.EXE).

Replacement of system files that are on the protected list by any other mechanism will result in the System File Protection mechanism of the Windows 2000 operating system replacing the new system file with a copy of the protected version that is stored in the dllcache directory.

Special Cases for Defragmentation

Defragmentation programs should be very careful not to defragment the hibernation or memory dump files. The Windows 2000 kernel determines the sectors that these files occupy at a very early phase in the boot process, and if the file is defragmented or otherwise moved on-disk, hibernation or the dump of system memory to a log file will write their data to the locations that these files previously occupied. If these files are defragmented or moved, this should be done at boot time, and the system immediately rebooted after defragmenting these files.

WINDOWS 2000 FEATURES THAT AFFECT STORAGE

The Microsoft Management Console

The Microsoft Management Console (MMC) architecture is an administrative tool for managing Windows 2000 components and services in the enterprise environment. The MMC hosts management applications called snap-ins that are used to perform management tasks. Thus, ISVs should plan to write an MMC snap-in as part of the administrative UI for their storage solution.

Unattended Installation or Automated Installation

Because the Windows NT operating system is quickly becoming the platform of choice for deploying information technology (IT) solutions, the actual number of servers being deployed is increasing dramatically. Many customers deploy hundreds to thousands of servers at a time as part of a solution rollout. In such rollouts, unattended or automated installations become crucial. Thus, storage solutions should either provide mechanisms for the unattended installation of products or should ensure the compatibility of their products with popular scripting and installation tools in the marketplace. Communicating a preferred rollout strategy to large customers helps to reduce support incidents and increases customer satisfaction.

Indexing Service

Indexing Service is part of the base operating system in Windows 2000. (Content Indexing Server is the feature included in Internet Information Server in the Windows NT 4 operating system.) The Windows 2000 Server (and Professional) Indexing Service indexes file system objects and intranet and Internet Web sites across volumes and machines. Indexing Service consists of two types of data: the actual index server binary components and the catalog or index data structures.

- **The index server**—The server itself can exist on any computer running Windows 2000. It consists of binary files and registry entries similar to many other Windows 2000 services.
- **The catalog files**—These files for Indexing Service can be on any volume local to the indexing service. The way that Indexing Service builds its catalogs is dynamic in nature, therefore, catalog files cannot simply be archived and later restored using standard file I/O techniques

Indexing Service is tightly integrated with NTFS in the Windows 2000 operating system and makes use of many new NTFS features including Native Property Sets. The index includes all words in the content of each document and all properties. Many property values are set automatically by the application that creates the document. In addition, users can create custom properties for any file on an NTFS volume.

Indexing Service uses the NTFS Change Journal to detect file additions, deletions, and modifications, even when the service is not running.

Indexing Service provides interfaces for management and content indexing

purposes. For more information, see the [Microsoft Platform SDK](#).

Installable File System Kit

Both file systems and file systems filters are installable in computers running Windows 2000. Microsoft now publishes an [IFS Kit](#) that provides the information required to write file systems and file system filter drivers in the Windows 2000 environment.

Cluster Service

Cluster service is the essential software component that allows multiple computers running Windows 2000 Server to work together to provide services that enhance both scalability and high availability. Cluster service makes NTFS partitions highly available, and can fail over physical disks, and mount NTFS partitions on any node in a cluster, thus protecting them against computer failures. Many applications can be cluster-enabled with little effort. Please refer to the Microsoft Platform SDK for a detailed description of the Windows Clustering architecture and how to add cluster awareness to an application, if appropriate.

Consider the following when using Cluster service:

- Is a custom resource DLL for the application appropriate?
- Clustering forces checkpoints to the application itself. ISVs will want to examine their checkpoint and recovery mechanisms, both from a server and a client perspective. This insures that data persistence issues and client connectivity are included in their application support.

Note: Some storage features are not supported by the Cluster service: shared, clustered disks cannot be dynamic, all partitions must be NTFS, and system disks on each node should be NTFS. Also note that reparse points, volume mount points, encrypted volumes, and logical volumes are not supported on shared disks.

Resource DLLs

Clustered computers running Windows 2000 Server use a Cluster service (among other components) that considers a resource to be any physical or logical component that can be managed. Thus a resource can be a physical item such as physical disk or a logical item such as an IP address, network name or software application. If software applications need to be cluster-aware and need to take full advantage of Windows 2000 clustering capabilities, they must provide a resource DLL.

A resource DLL provides entry points to the Resource Monitor. The Resource Monitor can call entry point functions to check the status of a resource, bring a resource online, or take a resource offline. Such actions are necessary to provide the health detection and failure responses that Cluster service requires. Details on Windows clustering technologies are provided on the [Exploring Windows Clustering Technologies Web site](#), as well as in the [Microsoft Platform SDK](#).

Recovery Logic

The Cluster service in the Windows 2000 operating system uses a shared nothing cluster model. Each node in a server cluster has sole ownership of its own memory, storage, and operating system. It also exclusively owns any resources that are hosted on this node. In the event that one node in a cluster fails, another node will take ownership of any resources that were online on this node. Any application that can store persistent data on NTFS can also become highly available if it can be restarted on any node.

