**Active Data Recovery Software**


# User Guide

Version Number 3.0

Active@ UNERASER for DOS v 3.0

END-USER LICENSE AGREEMENT



IMPORTANT-READ CAREFULLY

This End-User License Agreement (EULA) is a legal agreement between you (either an individual or a single entity) and Active Data Recovery Software for the personal use or business use of Active@ UNERASER for DOS (UNERASER).

By installing, copying, or otherwise using UNERASER you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use UNERASER.

ACTIVE DATA RECOVERY SOFTWARE REQUIRES THAT EACH PURCHASER USE THE FREE DEMO VERSION OF UNERASER (DEMO VERSION) BEFORE PAYING A LICENSE FEE FOR THE REGISTERED VERSION TO GET A FULL UNDERSTANDING OF THE CAPABILITIES AND THE EASE-OF-USE OF UNERASER. DEALERS MUST EITHER SUPPLY A COPY OF THE DEMO VERSION OR RECOMMEND THAT THE PURCHASER DOWNLOAD THE DEMO VERSION. ACTIVE DATA RECOVERY SOFTWARE WILL NOT ISSUE REFUNDS AFTER THE UNERASER LICENCE FEE HAS BEEN PAID.

Active Data Recovery Software may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give the user any license to these patents, trademarks, copyrights, or other intellectual property.

1. UNERASER LICENSE. UNERASER is licensed, not sold. Copyright laws and international copyright treaties, as well as other intellectual property laws and treaties protect UNERASER.

2. GRANT OF LICENSE.

(a) FREE DEMO VERSION. You may use the DEMO VERSION without charge on an evaluation basis to back up the Master Boot Record (MBR), create disk images, scan drives for deleted partitions, logical drives and files, and preview their contents. The license fee must be paid in full to acquire the REGISTERED VERSION of UNERASER and gain access to features that can restore files and folders back to the HDD.

(b) REDISTRIBUTION OF DEMO VERSION. If you are using the DEMO VERSION on an evaluation basis you may make copies of the DEMO VERSION as you wish, give exact copies of the original DEMO VERSION to anyone, or distribute the DEMO VERSION in its unmodified form via electronic means (Internet, BBS, Shareware distribution libraries, CD-ROM, etc.). You may not charge any fee for the copy or use of the evaluation DEMO VERSION itself, except that you may charge a distribution fee that is reasonably related to any cost you incur distributing the DEMO VERSION (e.g. packaging). You must not represent in any way that you are selling UNERASER itself. Your distribution of the DEMO VERSION will not entitle you to any compensation from Active Data Recovery Software. You must distribute a copy of this EULA with any copy of UNERASER and anyone to whom you distribute UNERASER is subject to this EULA.

(c) REGISTERED VERSION. After you have purchased the license for UNERASER, and have received UNERASER distribution package, you are licensed to copy UNERASER only into the number of floppy disks corresponding to the number of licenses purchased. Under no other circumstances may UNERASER be operated at the same time on more than the number of floppy disks for which you have paid a separate license fee. You may not duplicate UNERASER in whole or in part, except that you may make one copy of UNERASER for backup or archival purposes. You may terminate this license at any time by destroying the original and all copies of UNERASER in whatever form. You may permanently transfer all of your rights under this EULA provided the recipient agrees to the terms of this EULA and provided you turn over all copies of UNERASER (including copies of all prior versions if UNERASER is an upgrade) and retain no copies.

3. RESTRICTIONS. You may not reverse engineer, decompile, or disassemble UNERASER, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation. You may not rent, lease, or lend UNERASER. You may permanently transfer all of your rights under this EULA, provided the recipient agrees to the terms of this EULA. You may not use UNERASER to perform any unauthorized transfer of information (e.g. transfer of files in violation of a copyright) or for any illegal purpose.

4. SUPPORT SERVICES. Active Data Recovery Software may provide support services related to UNERASER. The nature of Support Services may be modified from time to time and is governed by Active Data Recovery Software polices and programs as described in the online documentation and web site, and/or in other materials provided by Active Data Recovery Software. Any supplemental software code provided to you as part of the Support Services shall be considered part of UNERASER and subject to the terms and conditions of this EULA. With respect to technical information you provide to Active Data Recovery Software as part of the Support Services, Active Data Recovery Software may use such information for its business purposes, including for product support and development. Active Data Recovery Software will not utilize such technical information in a form that personally identifies you.

5. TERMINATION. Without prejudice to any other rights, Active Data Recovery Software may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of UNERASER.

6. COPYRIGHT. UNERASER is protected by copyright law and international treaty provisions. You acknowledge that no title to the intellectual property in UNERASER is transferred to you. You further acknowledge that title and full ownership rights to UNERASER will remain the exclusive property of Active Data Recovery Software and you will not acquire any rights to UNERASER except as expressly set forth in this license. You agree that any copies of UNERASER will contain the same proprietary notices which appear on and in UNERASER.

7. DISCLAIMER OF WARRANTY. Active Data Recovery Software expressly disclaims any warranty for UNERASER. UNERASER AND ANY RELATED DOCUMENTATION IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OR MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF UNERASER REMAINS WITH YOU.

8. LIMITATION OF LIABILITY. IN NO EVENT SHALL ACTIVE DATA RECOVERY SOFTWARE OR ITS SUPPLIERS BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES OF ANY KIND ARISING OUT OF THE DELIVERY, PERFORMANCE, OR USE OF UNERASER, EVEN IF ACTIVE DATA RECOVERY SOFTWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY EVENT, ACTIVE DATA RECOVERY SOFTWARE'S ENTIRE LIABILITY UNDER ANY PROVOSION OF THIS EULA SHALL BE LIMITED EXCLUSIVELY TO PRODUCT REPLACEMENT.

Active Data Recovery Software reserves all rights not expressly granted here. Active Data Recovery Software is a registered business name of LSoft Technologies Inc.

# Contents

## Data Recovery Tips

# 1 OVERVIEW

This chapter gives an overview of **Active@ UNERASER** application.

## Welcome to Active@ UNERASER

Active@ UNERASER is a powerful software utility, designed to restore files and directories that may have been accidentally deleted. It allows you to recover files that have been deleted from the Recycle Bin, as well as those deleted when bypassing the Recycle Bin (for example when using **[Shift]+[Delete]**). The utility can be used under a Windows or DOS environment. Active@ UNERASER is compact enough to be installed on and run from a bootable floppy disk, so that the risk of overwriting your data is minimized.

## What's New in Version 3.0

The list below shows new features in version 3.0:

- Optimized drive scanning performance
- Two types of drive and device scan: Basic (Fast) and Thorough (Slow)
- Long and localized filenames full support for display and recovery
- Extended file and drive attributes support for info display
- Encrypted files (EFS) supported in Windows (Console) Application
- Advanced Searching: by Mask, Size, Attribute, Deleted/Existing Only
- Advanced Disk Imaging:
  - Raw and Compressed disk images
  - Composing disk images from raw chunks (created by other tools)
  - Checking Disk Image consistency
- Re-designed user interface to be more convenient
- Re-designed Windows Explorer Style user interface includes:
  - Menus
  - User's Activity Log
  - Progress Bar with timing
  - Extended File Attributes (**[Ctrl]+[Enter]**)

New Installation Features:

- Windows Installer simplifies software installation and configuration
- Active@ UNERASER for Windows (Console Application) is included
- Full featured *.chm Windows help instead of text manual
- Bootable Floppy creator for DEMO and Commercial (registered) version

- Bootable ISO CD-Image for DEMO and Commercial (registered) version

The features listed below are standard UNERASER features that you will recognize from previous versions:

- Recover data from a deleted NTFS partition or drive
- Recover one or several FAT or NTFS partitions or drives, as originally created
- Restore erased partitions even if new drive space has been formatted and has been used to store files, including using the new space for the operating system
- After partition space is restored, recover data from the deleted partition.
- Perform Extended Disk Scan in Interactive DOS mode to ignore Master Boot Record and scan drive contents.
- Easily create a “test” file using command line parameters. Send the test file to Active Data Recovery Software technicians for analysis if you encounter a drive that is difficult to UNERASE.

## Overview of Restoring Deleted Data

No one wants to lose vital data stored on a hard disk. To ensure the integrity of data, backup copies can be the best line of defence. When the unthinkable happens, and data cannot be seen from the operating system, another route must be taken to retrieve lost data.

Table 1-1, below outlines the main causes of data loss, showing the frequency of occurrence:

**Table 1-1** Common Causes of Data Loss

| Cause | Chance |
|---|---|
| Accidental removal of files and folders and then emptying Recycle Bin | ~ 75% |
| After physical damage of critical sectors on HDD (“bad clusters”) some drives become unreadable | ~ 6% |
| Loss of information due to a power failure or power surge | ~ 5% |
| Deletion of logical drive or partition itself then recalling important data on it | ~ 3% |
| Damage of MBR, Partition Table, Volume Boot Sectors by virus | ~ 3% |
| Other | ~ 8% |

When the above cases are assessed, the situation boils down to two states:

1. Individual files or folders have been deleted or overwritten and the drive partition or physical drive can still be displayed by the operating system
2. The partition table is deleted, damaged or overwritten and physical drives or logical drives become invisible to the operating system

### Files Erased

When files or folders have been deleted and the partition is intact, the task is to scan the surface of the hard disk or partition in search of deleted file or folder

entries. The search area is the **Root Folder (FAT)** or the **Master File Table (NTFS)** of the partition. When deleted entries are found, the utility displays them by name and offers the user an opportunity to save whatever is recoverable to a new location.

### Partition Damage

In the case of partition damage, the job for the recovery utility is to analyze the surface of the HDD and retrieve clues around structure of the logical data in order to reconstruct the partition or drive parameters (such as the first sector number, cluster size, file system type, etc.). As this process makes logical connections based on probability, it creates an entity called a **virtual drive**. Without any kind of drive structure, no data can be retrieved. The virtual drive partition will give the user some data to work with.

This User Guide assumes that you have some basic knowledge of hard disk drive and file system organization to be able to understand the recovery terminology and examples.

**Active@ UNERASER** helps you restore data residing on hard drives or floppy drives formatted in any of the following file systems:

- FAT12
- FAT16
- FAT32
- NTFS
- NTFS5
- NTFS + EFS

It works under all DOS and Microsoft Windows family of operating systems:

- MS-DOS
- PC-DOS
- FreeDOS
- DR-DOS
- Windows 95
- Windows 98
- Windows ME
- Windows NT
- Windows 2000
- Windows XP
- Windows Server 2003

Active@ UNERASER supports the following configurations:

- IDE, ATA, SCSI hard drives and floppy disks
- Large sized drives (more than 8 GB)
- Long file names and local language (non-English) file names

- Recovery of compressed, fragmented and encrypted files on NTFS
- Detection and recovery from deleted or damaged file partitions
- Previewing file content and any sectors on the drive in Hex/Text Viewer
- Advanced file search by location, mask, size, attributes
- Disk Image creation and restoring data from it

## Steps to UNERASE Files and Folders

This is a simplified list created to help you get inoformation quickly. In the steps below, click on the link to jump directly to the instruction. These are the steps:

1. If you can see all drive partitions and logical drives, perform a drive scan [Performing a Drive Scan].
2. If you know there is a device or drive partition missing, perform a device scan first, then perform step number 1, above. [Performing a Device Scan]
3. After scanning the drive, search for deleted files and folders. [Searching for Deleted Files and Folders]
4. After finding files and folders to recover, **UNERASE** deleted files and folders. [Unerasing a Deleted File]

# 2 System Requirements

This chapter outlines the minimum requirements for PCs using **Active@ UNERASER**

### Personal Computer

Minimum system requirements for Active@ UNERASER are:

- AT compatible CPU with 386 or newer processor
- 640Kb of RAM or more
- 1.44 Mb floppy diskette drive or CD-ROM drive
- EGA 640x480 or better screen resolution
- Bootable CD-ROM or Floppy disk containing MS-DOS, or startup disk for Windows 95/98/ME/XP
- HDD of type IDE/ATA/SCSI attached to be recovered.

### Active@ UNERASER Version

The performance of **Active@ UNERASER** depends on the version of the application, as displayed in the table below:

**Table 2-1** Differences Between Demo and Registered Versions

| Feature | Demo Version | Registered Version |
|---|---|---|
| Contains **Bootable Floppy Creator** utility and **Bootable ISO CD-Image Containing DOS** application | yes | yes |
| Displays complete physical and logical drive information (DOS and Windows environments) | yes | yes |
| Supports IDE / ATA / SCSI / Zip / Floppy drives (DOS and Windows environments) | yes | yes |
| Supports large drive partitions (more than 128GB) | yes | yes |
| Supports FAT12, FAT16, FAT32, NTFS, NTFS5 (DOS and Windows) and NTFS5+EFS (Windows) file systems | yes | yes |
| Supports Microsoft DOS, Windows 95 / 98 / ME / NT / 2000 / XP / 2003 partitions (DOS and Windows) | yes | yes |
| Detects deleted primary and extended partitions and drives (DOS and Windows) | yes | yes |
| Scans partitions damaged by virus or with damaged MBR (DOS and Windows) | yes | yes |
| Assesses ability to recover files and folders (DOS and Windows) | yes | yes |
| Previews files and folders before recovery (DOS and Windows) | yes | yes |

| Feature | Demo Version | Registered Version |
|---|---|---|
| Supports preview and recovery of *fragmented, compressed, sparse* (DOS and Windows) and *encrypted* files (Windows) | yes | yes |
| Includes advanced search by file name, mask, size range, attribute (DOS and Windows) | yes | yes |
| Displays the content of any sector on the drive with Disk Viewer (DOS and Windows) | yes | yes |
| Creates and works with Raw and Compressed Disk Images (DOS and Windows) | yes | yes |
| Supports local-language and long file names (DOS and Windows) | yes | yes |
| Contains two types of drive and device scan: Basic (fast) and Thorough (slow) (DOS and Windows) | yes | yes |
| Can be saved and run from a bootable 3.5-inch floppy (DOS) | yes | yes |
| Un-erases and saves deleted files and folders onto FAT drives (DOS) and FAT/NTFS drives (Windows) | - | yes |
| Copies existing files and folders from FAT or NTFS partitions to FAT drives (DOS) and FAT/NTFS drives (Windows) | - | yes |

# 3 Preparing to Use The Utility

This chapter describes how to prepare to use the application.

