Inside Windows NT Disk Defragmenting

Mark Russinovich

(Preprinted From WindowsItPro Magazine)

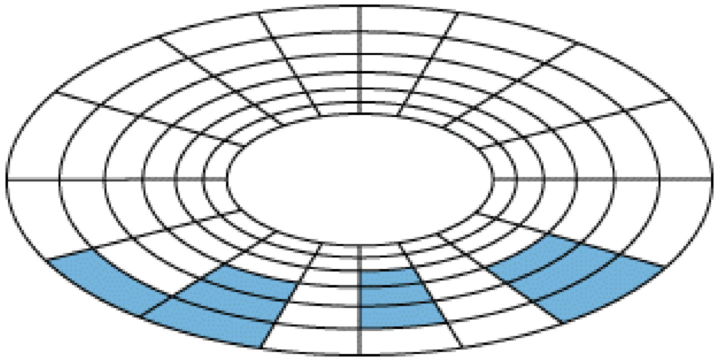
Welcome to the first installment of my regular column that explores the "guts" of Windows NT. I plan to cover topics ranging from high-level issues such as the architecture of NT to low-level issues such as details of algorithms NT employs. If you have a particular NT internals topic you'd like to see this column demystify, please drop me an email.

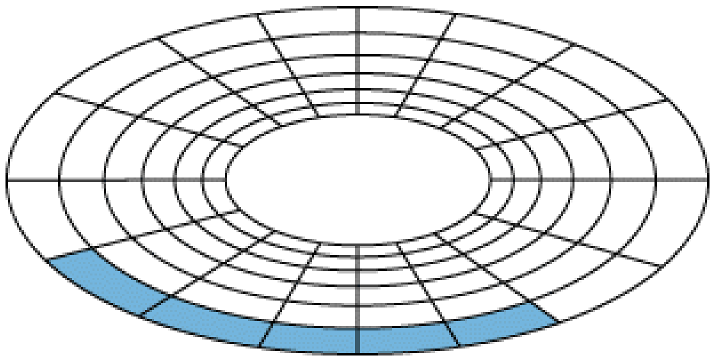
This month's topic is NT's disk defragmentation support. Disk defragmentation products are hot sellers, and many people want to know how these products work. After defining disk defragmentation, I'll provide a synopsis of the history of NT defragmentation products, and describe the support Microsoft added to NT 4.0's file systems specifically for use by defragmentation tools.

Disk Defragmentation

NT's file systems allocate the disk sectors of logical drives (e.g., C, D) in units known as clusters. When you create a file on a drive, the file system assigns clusters of storage for the file's data. As time passes and you expand and shrink the file and create and delete other files, both the free clusters on the drive and the clusters assigned to the files become fragmented (i.e., non-contiguous).

For example, suppose you create a Word document and the file system assigns six clusters to hold the document's contents. Later, you add a few pages to the document, and the file grows by six more clusters. If another file or directory uses the clusters that immediately follow the document's original clusters, the file system assigns the document's new file clusters from somewhere else on the drive--your file now consists of at least two fragments. Figure 1A, shows a simplified diagram of a fragmented file; Figure 1B depicts a defragmented file.





Fragmentation can negatively affect performance. When the disk head reads contiguous data, the level of throughput is high. When the system reads a fragmented disk file, the disk head must move from fragment to fragment. On a disk with many fragmented files, moving the disk head across fragments can consume significant time.

Disk defragmenters are products that aim to optimize system performance by reorganizing a drive's clusters to make file clusters contiguous, and in some cases, to make the free clusters contiguous. Defragmenters use heuristic algorithms that implement a fragment-shifting strategy to defragment the drive's files as quickly as possible.

The History of Disk Defragmenting on NT

In April 1995, Executive Software released the first defragmenter for NT: Diskeeper for NT 3.5 with Service Pack 1 (SP1) or Service Pack 2 (SP2). The company announced an update for NT 3.51 in July of the same year.

Because Microsoft did not design NT 3.5x's native file systems, FAT and NTFS, with disk defragmenting in mind, FAT and NTFS have no support for moving clusters around a disk. Executive Software purchased an NT source license from Microsoft to modify NT and the FAT and NTFS drivers to support disk defragmenting. As a result, Executive Software shipped its custom version of NT with Diskeeper for NT 3.51.