In order for storage applications to be highly available, they must be cluster aware. This means that storage applications must:

- Determine if they run on a cluster or a standalone server.
- Know how to discover and support cluster resources. For example, a backup tool must know how to distinguish between a clustered disk and a non-clustered disk.
- Be able to persist application data in a configurable location (this should be a cluster disk).
- Support graceful shut down and restart, and should recover the state from the persistent data stored on a cluster disk. For example, a network backup used to backup data for a cluster-aware application stored on a clustered disk should frequently check its state and, in case of a node failure, the disk will fail over to another node. The application will be restarted on this node. Backup should fail over together with the disk, and restart from the point where it was at the last checkpoint.
- Be aware that cluster resources, including physical devices, can be stopped and started at any time.
- Assure that disk and file system filter drivers should not interfere with the cluster clusdisk driver.

Microsoft Distributed File System

The Distributed File System (Dfs) for the Windows 2000 operating system is a network server component that presents a logical view of distributed physical storage. Dfs allows distributed file systems to be united into a single namespace. Its usage includes higher data availability, load balancing, name transparency, and flexible volume administration.

The Dfs server component presents distributed volumes in a logical manner by mapping physical UNC names to logical paths. Thus, using \\MS_Server\Public\Users\JoeUser as a logical path could have the following physical topology underneath.

DFS Logical Path	Physical Location	Explanation
\\MS_Server\Public	\\MS_Server\Public	Root Share
\\MS_Server\Public\Users	MS_Users1\Employees	Junction to employee directories.
\\MS_Server\Public\Users\JoeUser	\\JoeUser\JoeUser_Public	Junction to Joe User's computer.

Logical-to-physical mappings are stored on the server in the Partition Knowledge Table (PKT). The PKT is a sorted look-up table of 300 bytes per entry. The PKT is stored in the Active Directory metadata in Windows 2000, and in the registry in Windows NT 4.0. Dfs clients cache PKT information for each junction that is crossed.

Dfs presents a number of challenges for some storage applications. Any storage application that traverses directories as a part of its operation, such as backup/restore, should be concerned with Dfs. Consider the following points when designing a storage application that is Dfs aware:

- Storage applications should be aware of Dfs volumes. You can determine whether they are by checking a Server Message Block (SMB) bit.
- If a Dfs volume is encountered, storage applications should read the PKT to understand the physical topology of the Dfs implementation. Administrators or backup operators should be provided with intelligent options for dealing with Dfs volumes and storage applications.
- Junctions represent the point where a physical machine boundary is crossed. Junctions greatly affect the scope of a storage application. When a junction is encountered, administrators should be provided with intelligent options.
- When a load-balanced volume is encountered in Dfs, the server returns all choices to the client so that the client can choose the correct volume. A storage application can handle this in a variety of ways. The storage application can choose to pursue none, one, or all of the returned options. To achieve this functionality, use the PKT to determine the physical paths. (Replication will most likely apply here as well. Replication is addressed earlier in this paper.)
- Restoration functions in storage applications require that both the physical topology and the logical naming in the PKT be identical. If not, logical choices must be presented to an administrator so that restoration can be made to new locations or temporary holding areas.
- If the physical server representing the junction is down during file restoration across a junction, standard restore behavior of a backup/restore application is to create a directory. However, this happens on the wrong side of the junction. Backup/restore applications must be aware of this situation by manually processing the PKT. Then, intelligent options must be provided to the administrator or backup operator. These options could include:
 - Queuing the restore on the assumption the server is temporarily down.
 - Moving the contents to another location and automatically reconfiguring the logical namespace.
 - Canceling the operation.
- Storage applications are not guaranteed security rights on remote volumes (This is especially true if different network operating systems are involved).
- When enumerating the available storage space on a volume, **GetDiskFreeSpace** will return the space available at the root of the Dfs volume. It does *not* return the total of the free space of all the sub trees.

GetDiskFreeSpaceEX is a new function that returns the space available at a specified point within a namespace, and permits you to obtain the free space on the other side of a junction. This result is dependent on alternate volumes being mounted and on alternates having identical storage devices.

- Dfs roots are always hosted on Windows NT Server 4.0 or later. The root can be either FAT or NTFS format. When crossing junctions, it is possible to change from one file format to another. Also, when crossing junctions, it is possible to change from the Windows 2000 operating system to an alternate network operating system that is formatted uniquely and has a different security model. Another effect of junctions to different physical servers is that the network address between logical subdirectories can change. (This is transparent to end users and applications).

SUMMARY

The Windows 2000 operating system offers several core technologies that present numerous opportunities for ISVs to provide value-added solutions for the management and recovery of enterprise data. The architectural changes introduced in the Windows 2000 operating system present areas that developers and architects should consider when creating storage solutions for the Windows 2000 operating system. Storage applications designed for Windows NT version 4.0 or earlier should be updated to include the logic necessary to be effective when running in the Windows 2000 environment.

**FOR MORE
INFORMATION**

For the latest information on Windows 2000 Server, check out the [Windows 2000 Server Web site](#) or the Windows NT Server Forum on the Microsoft Network (GO WORD: MSNTS).

For the latest programming information, refer to the [Microsoft Developer Network](#) (MSDN), which includes the most recent version of the Microsoft Platform SDK.