## Preparing a DOS-Bootable Floppy Disk or a Bootable CD-ROM

**Active@ UNERASER** is compact enough to operate from a floppy drive. As an alternate, you may also run Acitve@ UNERASER from a bootable CD-ROM

If you have a bootable floppy or CD-ROM, skip to the Copying Active@ UNERASER section, below.

### Bootable Floppy Creator

When you installed Active@ UNERASER from our web site, a utility named **BootableFloppyCreator.EXE** was included. Follow the steps below to create a new bootable floppy and copy the utility to it:

1. Put a blank 3.5-inch floppy disk in the floppy drive.
2. Start the Bootable Floppy Creator utility using one of the methods below:
   - Open the folder where Active@ UNERASER is installed and double-click **BootableFloppyCreator.EXE**
   - From the Microsoft **Start** button, click **All Programs > Active@ UNERASER > Bootable Floppy Creator**
3. Follow directions to create the startup disk.

#### Creating a Bootable Floppy Manually

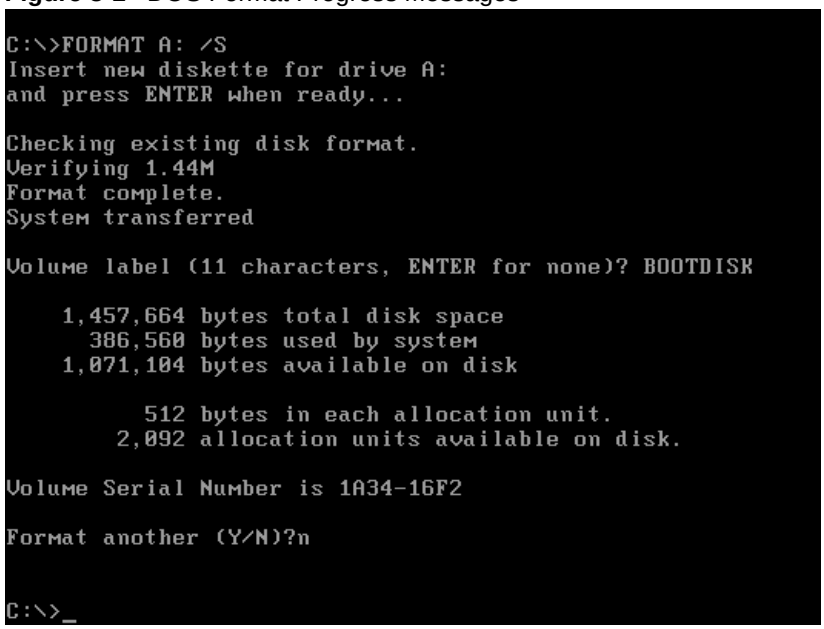
To prepare a bootable floppy manually from MS-DOS or Windows 95/98/ME/XP, put a blank 3.5-inch floppy in the floppy drive (A:) and follow the appropriate instructions below:

1. From MS-DOS or in Command Prompt mode of Windows 95/98:
   a. On the screen, type the format command as follows (see figure below):

```
FORMAT A: /S
```

**b** Follow on-screen messages until process is complete.

**Figure 3-1** DOS Format Progress Messages




**2** From the Windows 95/98/ME screen:

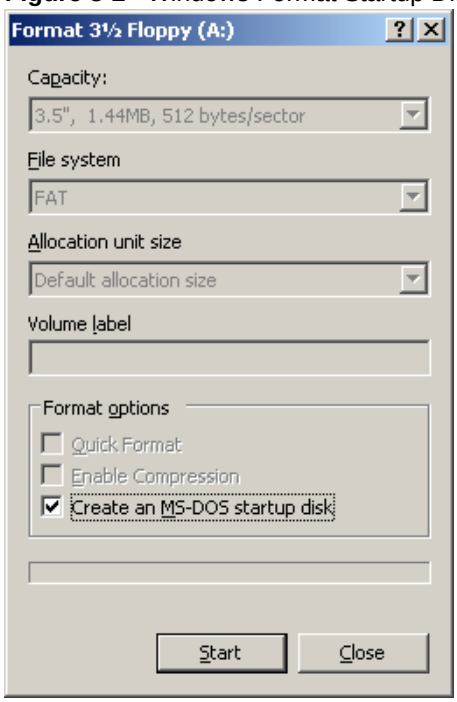
**a** Click the **Start** button and click **Settings**, **Control Panel**.

**b** From the **Control Panel** screen, click **Add/Remove Programs**.

**c** In the **Add/Remove Programs** screen, click the **Startup Disk** tab.

**d** Click the **Startup Disk...** button and follow the screen instructions until the process is complete.

**Figure 3-2** Windows Format Startup Disk Screen



3 From the Windows XP screen:

   a Right-click **A: drive**.

   b From the drop-down menu, click **Format...**

   c Enable the checkbox beside **Create an MS-DOS startup disk**.

   d Click the **Start** button and follow the screen instructions until the process is complete.

## Bootable ISO CD-ROM Image

When you installed Active@ UNERASER from our web site, an ISO image named **Uneraser-Boot-Image.ISO** was included. This image works with a CD-ROM writing utility such as Ahead Nero Express.

Follow the steps below to create a new bootable CD-ROM and copy the utility to it:

1 Put a blank 5-inch writable CD-ROM in the CD-ROM writer drive.

2 Start your CD-ROM writing utility.

3 Follow utility instructions to record a disk from a disk image previously burned onto the hard drive.

4 The CD-ROM image is located in the same folder where Active@ UNERASER was installed. Open this image and write it to the blank CD-ROM.

### Copying Active@ UNERASER

Copy the **Active@ UNERASER** file (UNERASER.EXE) to the bootable floppy disk or startup disk in drive a:.

If you don't have the **Active@ UNERASER** file, download it from our web site: http://www.uneraser.com.

After copying the file onto the floppy disk, remove it from the floppy drive.

Once preparation of the bootable 3.5-inch floppy disk is complete, you are ready to begin recovering data.

# 4 Using The UNERASER Console

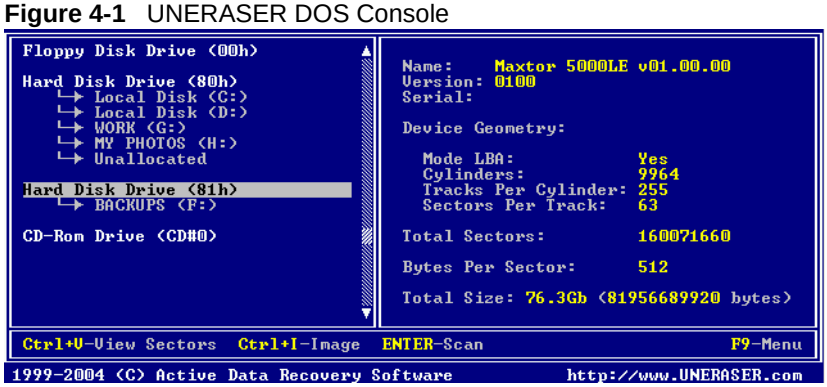
This chapter describes how to use the application with the UNERASER Console interface.

## Starting Active@ UNERASER for DOS

Start the program and display drive information in Microsoft DOS operating system using the following steps:

1 With power off, insert the prepared floppy disk into drive A:. Turn power on and boot from the floppy disk. **Active@ UNERASER** for DOS starts automatically. The UNERASER DOS Console appears.

**Figure 4-1** UNERASER DOS Console




2 On the left side of the window all detected drive devices or partitions are listed and numbered. Beneath each drive device or partition, logical drives appear in a tree formation, as shown in the figure above. Deleted partitions and hard drive space not occupied by partitions are listed as “Unallocated”.

Commands at the bottom of this screen are described below. Click on the links for more information about that item:

- **Ctrl+V - View Sectors** - View the sectors of this drive or partition with the Hex Viewer.
- **Ctrl+I - Image** - Create a Disk Image of this partition or drive. *See* **Creating a Disk Image**.
- **ENTER** - Scan - Perform a simple Drive Scan. *See* **Performing a Drive Scan**.
- **F9 - Menu** - Display a command menu bar at the top of the screen.

3 Use the arrow keys to move the cursor over items on the list of drives. Positioning the cursor on a named drive, displays its information on the right side of the program window.

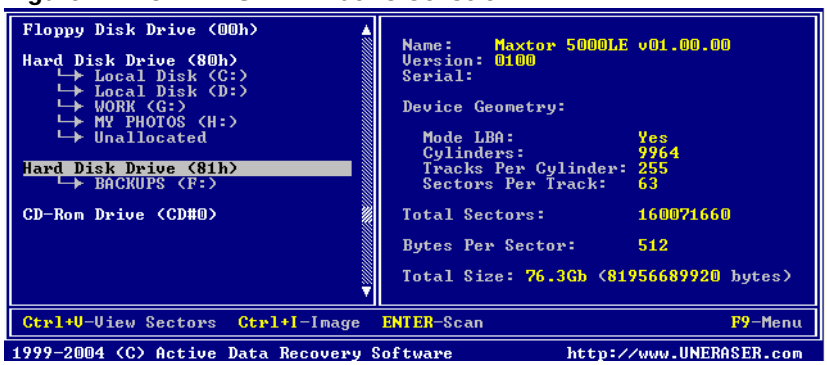
## Starting Active@ UNERASER for Windows

Start the program and display drive information in Microsoft Windows operating system using the following steps:

1 In Windows, click the Microsoft **Start** button.

2 Click **All Programs > Active@ UNERASER > Active@ UNERASER for Windows (Console)**. The **UNERASER Windows Console** appears.

**Figure 4-2** UNERASER Windows Console




3 On the left side of the window all detected drive devices or partitions are listed and numbered. Beneath each drive device or partition, logical drives appear in a tree formation, as shown in the figure above. Deleted partitions and hard drive space not occupied by partitions are listed as “Unallocated”.

Commands at the bottom of this screen are described below. Click on the links for more information about that item:

- **Ctrl+V - View Sectors** - View the sectors of this drive or partition with the Hex Viewer.
- **Ctrl+I - Image** - Create a Disk Image of this partition or drive. *See* .
- **ENTER** - Scan - Perform a simple Drive Scan. *See* **Performing a Drive Scan**.
- **F9 - Menu** - Display a command menu bar at the top of the screen. Choose a menu command with the mouse pointer, or using the arrow keys.

4 Use the mouse pointer or arrow keys to move the cursor over items on the list of drives. Positioning the cursor on a named drive, displays its information on the right side of the program window.

### Command Menu Bar

The table below describes the command menu bar at the top of the console in the main page.

**Table 4-1** Command Menu Bar

| Menu Item | Command | Keyboard Shortcut | Description |
|---|---|---|---|
| File | Refresh Devices | [Ctrl]+[D] | Refresh the report showing all devices on the system. |
| | Save Hardware Info | [Ctrl]+[H] | Save report of hardware information to a text file. |
| | Save Log | [Ctrl]+[L] | Save a log of UNERASER activity to a text file. |
| | Exit | [Esc] | From the main screen, exit the utility and return to DOS. From any other screen, exit that function and return to a previous screen. |
| View | Activity Log | [Ctrl]+[A] | Open the screen with the list of UNERASER activities. |
| | Sectors in Hex Viewer | [Ctrl]+[V] | Open the Hex Viewer to view sectors of the selected partition. |
| Scan | Advanced Device Scan | [Enter] | With a device selected, start the Advanced Device Scan. |
| | Low Level Device Scan | [Ctrl]+[Enter] | With a device selected, start the Low Level Device Scan |
| | Basic Drive Scan | [Enter] | With a logical drive selected, start the Basic Drive Scan. |
| | Advanced Drive Scan | [Ctrl]+[Enter] | With a logical drive selected, start the Advanced Drive Scan. |
| | Search for Files and Folders | [Ctrl]+[F] | Start a Basic Drive Scan and then open the Search parameters dialog box. |
| Image | Create Image | [Ctrl]+[I] | Create a Disk Image file. |
| | Open Image | [Ctrl]+[O] | Open an existing Disk Image file. |
| | Compose Image Manually | | Edit a Disk Image file. |
| | Check Image | [Ctrl]+[C] | Check the validity of an existing disk image. |
| | Check Composed Image | | Check the validity of an existing disk image that is composed from disk image chunks (a disk image created by a third-party utility) |

## Performing a Drive Scan

Before you start data recovery or search for files and folders, the hard drive must be scanned. Scanning uncovers areas of the hard drive that may contain deleted files. If you are searching for a deleted partition or logical drive, read the instructions in **Performing a Device Scan**.

There are two types of drive scan, as described below:

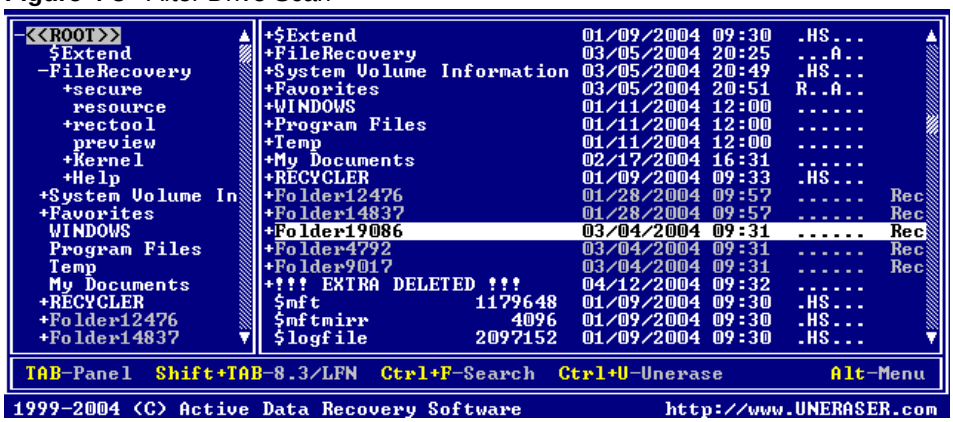
- **Basic Drive Scan** is a quick and general scan. Most of deleted files and folders can be found with this type of scan. Try this scan first **[Enter]**.
- **Advanced Drive Scan** is much slower as it processes the entire surface of the hard drive, detecting all possible clues that may reveal deleted data. If the Basic Drive Scan does not reveal the files you are searching for, try this scan next **[Ctrl]+[Enter]**.