But as Diskeeper users upgraded NT 3.51 with SPs, they experienced problems. Installing an SP overwrote some of Diskeeper's files and left others alone, causing incompatibility problems for upgraded systems--and technical support problems for Microsoft and Executive Software.

As development of NT 4.0 began in 1995, Microsoft invited a Diskeeper developer to Redmond, Washington, to participate in the design and implementation of NT 4.0's defragmentation support. Basing Diskeeper on NT's built-in support let Executive Software avoid shipping custom versions of NT and meant Microsoft's technical support didn't have to troubleshoot non-standard versions of NT. One month before NT 4.0's public release, Executive Software offered a trial version of Diskeeper 2.0--the version for NT 4.0--on its Web site. Symantec has since entered the NT disk defragmenting market with its Norton Utilities Speed Disk, which also uses the NT 4.0 defragmentation support.

NT 4.0 Support for Disk Defragmenting

The defragmentation support introduced in NT 4.0's file systems consists of five commands: GetVolumeBitmap, GetRetrievalPointers, and MoveFile (common to both FAT and NTFS), and GetVolumeData and ReadMFTRecord (specific to only NTFS). Microsoft doesn't document these commands or officially acknowledge that they exist.

TABLE 1: Example File Mapping Array Returned by GetRetrievalPointers

File Cluster	Drive Cluster
0	1200
4	1000
7	1300

12	

TABLE 2: Example Compressed File Mapping Array Returned by GetRetrievalPointers

File Cluster	Drive Cluster
0	1200
4	-1
16	

GetVolumeBitmap

The GetVolumeBitmap command obtains a map of the free and allocated clusters of a logical drive. NT file systems maintain an array of bits that track the drive's clusters. An "on" bit signals an allocated cluster; an "off" bit signals a free cluster.

GetRetrievalPointers

The GetRetrievalPointers command returns an array of mapping information about the clusters allocated to a file. Each fragment of the file corresponds to an array entry that consists of a logical cluster number within the file and a drive cluster number. A final array entry contains the file cluster number of the end of the file. For example, consider a file with three fragments: The first fragment starts at drive cluster 1200 and continues for 4 clusters, the second fragment starts at drive cluster 1000 and continues for 3 clusters, and the third fragment starts at drive cluster 1300 and continues for 5 clusters. Table 1 shows the mapping array that GetRetrievalPointers returns for the file.

Sometimes GetRetrievalPointers returns a mapping array that contains a drive cluster entry of -1, as shown in Table 2. This entry signals a compressed file. In Table 2's example, the compressed data starts at drive cluster 1200 and continues for 4 clusters. The final file cluster entry of 16 means that the uncompressed file will require 12 more clusters.

MoveFile

The MoveFile command is the heart of NT's defragmentation support. MoveFile requires a handle to both the file that contains segments to be moved and the file's drive. Additional parameters track the starting cluster (relative to the start of the file) of the segment to be moved, the target drive cluster number, and the length of the segment. If the target clusters are free, MoveFile relocates the original clusters in a way that prevents data loss in case the system crashes during the move.

GetVolumeData

NT includes the two NTFS-specific defragmenting commands because the NTFS driver uses clusters differently from the FAT driver. The GetVolumeData command obtains detailed information about an NTFS drive, including its cluster size (in bytes), the size of the drive (in clusters), and the amount of free space on the drive (in clusters).

Defragmenters use GetVolumeData to identify a reserved portion of the disk (known as the MFT-Zone) that NTFS uses for expanding the Master File Table (MFT). The MFT is NTFS's index to all the files and directories on a drive.

To optimize file lookups, NTFS tries to keep the MFT defragmented by not allocating clusters around the MFT to other files. GetVolumeBitmap reports free clusters in the MFT-Zone, but MoveFile will not relocate clusters to this area; defragmenters need to know the MFT-Zone's location to avoid it.

ReadMFTRecord

The other NTFS-specific command, ReadMFTRecord, obtains a record of information from the MFT that you can use to create a cluster map for a file. Alternatively, you can enumerate the files on the drive and use GetRetrievalPointers to obtain mapping information for each file. However, sequentially reading MFT records and interpreting the raw NTFS on-disk data structures can enhance performance.

How NT 4.0 Defragmenters Work

At press time, only Executive Software and Symantec offer defragmenters for NT 4.0. Both vendors provide downloadable versions of their products. The products perform the same basic operations, but they employ different defragmentation algorithms. Let's look at how the products work.

First, each product creates a map of the drive, which shows the file fragmentation to the user. Mapping a drive takes three steps: Get the map of free clusters on the drive with GetVolumeBitmap, enumerate all the files on the drive and obtain file cluster maps with GetRetrievalPointers or ReadMFTRecord, display the file mappings.

Some clusters in the bitmap appear to be in use but not allocated to a file. These clusters belong to system metadata files (i.e., files that store file system-related information), directories, or files accessed exclusively by a process other than the defragmenter. Both products designate these clusters as immovable in their GUIs. The products also identify the MFT-Zone if the drive is an NTFS volume.

Next, the products enter a defragmenting phase. Because the drive mapping information constantly changes as programs create, delete, grow, and shrink files, the products do not rely on the information displayed to the user. Instead, they again enumerate all the files on the drive, and perform the following steps for each file: Get the map of free clusters on the drive with GetVolumeBitmap; obtain a cluster map for the file in question using GetRetrievalPointers or ReadMFTRecord; move segments of the file with MoveFile in an attempt to defragment the file.

The logic behind the third step is different for each product, depending on whether the defragmenter tries to move files to defragment the drive's free space or make room for files in specific places. Defragmentation is an iterative process that can even be undone as other processes perform file operations, so the defragmenters often repeat the third step many times.

Because MoveFile requires a handle to the file to be moved, the defragmenters must open a file before moving it. Opening a file that another process has already opened for exclusive access is not possible. Neither product can move files such as the MFT, Registry files, and Paging files because the system opens these files for exclusive access.

NTFS Caveats

Some restrictions apply to the NTFS implementation of MoveFile because its cluster movement engine uses NTFS file compression code. NTFS file compression adds a twist to the way NTFS allocates clusters for files.

NTFS performs compression on 16-cluster segments of a file. If a 16-cluster segment of data compresses down to 5 clusters, for instance, NTFS stores the 5 clusters on disk and notes the remaining 11 clusters as virtual clusters. To read the compressed file, the system reads the 5-cluster compressed portion from the disk, allocates memory for the 11 virtual clusters, and fills those memory locations with 0s. The system passes this 16-cluster chunk to the decompression algorithm, which re-creates the original data.

On FAT volumes, MoveFile can move clusters individually. The NTFS MoveFile routine moves clusters in only 16-cluster blocks because NTFS file compression works with 16-cluster segments.
Furthermore, the NTFS MoveFile function does not work with clusters larger than 4KB (NTFS file compression buffers are 64KB in size: 64KB ÷ 16 = 4KB). On drives larger than 4GB, the FORMAT utility initializes NTFS partitions with cluster sizes greater than 4KB; consequently, large drives with FORMAT's default cluster sizes do not support defragmentation.

Finally, NTFS prevents deallocated clusters from being used again until NTFS checkpoints the drive's state. Once every few seconds, NTFS ensures that all its crash recovery data is safely on disk; only then can deallocated clusters be reused. This characteristic challenges defragmenters because they can't determine when they can reallocate free clusters without repeated calls to GetVolumeBitmap.

Inside Windows NT Disk Defragmenting

Note: The information presented here is the result of my own study. No source code was used.

Introduction

The first defragmentation support for Windows NT was introduced by Executive Software. Their Diskeeper product was initially released for Windows NT 3.51, and because the NTFS and FAT file systems for NT 3.51 have no native functions that provide for cluster movement, Executive Software was forced to purchase a source license for NT and to create and ship custom versions of NTFS and FAT, as well as NT itself, along with their defragmentation code. The divergence of Executive Software's version of NT from Microsoft's version resulted in severe compatibility problems for Diskeeper's users when NT 3.51 Service Pack releases were produced. It was also undoubtedly a support headache for Microsoft, because unsuspecting users would many times not realize that problems they encountered were the result of Diskeeper, and not a Service Pack.

With the development of NT 4.0, Executive Software and Microsoft had a chance to introduce native support for disk defragmentation. According to Executive Software, they requested specific functionality in NTFS and FAT for cluster reallocation, which Microsoft added for them. Just before the release of NT 4.0's retail version, Executive Software made its Diskeeper 2.0 product Beta version available for download from its Web site.