After the cursor is positioned over a logical drive, press **[Enter]** to scan that drive with a Basic Drive Scan.

To stop (cancel) the scanning process, press **[Esc]** at any time.

After completing a drive scan the screen displays the areas where data may have been deleted, similar to the figure below:

**Figure 4-3** After Drive Scan




Commands at the bottom of this screen are described below:

- **TAB-Panel** - Switch between left and right panels.
- **Shift+TAB-8.3/LFN** - Change file names from 8.3 notation to Long File Name notation.
- **Ctrl+F-Search** - Open the search parameter dialog box to begin search.
- **Ctrl+U-Unerase** - Unerase file.
- **F9-Menu** - Open the command menu at the top of the console.

Try to locate your files and folders visually by going through the folder tree. Alternately, use the instructions found in **Searching for Deleted Files and Folders**.

## Performing a Device Scan

Do a Device Scan when a partition or logical drive has been deleted or damaged. In other words, use it when you are unable to locate a drive listed under **My Computer**.

A Device Scan processes the surface of the physical device trying to locate all possible logical drives and partitions, whether they are existing, damaged or deleted.

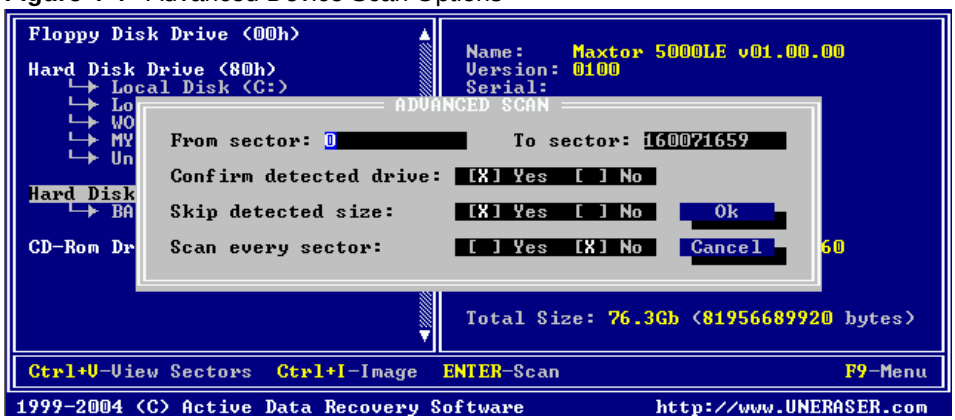
There are two types of device scan, as described below:

- **Advanced Device Scan** reads each hard drive track and looks for the boot sectors of deleted or damaged partitions. If found, a boot sector is interpreted as a drive. You can scan it and the look for deleted files and folders.
- **Low Level Device Scan** reads each hard drive sector looking for boot sectors and tries to reconstruct drive structures based on remnants of the drive's system structures that it finds. This is very slow process but usually detects more partitions.

To perform a Device Scan follow the steps below:

1. From the **UNERASER DOS/Console** main screen, choose a physical device or partition that contains your data. It may be a hard disk drive or a floppy disk drive.
2. Press **[F9] > Scan > Advanced Device Scan** or press **[Enter]** to start the **Advanced Device Scan**. The **Advanced Scan** dialog box appears.
3. Alternately, press **[F9] > Scan > Low Level Scan** or press **[Ctrl]+[Enter]** to start the **Low Level Device Scan**.

**Figure 4-4** Advanced Device Scan Options




4. Specify scanning parameters using **[Tab]** and arrows keys. Press **[Enter]** to start the scan. Press **[Esc]** to close this dialog box and return to a previous screen. Use the descriptions below to help configure the device scan:

- **From sector: - To sector** - If you know the approximate sector location, enter a range here to reduce scanning time.
- **Confirm detected drive** - **Yes** (default) = If a new drive is detected during a device scan, a dialog box appears and waits for an answer whether to add

the new device to the tree or not. **No** = Any newly detected device is added to the tree automatically with no human interaction.

- **Skip detected size** - **Yes** (default) = If a new drive is detected during a device scan, the size of the device is calculated and the area is not scanned (it is skipped). Scanning continues past the boundary of this new device. **No** = The new device is scanned.
- **Scan every sector** - Usually a Drive Boot Sector is located at the beginning of each track (every 63rd sector). **Yes** = Scan every sector (slow). **No** (default) = Scan every track (63 times faster).

5 After the scan starts, you can watch the progress bar and wait until the device scan is finished. You can cancel the operation anytime by pressing **[Esc]**.

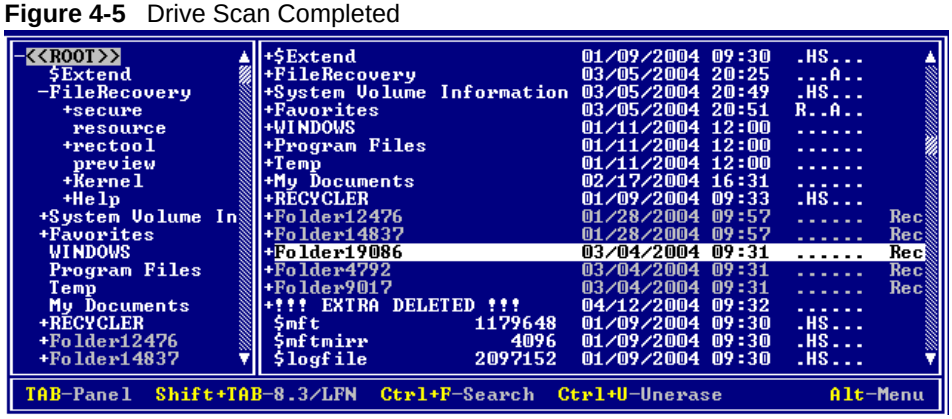
6 After the device scan is complete, all detected partitions and logical drives appear. These drives are ready for the Drive Scan process the same way as with regular drives.

## Searching for Deleted Files and Folders

Before performing the search process, scan the drive to reveal areas that contain damaged or deleted files and folders. Follow the steps below to locate individual deleted files and folders:

1. In Windows, check the **Windows Recycle Bin** to see if the file or folder is there. If it is, use standard **Windows Restore** command to recover it from there. If not, continue with step 2.
2. If working in DOS, restart your PC in DOS mode with the UNDELETE bootable floppy. If working in Windows, start Active@ UNERASER.
3. If you know exactly where the files or folders were located before being deleted, use **Performing a Drive Scan** procedure.

**Figure 4-5** Drive Scan Completed




4. After drive has been scanned go directly to the folder where you know the files or folders should be.

   If you do not see your files where they should be, look under the **!!!EXTRA DELETED!!!** tree node. If the parent of the file or folder has been deleted or damaged, most likely the item you are looking for will be in this virtual folder.

   If you are still unable to locate your data, proceed to the next step.

***(i)*** *Note: If you see **!!!EXTRA!!!**, use **[Shift]+[Tab]** to change the file name display from 8.3 to Long File Name (LFN).*

5. If you cannot find the deleted file or folder by viewing this list, try Search mode. Position marker to the Drive or Folder and press [F9 > Scan > Search for Files

or press **[Ctrl]+[F]**. The **Search** dialog box appears. Specify the search criteria here:

**Figure 4-6** Search Dialog Box


Use the **[Tab]** key to move from field to field. Use the spacebar to select or uncheck each item. An “X” indicates the item is selected. Use the descriptions below to help configure the search parameters:

- **Enter Mask** - Define a search pattern, for example type “*.doc” to find all Microsoft Word files (with **doc** extension). For help with this, see **Searching for Files by Name**, below.
- **Match case** - Case sensitive or non-sensitive search.
- **Size** - If you know the size range for files to be searched, you can reduce the length of the search time by eliminating files larger or smaller than the sizes you indicate here.
- **Attributes** - Each selected attribute will be included in the search. If you know, for example that the file you are looking for is not hidden, then uncheck that attribute. Search will ignore that attribute in all files.
- **Exclude Existing** - With this item selected, search will not report on files that presently exist on the drive. Only deleted or damaged files and folders will be reported.
- **Exclude Deleted** - With this item selected, search will not report on files that are known to be deleted. It will report on damaged files.

Press **[Enter]** to run the search. After the search is complete, examine the list of matched files and folders.

6 If no files were found after Basic Drive Scan, run an Advanced Drive Scan to look more closely at all the drive's surfaces. After the Advanced Scan, repeat Step 4, above.

If, after completing all five steps above, your files and folders still cannot be found, it is likely that the physical drive space has been completely overwritten with other data. If this is the case, no recovery tools can help you.

Overwriting a drive’s physical space can happen when a lot of writing operations occur on a drive (for example, during software installation). As well, Windows operating system sometimes creates temporary files for different processes. This might affect an area with deleted data as well.

### Searching for Files by Name

If you know the name or part of the name of the deleted files or folders, create a search pattern, similar to a search in Microsoft DOS or Windows.

The asterisk or star symbol (*) is used as a wild-card character. The search engine looks for the file name, replacing the star with any number of characters. The question mark symbol (?) is used to replace a single variable character. The search engine looks for the file name and replaces the question mark with another single character.

**Table 4-2** Examples of Searchable Expressions

| Example | Search Results |
|---|---|
| * | All named files and folders |
| *.txt | All files with the suffix “txt” |
| My*.* | All files starting with “My” |
| My p????. | All files starting with “My p” text, having eight characters in the file name and no suffix or extension |
| myfile.txt | Only the file named “myfile.txt” will be displayed, if found |

## Using the Hex/Text Viewer

Use the Hex/Text Viewer to view sectors in a file, a logical drive or physical device. Follow the steps below:

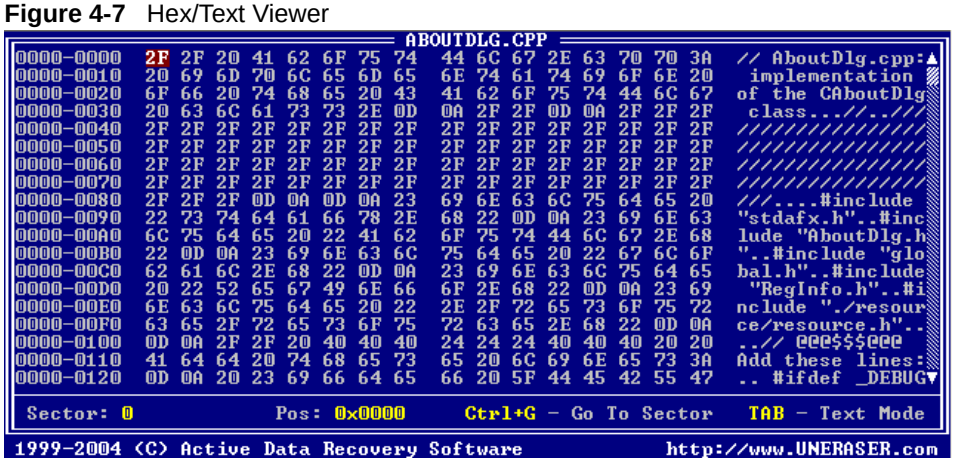
1 Start **Active@ UNERASER** and use the arrow keys to set the marker on a file, a drive or a device.

2 Run the **View Sectors** command by one of the following methods:

- Press **[Ctrl]+[V]** key combination
- Press **[F9] > View > Sectors in Hex/Text Viewer**.

A preview window appears and you can see the data in Hex/Text format, similar to the figure below:

**Figure 4-7** Hex/Text Viewer




3 You can inspect the data using keyboard keys, as described below:

- **[TAB]** - Switch between **Text** mode and **Hex/Text** mode

- **[Ctrl]+[G]** - Go to a specific sector number. Type the sector number manually
- **Arrow Keys** - Move the red marker within view area
- **[Page Up], [Page Down]** - Go one sector backward or forward
- **[Home], [End]** - Jump to the first or last sector

### Numbering Sectors

Sectors are numbered in three different ways, as described below:

**Files** (relative file sectors) - Numbering starts at sector zero and continues to last sector of the file (calculated as file size divided by 512 plus one)

**Logical Drives** (logical sectors) - Numbering starts at sector zero and continues to the last sectors on the drive (calculated as drive size divided by 512 minus one)

**Physical Device** (physical sectors) - Number starts at sector zero and continues to the last sector on the device (calculated as device size divided by 512 minus one)

## UNERASE Deleted Data

Use the scan or search procedures above to identify deleted files and folders and to inspect the contents to see if the data is worth recovering. Use the methods below to recover deleted files and folders.

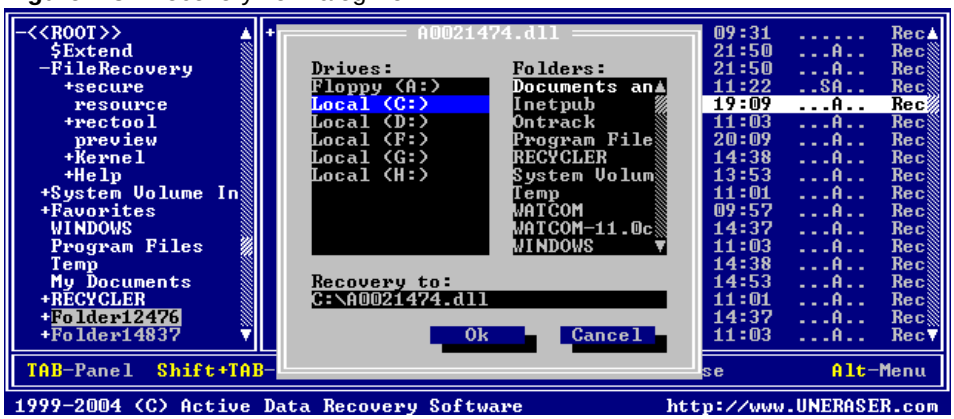
### Unerasing a Deleted File

To unerase file (copy its content to another safe location) execute the Unerase command by one of the following methods:

- Press **[Ctrl]+[U]** key combination
- Press **[F9] > File/Folder > UNERASE**

A dialog box appears, similar to the figure below:

**Figure 4-8** Recovery To Dialog Box




Select a secure drive and folder where the restored file will be saved.

Press **[Enter]** to begin the process.

After the recovery process is complete, verify the contents of recovered files and subfolders. In some cases, a file can not be restored completely because its contents or a part of it has been overwritten.

***(!)*** *Important: For the safety reasons, the utility warns you if you are trying to write the restored file back into the same drive. A newly-created file requires space to be saved. It is possible to overwrite the contents of the other deleted files or part of the very file you are trying to recover. Always restore files to another logical removable, floppy or network drive.*

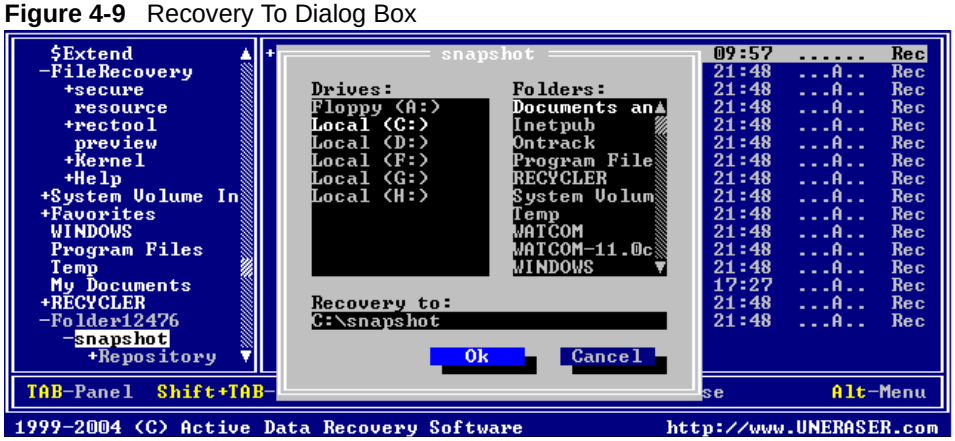
## Unerasing a Deleted Folder

To restore the contents of a folder recursively (including files and subfolders), execute the Unerase command by one of the following methods:

- Press **[Ctrl]+[U]** key combination
- Press **[F9] > File/Folder > UNERASE**

A dialog box appears, similar to the figure below:

**Figure 4-9** Recovery To Dialog Box




Select a secure drive and folder where the restored folder contents will be saved.

Press **[Enter]** to begin the process.

After the recovery process is complete, verify the contents of recovered files and subfolders.

***(i)*** *Note: If the folder that you've recovered under DOS contains files with long filenames, the batch file _RENAME.BAT is created at the destination folder and each subfolder. When you run this file under the Windows environment later on, it will rename temporary short (8.3) DOS filenames to more descriptive long file names supported by Windows. You cannot create files with long filenames under the DOS environment as long as DOS does not support them.*

***(!)*** *Important: For the safety reasons, the utility warns you if you are trying to write the restored file back into the same drive. A newly-created file requires space to be*

*saved. It is possible to overwrite the contents of the other deleted files or part of the very file you are trying to recover. Always restore files to another logical removable, floppy or network drive.*

## Creating a Disk Image

A Disk Image is a mirror copy of your entire logical drive or physical device stored as set of files. It may be a good idea to create a Disk Image for a drive containing deleted files that you want to recover, if you have enough space on another drive.

**"Why should I create a Disk Image on a drive that holds my deleted files?"**

If you run into difficulty, or do something wrong while attempting to recover deleted files (for example, by recovering them onto the originating drive instead of a different drive, thereby destroying their contents), you will be able to recover these deleted files and folders from the Disk Image that you have wisely created.

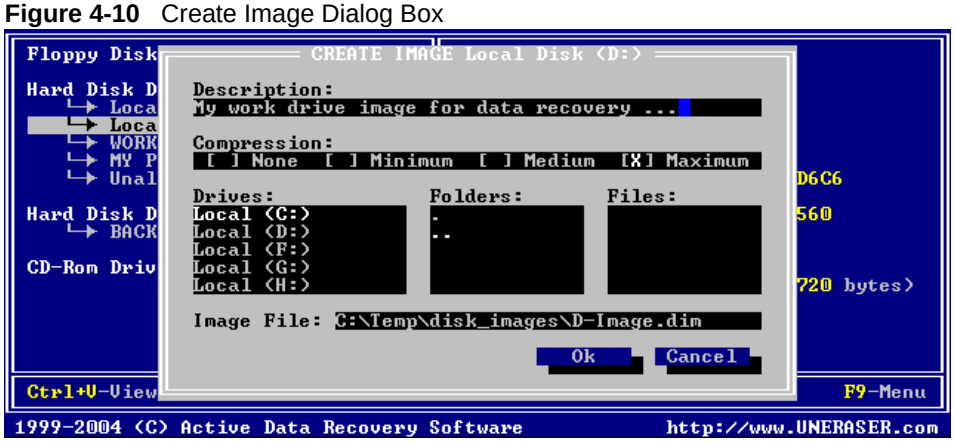
Disk Image consists of configuration file (with an extension .DIM) and set of files having extensions .000, .001, .002 and so on. The size of each file is 2 Gb to accommodate most file systems (FAT16 and FAT32 file systems do not support file sizes larger than 2 Gb and 4 Gb respectively).

Here are the steps to create a Disk Image:

1. Start **Active@ UNERASER** and select a drive or hardware device.
2. Run the **Create Image** command by doing one of the following:
   - Press **[Ctrl]+[I]** key combination
   - Press **[F9] > Image > Create Image**

   The Create Image dialog box appears, similar to the figure below:

**Figure 4-10** Create Image Dialog Box




3. In this dialog box, create a description, choose a compression ratio, a Disk Image location and create an Image File name.

   Navigate to the **Ok** button and press **[Enter]** when all parameters are complete.

4 Watch the progress and wait while drive's contents are copied to the new location. You can cancel the process of image creation anytime by pressing **[Esc]**.

***(!)*** *Important: The Target Location for the Create Image command must always be specified on another drive.*

#### Checking the Disk Image

After a Disk Image is created it is a good idea to check its validity to be sure that everything was written properly.

To check the validity of created Disk Image, follow these steps:

1 Start **Active@ UNERASER**.

2 Run the **Check Image** command by doing one of the following:

- Press **[Ctrl]+[C]** key combination
- Press **[F9] > Image > Check Image**

The **File Open** dialog box appears.

3 Select the Disk Image configuration file (*.DIM) in the File Open dialog and press **Ok**.

4 Watch the progress and view the result. You can stop data verification anytime by pressing **[Esc]**.

### Working with Disk Image

A Disk Image is a mirror copy of your entire logical drive or physical device stored as set of files. It may be a good idea to create a Disk Image for a drive containing deleted files that you want to recover, if you have enough space on another drive.

To open a Disk Image (via configuration file) follow these steps:

1 Start **Active@ UNERASER**.

2 Run the **Open Image** command by doing one of the following:

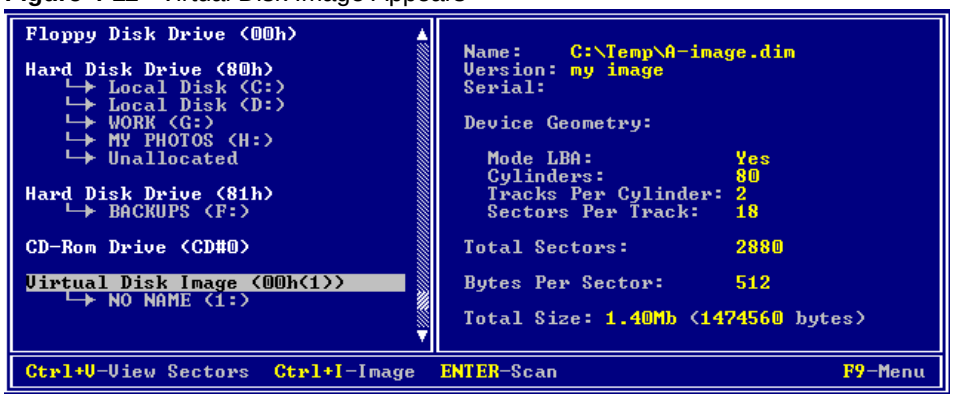
- Press **[Ctrl]+[O]** key combination
- Press **[F9] > Image > Open Image**

The Open Image dialog box appears.

3 Select an existing Disk Image (file with DIM extension) in the Open Image dialog and press [OK].

A Virtual Disk Image appears in the list of devices.

**Figure 4-11** Virtual Disk Image Appears




4 The opened disk image appears underneath existing devices in the tree in the left pane of the console. Work with an opened Disk Image the same way as with any regular drive or device, i.e. scan, find and restore files from it.

### Opening a Third Party Disk Image

If you have created a Raw Disk Image using tools other than Active@ UNERASER, you still can access and recover data from it by composing disk image from chunks.

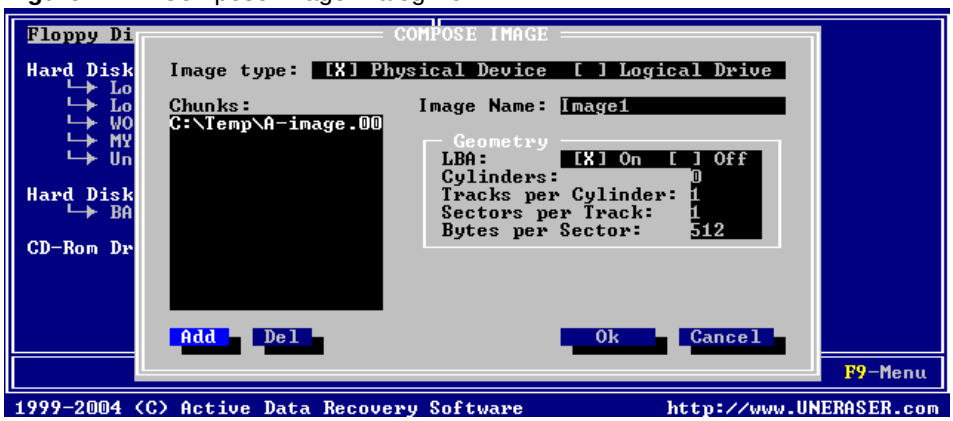
To open a Disk Image from chunks (without configuration file), follow these steps:

1 Start **Active@ UNERASER**.

2 Press **[F9] > Image > Run Compose Image**.

The **Compose Image** dialog box appears.

**Figure 4-12** Compose Image Dialog Box




3 In this dialog box, build a Disk Image by adding one or more chunks of the image. Press **Add**. A **File Open** dialog box appears.

4 Select files and press **Ok**. They appear in the list under **Chunks:**.

5 Specify image parameters in the Compose Image dialog box. A description of the parameters follows:

- **Image Type** - Specify whether the image was created for a logical drive or a physical device
- **Name** - Image name as it will be displayed in the device tree
- **Geometry** - Device geometry (if you know it). You can leave default values here, it will work, but performance of the utility will be a bit slow.

Press **Ok** when all parameters are complete.

6 After refreshing the devices list, the new Disk Image appears in the tree.

Work with this image the same way as with any regular device or drive. Scan it for partitions and files, find and restore files from it.

## Long File Names Support

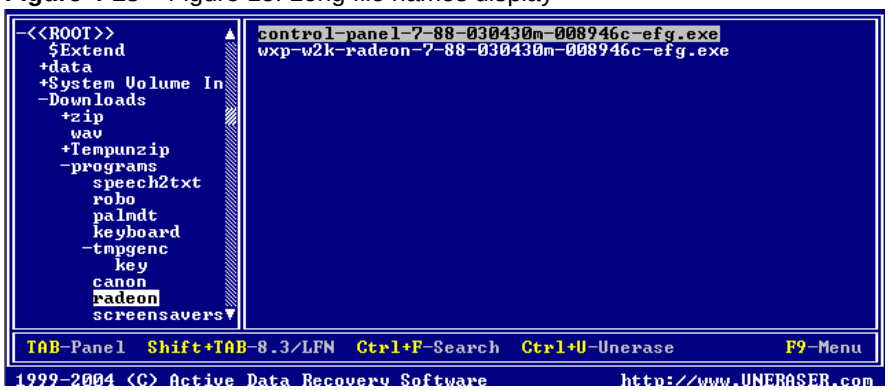
Since Windows 95, filenames are not limited in size to the 8.3 pattern and can have a length of up to 255 characters.

Standard View displays all files and folders the same way as DOS does, i.e. in 8.3 formats. However sometimes it is not convenient to see only first symbols of the long file name.

To display long filenames (up to 36 symbols):

1. Boot in DOS mode and run **Active@ UNERASER**
2. Scan the particular drive by pressing **[Enter]**.
3. Press the **[Tab]** key to switch to long filenames view (Figure 15)

**Figure 4-13** Figure 15: Long file names display




## Recovering Files With Long Names

It is important to note that you cannot create or recover files with long filenames under the DOS environment as long as DOS does not support long filenames. Files with long file names are saved with a temporary short file name (in the 8.3 format).

If you have recovered files and folders from the DOS environment and those files or folders have long filenames, you can recover the long filenames with a utility provided by Active@ UNERASER. This utility is a batch file named _RENAME.BAT. It is created in the UNERASE destination folder and in each subfolder.

After completing the recovery process in DOS, restart your PC in Windows. Run _RENAME.BAT in the Windows environment to restore the more descriptive long filenames to your files.

# 5 Using Command-Line Parameters

This chapter describes how to use the application with command line parameters.