In this page I'm going to document NT 4.0's defragmentation support interfaces and present a Win32 console program, along with its source code, that serves as a demonstration of them in action. The program, called Defrag, will let you manually move clusters around your FAT or NTFS file systems, and could easily be extended to actually become a full-blown defragmenter. You can learn more about NT defragmenting in my Windows NT Magazine NT Internals column, "Inside Windows NT Disk Defragmentation", which is available on-line.

The NT 4.0 Defragmentation Interface

NT 4.0's defragmentation support is based on file system control (FSCTL) commands, which are a form of device I/O control commands that are intended specifically for file systems. FSCTL's provide a file system-dependent interface that can extend the standard file system interface provided by NT's I/O Manager. Four FSCTL's were introduced for defragmentation and are listed below. The names I've assigned them are not necessarily the official Microsoft names, but match the names of the functions that are invoked when they are used.

```
FSCTL_GET_VOLUME_BITMAP
FSCTL_GET_RETRIEVAL_POINTERS
FSCTL_MOVE_FILE
FSCTL_READ_MFT_RECORD
```

The first three are implemented on both NTFS and FAT file systems, but the fourth, FSCTL_READ_MFT_RECORD, is only relevant to NTFS. FSCTL_READ_MFT_RECORD makes it possible to very efficiently determine what files are on a NTFS volume and where their clusters are located, but its use requires detailed knowledge of undocumented NTFS on-disk data structures, so I won't cover it. The remaining three FSCTL's are powerful enough that with them, a disk can be defragmented by a program that knows nothing about FAT or NTFS data structures, or that even cares which type of file system is being processed. The rest of this section will cover each in turn, but first I'll document a native Windows NT function, NtFsControlFile, that is used to send FSCTLs to a file system from a Win32 program. Throughout the descriptions of the APIs, the terms virtual cluster number (VCN or Vcn) and logical cluster number (LCN or Lcn) are used. VCNs are clusters within a particular file, and LCNs are clusters on the volume. Thus VCN 0 of a file may map to LCN 2394 of a volume, VCN 1 may map to LCN 104227, and so on.

NtFsControlFile

This function is exported by the Windows NT kernel-mode interface library called NTDLL.DLL. Many Microsoft programs use functions in NTDLL.DLL, including the various operating environment subsystems (Win32, POSIX, OS/2, and WOW), and other programs that ship as part of NT. Here is the definition for the command, Win32 SDK-style:

```
NTSTATUS NtFsControlFile(
    IN HANDLE FileHandle,
    IN HANDLE Event, // optional
    IN PIO_APC_ROUTINE UserApcRoutine, // optional
    IN PVOID UserApcContext, // optional
    OUT PIO_STATUS_BLOCK UserIoStatus,
    IN ULONG FsControlCode,
    IN PVOID InputBuffer, // optional
    IN ULONG InputBufferLength, // optional
    OUT PVOID OutputBuffer, // optional
    IN ULONG OutputBufferLength // optional
);
```

Parameters

FileHandle

This is the handle of the file or volume returned by CreateFile that the FSCTRL is directed at. Some FSCTL's are file-specfic, and others are volume specific.

Event

Handle to an event that can be synchronized on if the function returns STATUS_PENDING, indicating that it hasn't completed. Instead of passing in an event handle, callers of NtFsControlFile generally block on the file handle when I/O is pending, since it is also signaled when the operation completes.

UserApcRoutine

This points at a function that, if Event is NULL, will be executed when the operation has finished. Use of the function in this way is atypical.

UserIoStatus

This data structure is filled with the function's return status. It is where a caller should find the operation's final status in the case that the function originally returned STATUS_PENDING and the caller had wait for the it to complete.

InputBuffer

A pointer to the caller-allocated FSCTL input parameter, if it expects one.

InputBufferLength

The length in bytes of the variable or data structure passed as an input.

OutputBuffer

Points to buffer that will receive the FSCTL's output data, if it provides any.

OutputBufferLength

The length in bytes of the variable or data structure to receive output data.

Comments

This function is used by several Win32 functions, including DeviceIoControl. The easiest way to use it is to not pass an event handle or an Apc routine, but instead to wait for STATUS_PENDING operations to complete by blocking on the file handle. See defrag.c in the demonstration program for examples of its use.