## Overview of Command Line Parameters

**Active@ UNERASER** for DOS supports a set of command line parameters. To view them and their definitions, type:

```
A:\>UNERASER -?
```

**Table 5-1** Command Line Parameters

| Parameter | Description | Note |
|---|---|---|
| | No parameter | The DOS Interactive screens appear. |
| -? | Question mark | The table of parameters appears. |
| -writelog[=fullpath] | Create a log file with debug information | |
| -writetest[=fullpath] | Create a file containing hardware configuration | |
| -retries=[0...99] | When an error is detected, the utility attempts to read and write the area this number of times before moving on. | |

# 6 Data Recovery Concepts

This chapter describes some basic concepts that might help when unerasing data.

## Hard Disk Drive Basics

A hard disk is a sealed unit containing a number of **platters** in a stack. Hard disks may be mounted in a horizontal or a vertical position. In this description, the hard drive is mounted horizontally.

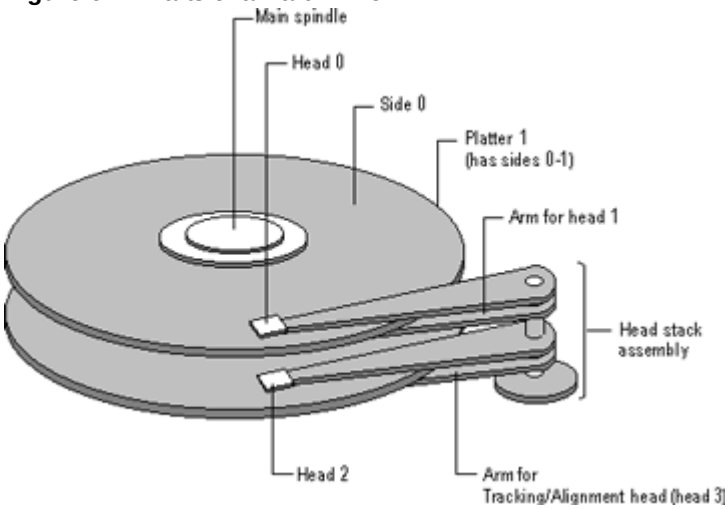
Electromagnetic read/write **heads** are positioned above and below each platter. As the platters spin, the drive heads move in toward the center surface and out toward the edge. In this way, the drive heads can reach the entire surface of each platter.

### Making Tracks

On a hard disk, data is stored in thin, concentric bands. A drive head, while in one position can read or write a circular ring, or band called a **track**. There can be more than a thousand tracks on a 3.5-inch hard disk. Sections within each track are called **sectors**. A sector is the smallest physical storage unit on a disk, and is almost always 512 bytes (0.5 kB) in size.

The figure below shows a hard disk with two platters.

**Figure 6-1** Parts of a Hard Drive




The structure of older hard drives (i.e. prior to Windows 95) will refer to a **cylinder/ head/ sector** notation. A cylinder is formed while all drive heads are in the same position on the disk. The tracks, stacked on top of each other form a cylinder. This scheme is slowly being eliminated with modern hard drives. All new disks use a translation factor to make their actual hardware layout appear

continuous, as this is the way that operating systems from Windows 95 onward like to work.

To the operating system of a computer, tracks are logical rather than physical in structure, and are established when the disk is low-level formatted. Tracks are numbered, starting at 0 (the outermost edge of the disk), and going up to the highest numbered track, typically 1,023, (close to the center). Similarly, there are 1,024 cylinders (numbered from 0 to 1,023) on a hard disk.

The stack of platters rotate at a constant speed. The drive head, while positioned close to the center of the disk reads from a surface that is passing by more slowly than the surface at the outer edges of the disk. To compensate for this physical difference, tracks near the outside of the disk are less-densely populated with data than the tracks near the center of the disk. The result of the different data density is that the same amount of data can be read over the same period of time, from any drive head position.

The disk space is filled with data according to a standard plan. One side of one platter contains space reserved for hardware track-positioning information and is not available to the operating system. Thus, a disk assembly containing two platters has three sides available for data. Track-positioning data is written to the disk during assembly at the factory. The system **disk controller** reads this data to place the drive heads in the correct sector position.

## Sectors and Clusters

A sector, being the smallest physical storage unit on the disk, is almost always 512 bytes in size because 512 is a power of 2 (2 to the power of 9). The number 2 is used because there are two states in the most basic of computer languages - on and off.

Each disk sector is labelled using the factory track-positioning data. Sector identification data is written to the area immediately before the contents of the sector and identifies the starting address of the sector.

The optimal method of storing a file on a disk is in a **contiguous** series, that is, all data in a stream stored end-to-end in a single line. As many files are larger than 512 bytes, it is up to the file system to allocate sectors to store the file's data. For example, if the file size is 800 bytes, two 512 k sectors are allocated for the file. A **cluster** is typically the same size as a sector. These two sectors with 800 bytes of data are called two clusters. They are called clusters because the space is reserved for the data contents. This process protects the stored data from being over-written. Later, if data is appended to the file and its size grows to 1600 bytes, another two clusters are allocated, storing the entire file within four clusters.

If contiguous clusters are not available (clusters that are adjacent to each other on the disk), the second two clusters may be written elsewhere on the same disk or within the same cylinder or on a different cylinder - wherever the file system finds two sectors available. A file stored in this non-contiguous manner is considered to be **fragmented**. Fragmentation can slow down system performance if the file system must direct the drive heads to several different addresses to find all the data in the file you want to read. The extra time for the heads to travel to a number of addresses causes a delay before the entire file is retrieved.

Cluster size can be changed to optimize file storage. A larger cluster size reduces the potential for fragmentation, but increases the likelihood that clusters will have unused space. Using clusters larger than one sector reduces fragmentation, and reduces the amount of disk space needed to store the information about the used and unused areas on the disk.

## The FAT File System

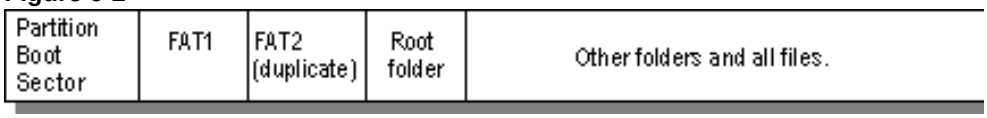
The File Allocation Table (FAT) file system is a simple file system originally designed for small disks and simple folder structures. The FAT file system is named for its method of organization, the file allocation table, which resides at the beginning of the volume. To protect the volume, two copies of the table are kept, in case one becomes damaged. In addition, the file allocation tables and the root folder must be stored in a fixed location so that the files needed to start the system can be correctly located.

A volume formatted with the FAT file system is allocated in clusters. The default cluster size is determined by the size of the volume. For the FAT file system, the cluster number must fit in 16 bits and must be a power of two.

### Structure of a FAT Volume

The figure below illustrates how the FAT file system organizes a volume.

**Figure 6-2**




This section covers information about the FAT system. Topics covered are:

- FAT Partition Boot Sector
- FAT File System
- FAT Root Folder
- FAT Folder Structure
- FAT32 Features

Table 6-1 displays differences between the FAT systems:

**Table 6-1** Differences Between FAT Systems

| System | Bytes Per Cluster Within File Allocation Table | Cluster Limit |
|---|---|---|
| FAT12 | 1.5 | Fewer than 4,087 clusters. |
| FAT16 | 2 | Between 4,087 and 65,526 clusters, inclusive. |
| FAT32 | 4 | Between 65,526 and 268,435,456 clusters, inclusive. |

For more detailed information see resource kits on Microsoft's web site http://www.microsoft.com/windows/reskits/webresources/default.asp or Microsoft Developers Network (MSDN) http://msdn.microsoft.com.

### FAT Partition Boot Sector

The Partition Boot Sector contains information that the file system uses to access the volume. On x86-based computers, the Master Boot Record use the Partition Boot Sector on the system partition to load the operating system kernel files.

Table 6-2 describes the fields in the Partition Boot Sector for a volume formatted with the FAT file system.

**Table 6-2** Fields in Partition Boot Sector (FAT File System)

| Byte Offset (in hex) | Field Length | Sample Value | Description |
|---|---|---|---|
| 00 | 3 bytes | EB 3C 90 | Jump instruction. |
| 03 | 8 bytes | MSDOS5.0 | OEM Name in text |
| 0B | 25 bytes | | BIOS Parameter Block |
| 24 | 26 bytes | | Extended BIOS Parameter Block |
| 3E | 448 bytes | | Bootstrap code |
| 1FE | 2 bytes | 0x55AA | End of sector marker |

Table 6-3 describes BIOS Parameter Block and Extended BIOS Parameter Block Fields.

**Table 6-3** BIOS Parameter Block and Extended BIOS Parameter Block Fields

| Byte Offset | Field Length | Sample Value | Description |
|---|---|---|---|
| 0x0B | WORD | 0x0002 | Bytes per Sector. The size of a hardware sector. For most disks in use in the United States, the value of this field is 512. |
| 0x0D | BYTE | 0x08 | Sectors Per Cluster. The number of sectors in a cluster. The default cluster size for a volume depends on the volume size and the file system. |
| 0x0E | WORD | 0x0100 | Reserved Sectors. The number of sectors from the Partition Boot Sector to the start of the first file allocation table, including the Partition Boot Sector. The minimum value is 1. If the value is greater than 1, it means that the bootstrap code is too long to fit completely in the Partition Boot Sector. |
| 0x10 | BYTE | 0x02 | Number of file allocation tables (FATs). The number of copies of the file allocation table on the volume. Typically, the value of this field is 2. |
| 0x11 | WORD | 0x0002 | Root Entries. The total number of file name entries that can be stored in the root folder of the volume. One entry is always used as a Volume Label. Files with long filenames use up multiple entries per file. Therefore, the largest number of files in the root folder is typically 511, but you will run out of entries sooner if you use long filenames. |
| 0x13 | WORD | 0x0000 | Small Sectors. The number of sectors on the volume if the number fits in 16 bits (65535). For volumes larger than 65536 sectors, this field has a value of 0 and the Large Sectors field is used instead. |

| Byte Offset | Field Length | Sample Value | Description |
|---|---|---|---|
| 0x15 | BYTE | 0xF8 | Media Type. Provides information about the media being used. A value of 0xF8 indicates a hard disk. |
| 0x16 | WORD | 0xC900 | Sectors per file allocation table (FAT). Number of sectors occupied by each of the file allocation tables on the volume. By using this information, together with the Number of FATs and Reserved Sectors, you can compute where the root folder begins. By using the number of entries in the root folder, you can also compute where the user data area of the volume begins. |
| 0x18 | WORD | 0x3F00 | Sectors per Track. The apparent disk geometry in use when the disk was low-level formatted. |
| 0x1A | WORD | 0x1000 | Number of Heads. The apparent disk geometry in use when the disk was low-level formatted. |
| 0x1C | DWORD | 3F 00 00 00 | Hidden Sectors. Same as the Relative Sector field in the Partition Table. |
| 0x20 | DWORD | 51 42 06 00 | Large Sectors. If the Small Sectors field is zero, this field contains the total number of sectors in the volume. If Small Sectors is nonzero, this field contains zero. |
| 0x24 | BYTE | 0x80 | Physical Disk Number. This is related to the BIOS physical disk number. Floppy drives are numbered starting with 0x00 for the A disk. Physical hard disks are numbered starting with 0x80. The value is typically 0x80 for hard disks, regardless of how many physical disk drives exist, because the value is only relevant if the device is the startup disk. |
| 0x25 | BYTE | 0x00 | Current Head. Not used by the FAT file system. |
| 0x26 | BYTE | 0x29 | Signature. Must be either 0x28 or 0x29 in order to be recognized by Windows NT. |
| 0x27 | 4 bytes | CE 13 46 30 | Volume Serial Number. A unique number that is created when you format the volume. |
| 0x2B | 11 bytes | NO NAME | Volume Label. This field was used to store the volume label, but the volume label is now stored as special file in the root directory. |
| 0x36 | 8 bytes | FAT16 | System ID. Either FAT12 or FAT16, depending on the format of the disk. |

For more detailed information see resource kits on Microsoft's web site http://www.microsoft.com/windows/reskits/webresources/default.asp or Microsoft Developers Network (MSDN) http://msdn.microsoft.com

## File Allocation System

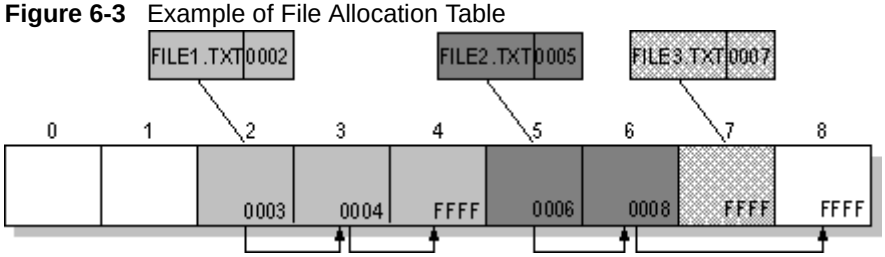
The FAT file allocation system is named for its method of organization, the file allocation table, which resides at the beginning of the volume. To protect the volume, two copies of the table are kept, in case one becomes damaged. In addition, the file allocation tables must be stored in a fixed location so that the files needed to start the system can be correctly located.

The file allocation table contains the following types of information about each cluster on the volume (see example below for FAT16):

- Unused (0x0000)
- Cluster in use by a file
- Bad cluster (0xFFF7)
- Last cluster in a file (0xFFF8-0xFFFF)

There is no organization to the FAT folder structure, and files are given the first available location on the volume. The starting cluster number is the address of the first cluster used by the file. Each cluster contains a pointer to the next cluster in the file, or an indication (0xFFFF) that this cluster is the end of the file. These links and end of file indicators are shown below.

**Figure 6-3** Example of File Allocation Table




This illustration shows three files. The file File1.txt is a file that is large enough to use three clusters. The second file, File2.txt, is a fragmented file that also requires three clusters. A small file, File3.txt, fits completely in one cluster. In each case, the folder structure points to the first cluster of the file.

For more detailed information see resource kits on Microsoft's web site http://www.microsoft.com/windows/reskits/webresources/default.asp or Microsoft Developers Network (MSDN) http://msdn.microsoft.com

## FAT Root Folder

The root folder contains an entry for each file and folder on the root. The only difference between the root folder and other folders is that the root folder is on a specified location on the disk and has a fixed size (512 entries for a hard disk, number of entries on a floppy disk depends on the size of the disk).