The Volume Map

FSCTL_GET_VOLUME_BITMAP

Operation

This is generally the first FSCTL that is used by a defragmenter, because it is required to get a map of the clusters that are free and in use on a volume. Each bit in the bitmap returned in the output buffer represents one custer, where a 1 signifies an allocated cluster and a 0 signifies a free cluster.

FileHandle

The file handle passed to this call must represent a volume opened with the GENERIC_READ flag. A volume can be opened with a name of the form, "\\.\X:", where 'X' is the volume's drive letter.

InputBuffer

The input buffer must point to a ULONGLONG value that specifies the 8-cluster aligned starting cluster that the first bit in the returned bitmap buffer will correspond to. If the value is not 8-cluster aligned, the bitmap data returned will begin at the 8-cluster aligned value just below the one passed. InputBufferLength must be the size of a ULONGLONG in bytes, which is 8.

OutputBuffer

Points at a BITMAP_DESCRIPTOR data structure, which is defined as follows:

```
typedef struct {
    ULONGLONG StartLcn;
    ULONGLONG ClustersToEndOfVol;
    BYTE Map[1];
} BITMAP_DESCRIPTOR, *PBITMAP_DESCRIPTOR;
```

StartLcn is filled in with the cluster number corresponding to the first bit in the bitmap data. ClustersToEndOfVol represents the number of clusters on the volume minus the StartLcn cluster. Map is the cluster bitmap data. The length of the bitmap data is the difference between OutputBufferLength and 2 * sizeof(ULONGLONG).

Return

If there are errors related to the volume's support for this FSCTL or FileHandle representing a valid volume handle, an appropriate native NT error code is returned (as defined in the NT DDK file NTSTATUS.H). If the cluster specified in InputBuffer is out of range for the volume, the call will return STATUS_INVALID_PARAMETER. If there are no errors and there are no clusters beyond the last one described in the Map array, the FSCTL returns STATUS_SUCCESS. Otherwise STATUS_BUFFER_OVERFLOW is returned to notify the caller that further calls should be made to retrieve subsequent mappings.

File Maps

FSCTL_GET_RETRIEVAL_POINTERS

Operation

This function returns the cluster map for a specified file. The cluster map indicates where particular clusters belonging to the file reside on a volume.

FileHandle

This is a file handle returned by a CreateFile call that opened the target file.

InputBuffer

The ULONGLONG starting cluster within the file at which the mapping information returned will commence. InputBufferLength must be 8.

OutputBuffer

Points at a GET_RETRIEVAL_DESCRIPTOR data structure:

```
typedef struct {
    ULONG NumberOfPairs;
    ULONGLONG StartVcn;
    MAPPING_PAIR Pair[1];
} GET_RETRIEVAL_DESCRIPTOR, *PGET_RETRIEVAL_DESCRIPTOR;
```

The number of mapping pairs returns in the Pair array is placed in NumberOfPairs. StartVcn indicates the first cluster within the file that is mapped by the Pair array. The Pair array is a list of MAPPING_PAIR entries:

```
typedef struct {
    ULONGLONG Vcn;
    ULONGLONG Lcn;
} MAPPING_PAIR, *PMAPPING_PAIR;
```

Each entry is made up of a virtual cluster offset, and a logical cluster on the volume. The interpretation of the array is as follows: the first Lcn in the Pairs array corresponds to the StartVcn of the GET_RETRIEVAL_DESCRIPTOR data structure. The length of the file segment that starts at that cluster can be calculated by subtracting the GET_RETRIEVAL_DESCRIPTOR StartVcn from the Vcn of the first entry in the Pairs array. The second segment of the file starts at the Vcn of the second entry in the Pairs array, with a corresponding Lcn described by the Lcn of the first entry. Its length is the difference between the Vcns of the first and second entries. These relationships are shown more clearly in the figure below.

Pairs Array

On NTFS volumes, compressed files contain 0-filled clusters that have no correspondence to Lcns on the volume. These clusters are described with Lcns in the Pairs array equal to (ULONGLONG) -1.

The maximum number of entries that can be returned in the Pairs array is equal to (OutputBufferLength - 2 * sizeof(ULONGLONG))/ (2 * sizeof(ULONGLONG)).