See Folder Structure topic for details about folder organization.

For more detailed information see resource kits on Microsoft's web site http://www.microsoft.com/windows/reskits/webresources/default.asp or Microsoft Developers Network (MSDN) http://msdn.microsoft.com

## FAT Folder Structure

Folders have set of 32-byte **Folder Entries** for each file and subfolder contained in the folder (see example figure below).

The **Folder Entry** includes the following information:

- Name (eight-plus-three characters)
- Attribute byte (8 bits worth of information, described later in this section)
- Create time (24 bits)

- Create date (16 bits)
- Last access date (16 bits)
- Last modified time (16 bits)
- Last modified date (16 bits.)
- Starting cluster number in the file allocation table (16 bits)
- File size (32 bits)

There is no organization to the FAT folder structure, and files are given the first available location on the volume. The starting cluster number is the address of the first cluster used by the file. Each cluster contains a pointer to the next cluster in the file, or an indication (0xFFFF) that this cluster is the end of the file. See File Allocation System for details.

The information in the folder is used by all operating systems that support the FAT file system. In addition, Windows NT can store additional time stamps in a FAT folder entry. These time stamps show when the file was created or last accessed and are used principally by POSIX applications.

Because all entries in a folder are the same size, the attribute byte for each entry in a folder describes what kind of entry it is. One bit indicates that the entry is for a subfolder, while another bit marks the entry as a volume label. Normally, only the operating system controls the settings of these bits.

A FAT file has four attributes bits that can be turned on or off by the user — archive file, system file, hidden file, and read-only file.

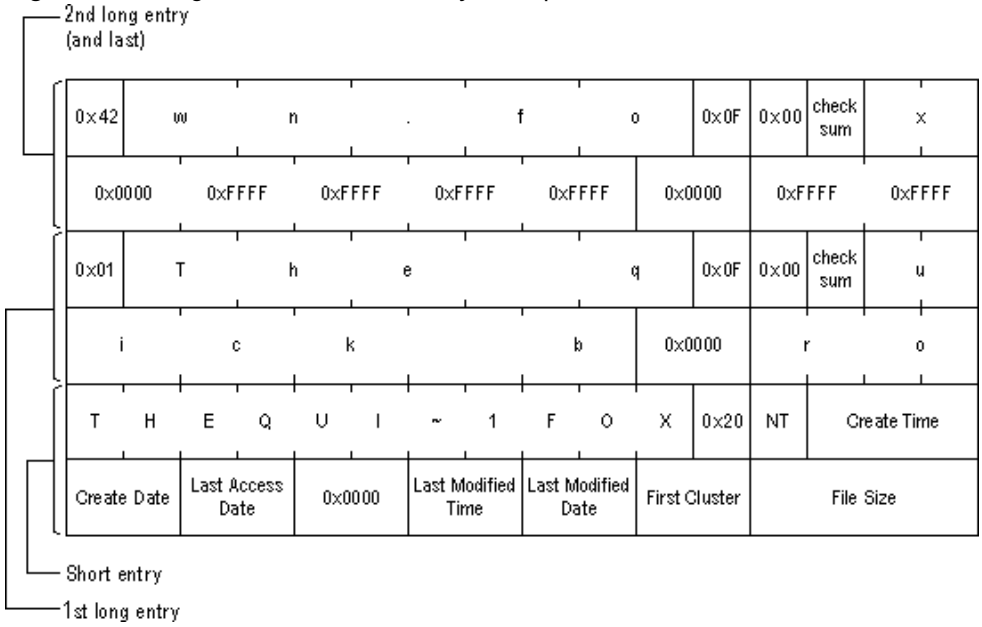
### Filenames on FAT Volumes

Beginning with Windows NT 3.5, files created or renamed on FAT volumes use the attribute bits to support long filenames in a way that does not interfere with how MS-DOS or OS/2 accesses the volume. Whenever a user creates a file with a long filename, Windows creates an eight-plus-three name for the file. In addition to this conventional entry, Windows creates one or more secondary folder entries for the file, one for each 13 characters in the long filename. Each of these secondary folder entries stores a corresponding part of the long filename in Unicode. Windows sets the volume, read-only, system, and hidden file attribute bits of the secondary folder entry to mark it as part of a long filename. MS-DOS and OS/2 generally ignore folder entries with all four of these attribute bits set, so these entries are effectively invisible to these operating systems. Instead, MS-DOS and OS/2 access the file by using the conventional eight-plus-three filename contained in the folder entry for the file.

### Example of Folder Entries for the long filename

Figure 6-4 below shows all of the folder entries for the file Thequi~1.fox, which has a long name of The quick brown.fox. The long name is in Unicode, so each character in the name uses two bytes in the folder entry. The attribute field for the long name entries has the value 0x0F. The attribute field for the short name is 0x20.

**Figure 6-4** Long File Name Folder Entry Example




***(i)*** *Note: Windows NT/2000/XP and Windows 95/98/ME use the same algorithm to create long and short filenames. On computers that dual-boot these two operating systems, files that you create when running one of the operating systems can be accessed when running the other.*

## FAT32 Features

The following topics describe the FAT32 file system.

- File System Specifications
- Boot Sector and Bootstrap Modifications
- FAT Mirroring
- Partition Types

### File System Specifications

FAT32 is a derivative of the File Allocation Table (FAT) file system that supports drives with over 2GB of storage. Because FAT32 drives can contain more than 65,526 clusters, smaller clusters are used than on large FAT16 drives. This method results in more efficient space allocation on the FAT32 drive.

The largest possible file for a FAT32 drive is 4GB minus 2 bytes.

The FAT32 file system includes four bytes per cluster within the file allocation table. Note that the high 4 bits of the 32-bit values in the FAT32 file allocation table are reserved and are not part of the cluster number.

### Boot Sector and Bootstrap Modifications

**Table 6-4** Modifications to Boot Sector

| Modifications | Description |
|---|---|
| Reserved Sectors | FAT32 drives contain more reserved sectors than FAT16 or FAT12 drives. The number of reserved sectors is usually 32, but can vary. |
| Boot Sector Modifications | Because a FAT32 BIOS Parameter Block (BPB), represented by the BPB structure, is larger than a standard BPB, the boot record on FAT32 drives is greater than 1 sector. In addition, there is a sector in the reserved area on FAT32 drives that contains values for the count of free clusters and the cluster number of the most recently allocated cluster. These values are members of the BIGFATBOOTFSINFO structure which is contained within this sector. These additional fields allow the system to initialize the values without having to read the entire file allocation table. |
| Root Directory | The root directory on a FAT32 drive is not stored in a fixed location as it is on FAT16 and FAT12 drives. On FAT32 drives, the root directory is an ordinary cluster chain. The **A_BF_BPB_RootDirStrtClus** member in the BPB structure contains the number of the first cluster in the root directory. This allows the root directory to grow as needed. In addition, the **BPB_RootEntries** member of BPB is ignored on a FAT32 drive. |
| Sectors Per FAT | The **A_BF_BPB_SectorsPerFAT** member of BPB is *always zero* on a FAT32 drive. Additionally, the **A_BF_BPB_BigSectorsPerFat** and **A_BF_BPB_BigSectorsPerFatHi** members of the updated BPB provide equivalent information for FAT32 media. |

### BPB (FAT32)

The BPB for FAT32 drives is an extended version of the FAT16/FAT12 BPB. It contains identical information to a standard BPB, but also includes several extra fields for FAT32 specific information.

This structure is implemented in Windows OEM Service Release 2 and later.

```
A_BF_BPB    STRUC
  A_BF_BPB_BytesPerSector      DW    ?
  A_BF_BPB_SectorsPerCluster   DB    ?
  A_BF_BPB_ReservedSectors     DW    ?
  A_BF_BPB_NumberOfFATs        DB    ?
  A_BF_BPB_RootEntries         DW    ?
  A_BF_BPB_TotalSectors        DW    ?
  A_BF_BPB_MediaDescriptor     DB    ?
  A_BF_BPB_SectorsPerFAT       DW    ?
```

```
  A_BF_BPB_SectorsPerTrack     DW   ?
  A_BF_BPB_Heads               DW   ?
  A_BF_BPB_HiddenSectors       DW   ?
  A_BF_BPB_HiddenSectorsHigh   DW   ?
  A_BF_BPB_BigTotalSectors     DW   ?
  A_BF_BPB_BigTotalSectorsHigh DW   ?
  A_BF_BPB_BigSectorsPerFat    DW   ?
  A_BF_BPB_BigSectorsPerFatHi  DW   ?
  A_BF_BPB_ExtFlags            DW   ?
  A_BF_BPB_FS_Version          DW   ?
  A_BF_BPB_RootDirStrtClus     DW   ?
  A_BF_BPB_RootDirStrtClusHi   DW   ?
  A_BF_BPB_FSInfoSec           DW   ?
  A_BF_BPB_BkUpBootSec         DW   ?
  A_BF_BPB_Reserved            DW   6 DUP (?)
A_BF_BPB   ENDS
```

**Table 6-5** BPB Members

| Member Name | Description |
|---|---|
| A_BF_BPB_BytesPerSector | The number of bytes per sector. |
| A_BF_BPB_SectorsPerCluster | The number of sectors per cluster. |
| A_BF_BPB_ReservedSectors | The number of reserved sectors, beginning with sector 0. |
| A_BF_BPB_NumberOfFATs | The number of File Allocation Tables. |
| A_BF_BPB_RootEntries | This member is ignored on FAT32 drives. |
| A_BF_BPB_TotalSectors | The size of the partition, in sectors. |
| A_BF_BPB_MediaDescriptor | The media descriptor. Values in this member are identical to standard BPB. |
| A_BF_BPB_SectorsPerFAT | The number of sectors per FAT. |

***(i)*** *Note: This member will always be zero in a FAT32 BPB. Use the values from A_BF_BPB_BigSectorsPerFat and A_BF_BPB_BigSectorsPerFatHi for FAT32 media.*

| | |
|---|---|
| A_BF_BPB_SectorsPerTrack | The number of sectors per track. |
| A_BF_BPB_Heads | The number of read/write heads on the drive. |
| A_BF_BPB_HiddenSectors | The number of hidden sectors on the drive. |
| A_BF_BPB_HiddenSectorsHigh | The high word of the hidden sectors value. |
| A_BF_BPB_BigTotalSectors | The total number of sectors on the FAT32 drive. |
| A_BF_BPB_BigTotalSectorsHigh | The high word of the FAT32 total sectors value. |
| A_BF_BPB_BigSectorsPerFat | The number of sectors per FAT on the FAT32 drive. |
| A_BF_BPB_BigSectorsPerFatHi | The high word of the FAT32 sectors per FAT value. |
| A_BF_BPBExtFlags | Flags describing the drive. Bit 8 of this value indicates whether or not information written to the active FAT will be written to all copies of the FAT. The low 4 bits of this value contain the 0-based FAT number of the Active FAT, but are only meaningful if bit 8 is set. This member can contain a combination of the following values. |

| Member Name | Description |
|---|---|
| Value | Description |
| BGBPB_F_ActiveFATMsk | Mask for low four bits. (000Fh) |
| BGBPB_F_NoFATMirror | Mask indicating FAT (0080h) mirroring state. If set, FAT mirroring is disabled. If clear, FAT mirroring is enabled. |
| | Bits 4-6 and 8-15 are reserved. |
| A_BF_BPB_FS_Version | The file system version number of the FAT32 drive. The high byte represents the major version, and the low byte represents the minor version. |
| A_BF_BPB_RootDirStrtClus | The cluster number of the first cluster in the FAT32 drive's root directory. |
| A_BF_BPB_RootDirStrtClusHi | The high word of the FAT32 starting cluster number. |
| A_BF_BPB_FSInfoSec | The sector number of the file system information sector. The file system info sector contains a BIGFATBOOTFSINFO structure. This member is set to 0FFFFh if there is no FSINFO sector. Otherwise, this value must be non-zero and less than the reserved sector count. |
| A_BF_BPB_BkUpBootSec | The sector number of the backup boot sector. This member is set to 0FFFFh if there is no backup boot sector. Otherwise, this value must be non-zero and less than the reserved sector count. |
| A_BF_BPB_Reserved | Reserved member. |

### BIGFATBOOTFSINFO (FAT32)

Contains information about the file system on a FAT32 volume. This structure is implemented in Windows OEM Service Release 2 and later.

```
BIGFATBOOTFSINFO STRUC
  bfFSInf_Sig             DD    ?
  bfFSInf_free_clus_cnt   DD    ?
  bfFSInf_next_free_clus  DD    ?
  bfFSInf_resvd           DD    3 DUP (?)
BIGFATBOOTFSINFO ENDS
```

**Table 6-6** BIGFATBOOTFSINFO Members

| Member Name | Description |
|---|---|
| bfFSInf_Sig | The signature of the file system information sector. The value in this member is FSINFOSIG (0x61417272L). |
| bfFSInf_free_clus_cnt | The count of free clusters on the drive. Set to -1 when the count is unknown. |
| bfFSInf_next_free_clus | The cluster number of the cluster that was most recently allocated. |
| bfFSInf_resvd | Reserved member. |

### FAT Mirroring

On all FAT drives, there may be multiple copies of the FAT. If an error occurs reading the primary copy, the file system will attempt to read from the backup copies. On FAT16 and FAT12 drives, the first FAT is always the primary copy and any modifications will automatically be written to all copies. However, on FAT32 drives, FAT mirroring can be disabled and a FAT other than the first one can be the primary (or “active”) copy of the FAT.

Mirroring is enabled by clearing bit 0x0080 in the extdpb_flags member of a FAT32 Drive Parameter Block (DPB) structure.

**Table 6-7** FAT Mirroring

| Mirroring | Description |
|---|---|
| When Enabled (bit 0x0080 clear) | With mirroring enabled, whenever a FAT sector is written, it will also be written to every other FAT. Also, a mirrored FAT sector can be read from any FAT.<br><br>A FAT32 drive with multiple FATs will behave the same as FAT16 and FAT12 drives with multiple FATs. That is, the multiple FATs are backups of each other. |
| When Disabled (bit 0x0080 set) | With mirroring disabled, only one of the FATs is active. The active FAT is the one specified by bits 0 through 3 of the **extdpb_flags** member of DPB. The other FATs are ignored.<br><br>Disabling mirroring allows better handling of a drive with a bad sector in one of the FATs. If a bad sector exists, access to the damaged FAT can be completely disabled. Then, a new FAT can be built in one of the inactive FATs and then made accessible by changing the active FAT value in **extdpb_flags**. |

### Drive Parameter Block (FAT32)

The DPB was extended to include FAT32 information. Changes are effective for Windows 95 OEM Service Release 2 and later.

```
DPB STRUC
   dpb_drive          DB    ?
   dpb_unit           DB    ?
   dpb_sector_size    DW    ?
   dpb_cluster_mask   DB    ?
   dpb_cluster_shift  DB    ?
   dpb_first_fat      DW    ?
   dpb_fat_count      DB    ?
   dpb_root_entries   DW    ?
   dpb_first_sector   DW    ?
   dpb_max_cluster    DW    ?
   dpb_fat_size       DW    ?
   dpb_dir_sector     DW    ?
   dpb_reserved2      DD    ?
   dpb_media          DB    ?
ifdef NOTFAT32
   dpb_first_access   DB    ?
else
   dpb_reserved       DB    ?
endif
   dpb_reserved3      DD    ?
   dpb_next_free      DW    ?
   dpb_free_cnt       DW    ?
ifndef NOTFAT32
   extdpb_free_cnt_hi DW    ?
   extdpb_flags       DW    ?
   extdpb_FSInfoSec   DW    ?
   extdpb_BkUpBootSec DW    ?
   extdpb_first_sector DD   ?
   extdpb_max_cluster DD    ?
   extdpb_fat_size    DD    ?
   extdpb_root_clus   DD    ?
   extdpb_next_free   DD    ?
endif
DPB ENDS
```

**Table 6-8** DBP Members

| Member Name | Description |
|---|---|
| dpb_drive | The drive number (0 = A, 1 = B, and so on). |
| dpb_unit | Specifies the unit number. The device driver uses the unit number to distinguish the specified drive from the other drives it supports. |
| dpb_sector_size | The size of each sector, in bytes. |
| dpb_cluster_mask | The number of sectors per cluster minus 1. |
| dpb_cluster_shift | The number of sectors per cluster, expressed as a power of 2. |
| dpb_first_fat | The sector number of the first sector containing the file allocation table (FAT). |
| dpb_fat_count | The number of FATs on the drive. |
| dpb_root_entries | The number of entries in the root directory. |
| dpb_first_sector | The sector number of the first sector in the first cluster. |

| Member Name | Description |
|---|---|
| dpb_max_cluster | The number of clusters on the drive plus 1. This member is undefined for FAT32 drives. |
| dpb_fat_size | The number of sectors occupied by each FAT. The value of zero indicates a FAT32 drive. Use the value in extdpb_fat_size instead. |
| dpb_dir_sector | The sector number of the first sector containing the root directory. This member is undefined for FAT32 drives. |
| dpb_reserved2 | Reserved member. Do not use. |
| dpb_media | Specifies the media descriptor for the medium in the specified drive. |
| reserved | Reserved member. Do not use. |
| dpb_first_access | Indicates whether the medium in the drive has been accessed. This member is initialized to -1 to force a media check the first time this DPB is used. |
| dpb_reserved3 | Reserved member. Do not use. |
| dpb_next_free | The cluster number of the most recently allocated cluster. |
| dpb_free_cnt | The number of free clusters on the medium. This member is 0FFFFh if the number is unknown. |
| extdpb_free_cnt_hi | The high word of free count. |
| extdpb_flags | Flags describing the drive. The low 4 bits of this value contain the 0-based FAT number of the Active FAT. This member can contain a combination of the following values. |
| Value | Description |
| BGBPB_F_ActiveFATMsk (000Fh) | Mask for low four bits. |
| BGBPB_F_NoFATMirror (0080h) | Do not mirror active FAT to inactive FATs. |
| | Bits 4-6 and 8-15 are reserved. |
| extdpb_FSInfoSec | The sector number of the file system information sector. This member is set to 0FFFFh if there is no FSINFO sector. Otherwise, this value must be non-zero and less than the reserved sector count. |
| extdpb_BkUpBootSec | The sector number of the backup boot sector. This member is set to 0FFFFh if there is no backup boot sector. Otherwise, this value must be non-zero and less than the reserved sector count. |
| extdpb_first_sector | The first sector of the first cluster. |
| extdpb_max_cluster | The number of clusters on the drive plus 1. |
| extdpb_fat_size | The number of sectors occupied by the FAT. |
| extdpb_root_clus | The cluster number of the first cluster in the root directory. |
| extdpb_next_free | The number of the cluster that was most recently allocated. |

### FAT32 Partition Types

The following table displays all valid partition types and their corresponding values for use in the **Part_FileSystem** member of the s_partition structure.

**Table 6-9** Partition Type Values

| Value | Description |
|---|---|
| PART_UNKNOWN (00h) | Unknown |

| Value | Description |
|---|---|
| PART_DOS2_FAT (01h) | 12-bit FAT |
| PART_DOS3_FAT (04h) | 16-bit FAT. Partitions smaller than 32MB. |
| PART_EXTENDED (05h) | Extended MS-DOS Partition |
| PART_DOS4_FAT (06h) | 16-bit FAT. Partitions larger than or equal to 32MB. |
| PART_DOS32 (0Bh) | 32-bit FAT. Partitions up to 2047GB. |
| PART_DOS32X (0Ch) | Same as PART_DOS32 (0Bh), but uses Logical Block Address Int 13h extensions. |
| PART_DOSX13 (0Eh) | Same as PART_DOS4_FAT (06h), but uses Logical Block Address Int 13h extensions. |
| PART_DOSX13X (0Fh) | Same as PART_EXTENDED (05h), but uses Logical Block Address Int 13h extensions. |

### s_partition (FAT32)

***(i)*** *Note: Values for head and track are 0-based. Sector values are 1-based. This structure is implemented in Windows OEM Service Release 2 and later.*

```
s_partition   STRUC
   Part_BootInd      DB   ?
   Part_FirstHead    DB   ?
   Part_FirstSector  DB   ?
   Part_FirstTrack   DB   ?
   Part_FileSystem   DB   ?
   Part_LastHead     DB   ?
   Part_LastSector   DB   ?
   Part_LastTrack    DB   ?
   Part_StartSector  DD   ?
   Part_NumSectors   DD   ?
s_partition   ENDS
```

**Table 6-10** s_partition Members

| Member Name | Description |
|---|---|
| Part_BootInd | Specifies whether the partition is bootable or not. This value could be set to PART_BOOTABLE (80h), or PART_NON_BOOTABLE(00h). The first partition designated as PART_BOOTABLE is the boot partition. All others are not. Setting multiple partitions to PART_BOOTABLE will result in boot errors. |
| Part_FirstHead | The first head of this partition. This is a 0-based number representing the offset from the beginning of the disk. The partition includes this head. |
| Part_FirstSector | The first sector of this partition. This is a 1-based, 6-bit number representing the offset from the beginning of the disk. The partition includes this sector. Bits 0 through 5 specify the 6-bit value; bits 6 and 7 are used with the Part_FirstTrack member. |
| Part_FirstTrack | The first track of this partition. This is an inclusive 0-based, 10-bit number that represents the offset from the beginning of the disk. The high 2 bits of this value are specified by bits 6 and 7 of the Part_FirstSector member. |

| Member Name | Description |
|---|---|
| PartFileSystem | Specifies the file system for the partition. The following are acceptable values: |
| Value | Description |
| PART_UNKNOWN(00h) | Unknown. |
| PART_DOS2_FAT(01h) | 12-bit FAT. |
| PART_DOS3_FAT(04h) | 16-bit FAT. Partition smaller than 32MB. |
| PART_EXTENDED(05h) | Extended MS-DOS Partition. |
| PART_DOS4_FAT(06h) | 16-bit FAT. Partition larger than or equal to 32MB. |
| PART_DOS32(0Bh) | 32-bit FAT. Partition up to 2047GB. |
| PART_DOS32X(0Ch) | Same as PART_DOS32(0Bh), but uses Logical Block Address **Int 13h** extensions. |
| PART_DOSX13(0Eh) | Same as PART_DOS4_FAT(06h), but uses Logical Block Address **Int 13h** extensions. |
| PART_DOSX13X(0Fh) | Same as PART_EXTENDED(05h), but uses Logical Block Address **Int 13h** extensions. |
| Part_LastHead | The last head of the partition. This is a 0-based number that represents the offset from the beginning of the disk. The partition includes the head specified by this member. |
| Part_LastSector | The last sector of this partition. This is a 1-based, 6-bit number representing offset from the beginning of the disk. The partition includes the sector specified by this member. Bits 0 through 5 specify the 6-bit value; bits 6 and 7 are used with the **Part_LastTrack** member. |
| Part_LastTrack | The last track of this partition. This is a 0-based, 10-bit number that represents offset from the beginning of the disk. The partition includes this track. The high 2 bits of this value are specified by bits 6 and 7 of the **Part_LastSector** member. |
| Part_StartSector | Specifies the 1-based number of the first sector on the disk. This value may not be accurate for extended partitions. Use the **Part_FirstSector** value for extended partitions. |
| Part_NumSectors | The 1-based number of sectors in the partition. |

## The NTFS File System

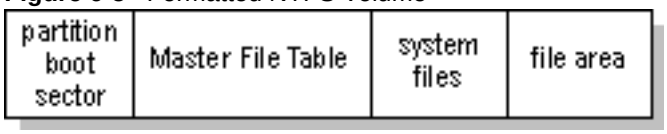
The Windows NT file system (NTFS) provides a combination of performance, reliability, and compatibility not found in the FAT file system. It is designed to quickly perform standard file operations such as read, write, and search — and even advanced operations such as file-system recovery — on very large hard disks.

Formatting a volume with the NTFS file system results in the creation of several system files and the Master File Table (MFT), which contains information about all the files and folders on the NTFS volume.

The first information on an NTFS volume is the Partition Boot Sector, which starts at sector 0 and can be up to 16 sectors long. The first file on an NTFS volume is the Master File Table (MFT).

The following figure illustrates the layout of an NTFS volume when formatting has finished.

**Figure 6-5** Formatted NTFS Volume




This chapter covers information about NTFS. Topics covered are listed below:

- NTFS Partition Boot Sector
- NTFS Master File Table (MFT)
- NTFS File Types
- NTFS Data Integrity and Recoverability

The NTFS file system includes security features required for file servers and high-end personal computers in a corporate environment. The NTFS file system also supports data access control and ownership privileges that are important for the integrity of critical data. While folders shared on a Windows NT computer are assigned particular permissions, NTFS files and folders can have permissions assigned whether they are shared or not. NTFS is the only file system on Windows NT that allows you to assign permissions to individual files.

The NTFS file system has a simple, yet very powerful design. Basically, everything on the volume is a file and everything in a file is an attribute, from the data attribute, to the security attribute, to the file name attribute. Every sector on an NTFS volume that is allocated belongs to some file. Even the file system metadata (information that describes the file system itself) is part of a file.

### What's New in NTFS5 (Windows 2000)

***Encryption*** The Encrypting File System (EFS) provides the core file encryption technology used to store encrypted files on NTFS volumes. EFS keeps files safe from intruders who might gain unauthorized physical access to sensitive, stored data (for example, by stealing a portable computer or external disk drive).

***Disk Quotas*** Windows 2000 supports disk quotas for NTFS volumes. You can use disk quotas to monitor and limit disk-space use.

***Reparse Points*** Reparse points are new file system objects in NTFS that can be applied to NTFS files or folders. A file or folder that contains a reparse point acquires additional behavior not present in the underlying file system. Reparse points are used by many of the new storage features in Windows 2000, including volume mount points.

***Volume Mount Points*** Volume mount points are new to NTFS. Based on reparse points, volume mount points allow administrators to graft access to the root of one local volume onto the folder structure of another local volume.

***Sparse Files*** Sparse files allow programs to create very large files but consume disk space only as needed.

***Distributed Link Tracking*** NTFS provides a link-tracking service that maintains the integrity of shortcuts to files as well as OLE links within compound documents.

For more detailed information see resource kits on Microsoft's web site http://www.microsoft.com/windows/reskits/webresources/default.asp or Microsoft Developers Network (MSDN) http://msdn.microsoft.com

## NTFS Partition Boot Sector

Table 6-11 describes the boot sector of a volume formatted with NTFS. When you format an NTFS volume, the format program allocates the first 16 sectors for the boot sector and the bootstrap code.

**Table 6-11** NTFS Boot Sector

| Byte Offset | Field Length | Field Name |
|---|---|---|
| 0x00 | 3 bytes | Jump Instruction |
| 0x03 | LONGLONG | OEM ID |
| 0x0B | 25 bytes | BPB |
| 0x24 | 48 bytes | Extended BPB |
| 0x54 | 426 bytes | Bootstrap Code |
| 0x01FE | WORD | End of Sector Marker |

On NTFS volumes, the data fields that follow the BPB form an extended BPB. The data in these fields enables Ntldr (NT loader program) to find the master file table (MFT) during startup. On NTFS volumes, the MFT is not located in a predefined sector, as on FAT16 and FAT32 volumes. For this reason, the MFT can be moved if there is a bad sector in its normal location. However, if the data is corrupted, the MFT cannot be located, and Windows NT/2000 assumes that the volume has not been formatted.

The following example illustrates the boot sector of an NTFS volume formatted while running Windows 2000. The printout is formatted in three sections:

- Bytes 0x00– 0x0A are the jump instruction and the OEM ID (shown in bold print).