Return

If there are errors related to the volume's support for this FSCTL or FileHandle representing a valid volume handle, an appropriate native NT error code is returned (as defined in the NT DDK file NTSTATUS.H). If the cluster specified in InputBuffer is out of range for the volume, the call will return STATUS_INVALID_PARAMETER. If there are no errors and there are no clusters beyond the last one described in the Map array, the FSCTL returns STATUS_SUCCESS. Otherwise STATUS_BUFFER_OVERFLOW is returned to notify the caller that further calls should be made to retrieve subsequent mappings.

Moving Clusters

FSCTL_MOVE_FILE

Operation

This is the core of the defragmentation support. It is used to move the clusters of a particular file to a currently unallocated position on the volume.

FileHandle

The file handle passed to this call must represent a volume opened with the GENERIC_READ flag. A volume can be opened with a name of the form, "\\.\X:", where 'X' is the volume's drive letter.

InputBuffer

A pointer to a MOVEFILE_DESCRIPTOR:

```
typedef struct {
    HANDLE FileHandle;
    ULONG Reserved;
    LARGE_INTEGER StartVcn;
    LARGE_INTEGER TargetLcn;
    ULONG NumVcns;
    ULONG Reserved1;
} MOVEFILE_DESCRIPTOR, *PMOVEFILE_DESCRIPTOR;
```

FileHandle is a handle to a file previously opened by CreateFile, and represents the file that is the subject of the move operation. StartVcn is the start of the segment within the file that will be moved. TargetVcn is the Lcn on the volume to which the files clusters will be moved, and NumVcns are the number of clusters making up the segment that will be moved.

NTFS Caveats: The NTFS implementation of this command uses some of the logic present in NTFS that supports the reallocation of data within compressed files. The NTFS compression algorithm divides compressed files into 16-cluster segments, so the MoveFile functionality on NTFS suffers a similar restriction. When the clusters of an uncompressed file are moved, StartVcn must be 16-cluster aligned, or MoveFile will adjust it to the next lowest 16-cluster boundary. Similarly, NumVcns must be a multiple of 16 or MoveFile will round it up to the next multiple of 16. If the clusters of a compressed file are being moved, the rules are little more complex: StartVcn can specify the beginning of any non 0-filled segment of the file that immediately follows a 0-filled segment, and NumVcns must specify a run of clusters within the file that precisely encompasses non 0-filled segments of the file. The same type of rounding described for movement of non-compressed files will be performed if the rules are not followed.

OutputBuffer

This function does not return any data, so this parameter should be NULL.

Return

If either the volume file handle or the file's handle are invalid an appropriate error code is returned. If StartVcn is out of range for the file or TargetLcn is out of range for the volume, STATUS_INVALID_PARAMETER is returned.

If the move is directed at a FAT volume, the only way to tell if the move was successful is to re-examine the file's mapping using FSCTL_GET_RETRIEVAL_POINTERS, since the call will always return STATUS_SUCCESS.

If the move is directed at an NTFS volume, the function will return STATUS_SUCCESS if it was successful. Otherwise it will return STATUS_ALREADY_COMMITTED to indicate that some range of the target clusters are already in use. In some cases, attempting to move clusters to an area that is marked as unallocated in an NTFS volume bitmap as free will result an unexpected STATUS_INVALID_PARAMETER. This is due to the fact that NTFS reserves ranges of clusters for the expansion of its own metadata files, which the cluster reallocation code will not attempt to use.

FSCTL_MOVE_FILE does not work on volumes with cluster sizes larger than 4KB. The error returned when moves are attempted on such volumes is STATUS_INVALID_DEVICE_REQUEST. This limitation, which is tied to its implementor's mistaken belief that FSCTL_MOVE_FILE must suffer the same limitations as NTFS compression, is relatively serious because FORMAT uses cluster sizes larger than 4KB on volumes larger than 4GB.

Another thing to be aware of when moving clusters on a NTFS volume is that clusters that have been freed by a MoveFile operation will not be available for reallocation themselves until the volume has been checkpointed and if the volume has not been checkpointed, the function will return STATUS_ALREADY_COMMITTED. NTFS checkpoints volumes every few seconds so the only remedy is to wait and try again.

Note that because a file handle for the file to be moved must be passed to FSCTL_MOVE_FILE, it is not possible to move the clusters of files that have been opened by another process for exclusive access. Since the system opens paging files, the Registry, and NTFS metadata files for exclusive access, they cannot be defragmented. Also, because of the way the FSCTL_MOVE_FILE routine is written, it is only possible to reallocate file data clusters, and not directories or other file metadata.