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

```
Physical Sector: Cyl 0, Side 1, Sector 1
00000000: EB 52 90 4E 54 46 53 20 - 20 20 20 00 02
08 00 00 .R.NTFS ........ 00000010: 00 00 00 00 00
F8 00 00 - 3F 00 FF 00 3F 00 00 00 ........?...?...
00000020: 00 00 00 00 80 00 80 00 - 4A F5 7F 00 00
00 00 00 ........J....... 00000030: 04 00 00 00 00
00 00 00 - 54 FF 07 00 00 00 00 00 ........T.......
00000040: F6 00 00 00 01 00 00 00 - 14 A5 1B 74 C9
1B 74 1C ..........t..t. 00000050: 00 00 00 00 FA
33 C0 8E - D0 BC 00 7C FB B8 C0 07 .....3.....|....
00000060: 8E D8 E8 16 00 B8 00 0D - 8E C0 33 DB C6
06 0E 00 ..........3..... 00000070: 10 E8 53 00 68
00 0D 68 - 6A 02 CB 8A 16 24 00 B4 ..S.h..hj....$..
00000080: 08 CD 13 73 05 B9 FF FF - 8A F1 66 0F B6
C6 40 66 ...s......f...@f 00000090: 0F B6 D1 80 E2
3F F7 E2 - 86 CD C0 ED 06 41 66 0F .....?.......Af.
000000A0: B7 C9 66 F7 E1 66 A3 20 - 00 C3 B4 41 BB
AA 55 8A ..f..f. ...A..U. 000000B0: 16 24 00 CD 13
72 0F 81 - FB 55 AA 75 09 F6 C1 01 .$...r...U.u....
000000C0: 74 04 FE 06 14 00 C3 66 - 60 1E 06 66 A1
10 00 66 t......f`..f...f 000000D0: 03 06 1C 00 66
3B 06 20 - 00 0F 82 3A 00 1E 66 6A ....f;. ...:..fj
000000E0: 00 66 50 06 53 66 68 10 - 00 01 00 80 3E
14 00 00 .fP.Sfh.....>... 000000F0: 0F 85 0C 00 E8
B3 FF 80 - 3E 14 00 00 0F 84 61 00 ........>.....a.
00000100: B4 42 8A 16 24 00 16 1F - 8B F4 CD 13 66
58 5B 07 .B..$......fX[.. 00000110: 66 58 66 58 1F
EB 2D 66 - 33 D2 66 0F B7 0E 18 00 fXfX.-f3.f......
00000120: 66 F7 F1 FE C2 8A CA 66 - 8B D0 66 C1 EA
10 F7 36 f......f..f....6 00000130: 1A 00 86 D6 8A
16 24 00 - 8A E8 C0 E4 06 0A CC B8 ......$.........
00000140: 01 02 CD 13 0F 82 19 00 - 8C C0 05 20 00
8E C0 66 .......... ...f 00000150: FF 06 10 00 FF
0E 0E 00 - 0F 85 6F FF 07 1F 66 61 ..........o...fa
00000160: C3 A0 F8 01 E8 09 00 A0 - FB 01 E8 03 00
FB EB FE ............... 00000170: B4 01 8B F0 AC
3C 00 74 - 09 B4 0E BB 07 00 CD 10 .....<.t........
00000180: EB F2 C3 0D 0A 41 20 64 - 69 73 6B 20 72
65 61 64 .....A disk read 00000190: 20 65 72 72 6F
72 20 6F - 63 63 75 72 72 65 64 00  error occurred.
000001A0: 0D 0A 4E 54 4C 44 52 20 - 69 73 20 6D 69
73 73 69 ..NTLDR is missi 000001B0: 6E 67 00 0D 0A
4E 54 4C - 44 52 20 69 73 20 63 6F ng...NTLDR is co
```

```
000001C0: 6D 70 72 65 73 73 65 64 - 00 0D 0A 50 72
65 73 73 mpressed...Press 000001D0: 20 43 74 72 6C
2B 41 6C - 74 2B 44 65 6C 20 74 6F  Ctrl+Alt+Del to
000001E0: 20 72 65 73 74 61 72 74 - 0D 0A 00 00 00
00 00 00  restart........ 000001F0: 00 00 00 00 00
00 00 00 - 83 A0 B3 C9 00 00 55 AA ..............U.
```

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

**Table 6-12** BPB Fields on NTFS

| Byte Offset | Field Length | Sample Value | Field Name |
|---|---|---|---|
| 0x0B | WORD | 0x0002 | Bytes Per Sector |
| 0x0D | BYTE | 0x08 | Sectors Per Cluster |
| 0x0E | WORD | 0x0000 | Reserved Sectors |
| 0x10 | 3 BYTES | 0x000000 | always 0 |
| 0x13 | WORD | 0x0000 | not used by NTFS |
| 0x15 | BYTE | 0xF8 | Media Descriptor |
| 0x16 | WORD | 0x0000 | always 0 |
| 0x18 | WORD | 0x3F00 | Sectors Per Track |
| 0x1A | WORD | 0xFF00 | Number Of Heads |
| 0x1C | DWORD | 0x3F000000 | Hidden Sectors |
| 0x20 | DWORD | 0x00000000 | not used by NTFS |
| 0x24 | DWORD | 0x80008000 | not used by NTFS |
| 0x28 | LONGLONG | 0x4AF57F0000000000 | Total Sectors |
| 0x30 | LONGLONG | 0x0400000000000000 | Logical Cluster Number for the file $MFT |
| 0x38 | LONGLONG | 0x54FF070000000000 | Logical Cluster Number for the file $MFTMirr |
| 0x40 | DWORD | 0xF6000000 | Clusters Per File Record Segment |
| 0x44 | DWORD | 0x01000000 | Clusters Per Index Block |
| 0x48 | LONGLONG | 0x14A51B74C91B741C | Volume Serial Number |
| 0x50 | DWORD | 0x00000000 | Checksum |

### Protecting the Boot Sector

Because a normally functioning system relies on the boot sector to access a volume, it is highly recommended that you run disk scanning tools such as Chkdsk regularly, as well as back up all of your data files to protect against data loss if you lose access to a volume.

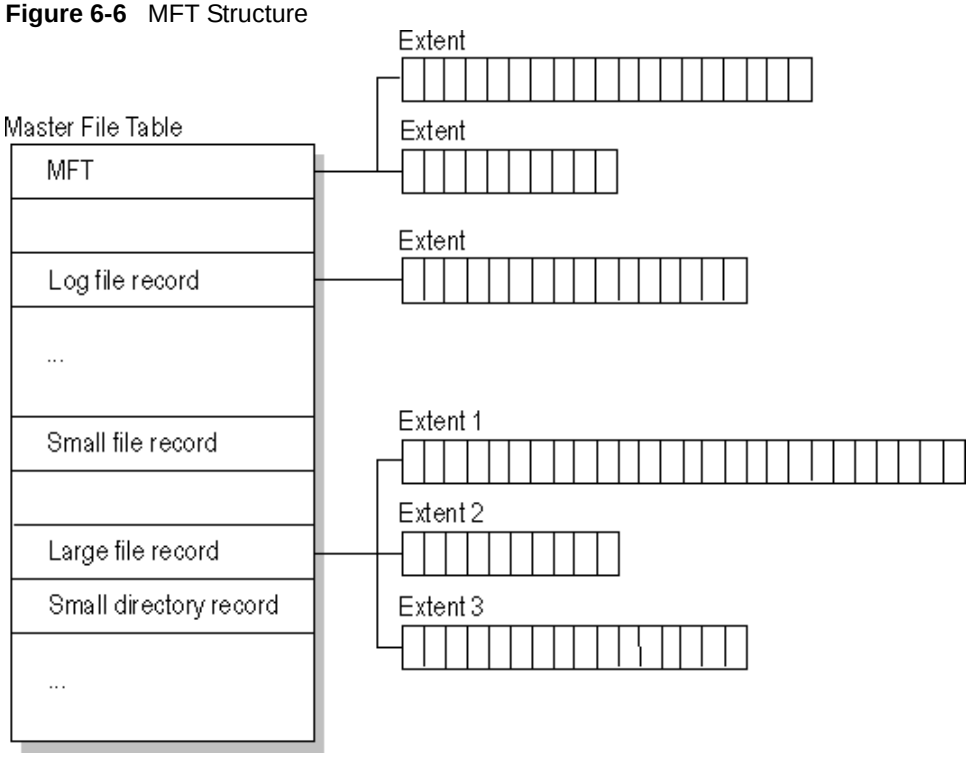
## NTFS Master File Table (MFT)

Each file on an NTFS volume is represented by a record in a special file called the master file table (MFT). NTFS reserves the first 16 records of the table for special information. The first record of this table describes the master file table itself, followed by a MFT mirror record. If the first MFT record is corrupted, NTFS reads the second record to find the MFT mirror file, whose first record is identical to the first record of the MFT. The locations of the data segments for both the MFT and MFT mirror file are recorded in the boot sector. A duplicate of the boot sector is located at the logical center of the disk.

The third record of the MFT is the log file, used for file recovery. The log file is discussed in detail later in this chapter. The seventeenth and following records of the master file table are for each file and directory (also viewed as a file by NTFS) on the volume.

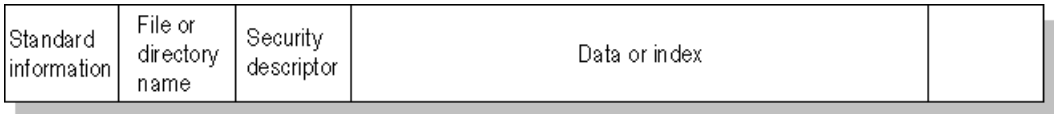
Figure provides a simplified illustration of the MFT structure:

**Figure 6-6** MFT Structure




The master file table allocates a certain amount of space for each file record. The attributes of a file are written to the allocated space in the MFT. Small files and directories (typically 1500 bytes or smaller), such as the file illustrated in next figure, can entirely be contained within the master file table record.

**Figure 6-7** MFT Record for a Small File or Directory




This design makes file access very fast. Consider, for example, the FAT file system, which uses a file allocation table to list the names and addresses of

each file. FAT directory entries contain an index into the file allocation table. When you want to view a file, FAT first reads the file allocation table and assures that it exists. Then FAT retrieves the file by searching the chain of allocation units assigned to the file. With NTFS, as soon as you look up the file, it's there for you to use.

Directory records are housed within the master file table just like file records. Instead of data, directories contain index information. Small directory records reside entirely within the MFT structure. Large directories are organized into B-trees, having records with pointers to external clusters containing directory entries that could not be contained within the MFT structure.

## NTFS File Types

This section covers the following topics:

- NTFS File Attributes
- NTFS System Files
- NTFS Multiple Data Streams
- NTFS Compressed Files
- NTFS Encrypted Files
- NTFS Sparse Files

### NTFS File Attributes

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

When a file's attributes can fit within the MFT file record, they are called resident attributes. For example, information such as filename and time stamp are always included in the MFT file record. When all of the information for a file is too large to fit in the MFT file record, some of its attributes are nonresident. The nonresident attributes are allocated one or more clusters of disk space elsewhere in the volume. NTFS creates the Attribute List attribute to describe the location of all of the attribute records.

Table 6-13 lists all of the file attributes currently defined by the NTFS file system. This list is extensible, meaning that other file attributes can be defined in the future.

**Table 6-13** File Attributes Defined by NTFS

| Attribute Type | Description |
|---|---|
| Standard Information | Includes information such as timestamp and link count. |
| Attribute List | Lists the location of all attribute records that do not fit in the MFT record. |
| File Name | A repeatable attribute for both long and short file names. The long name of the file can be up to 255 Unicode characters. The short name is the 8.3, case-insensitive name for the file. Additional names, or hard links, required by POSIX can be included as additional file name attributes. |
| Security Descriptor | Describes who owns the file and who can access it. |

| Attribute Type | Description |
|---|---|
| Data | Contains file data. NTFS allows multiple data attributes per file. Each file typically has one unnamed data attribute. A file can also have one or more named data attributes, each using a particular syntax. |
| Object ID | A volume-unique file identifier. Used by the distributed link tracking service. Not all files have object identifiers. |
| Logged Tool Stream | Similar to a data stream, but operations are logged to the NTFS log file just like NTFS metadata changes. This is used by EFS. |
| Reparse Point | Used for volume mount points. They are also used by Installable File System (IFS) filter drivers to mark certain files as special to that driver. |
| Index Root | Used to implement folders and other indexes. |
| Index Allocation | Used to implement folders and other indexes. |
| Bitmap | Used to implement folders and other indexes. |
| Volume Information | Used only in the $Volume system file. Contains the volume version. |
| Volume Name | Used only in the $Volume system file. Contains the volume label. |

### NTFS System Files

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

**Table 6-14** Metadata Stored in the Master File Table

| System File | File Name | MFT Record | Purpose of the File |
|---|---|---|---|
| Master file table | $Mft | 0 | Contains one base file record for each file and folder on an NTFS volume. If the allocation information for a file or folder is too large to fit within a single record, other file records are allocated as well. |
| Master file table 2 | $MftMirr | 1 | A duplicate image of the first four records of the MFT. This file guarantees access to the MFT in case of a single-sector failure. |
| Log file | $Log file | 2 | Contains a list of transaction steps used for NTFS recoverability. Log file size depends on the volume size and can be as large as 4 MB. It is used by Windows NT/2000 to restore consistency to NTFS after a system failure. |

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

### NTFS Multiple Data Streams

NTFS supports multiple data streams, where the stream name identifies a new data attribute on the file. A handle can be opened to each data stream. A data stream, then, is a unique set of file attributes. Streams have separate opportunistic locks, file locks, and sizes, but common permissions.

This feature enables you to manage data as a single unit. The following is an example of an alternate stream:

**myfile.dat:stream2**

A library of files might exist where the files are defined as alternate streams, as in the following example:

**library:file1**
**:file2**
**:file3**

A file can be associated with more than one application at a time, such as Microsoft® Word and Microsoft® WordPad. For instance, a file structure like the following illustrates file association, but not multiple files:

**program:source_file**
**:doc_file**
**:object_file**
**:executable_file**

To create an alternate data stream, at the command prompt, you can type commands such as:

```
echo text>program:source_file
more <program:source_file
```

***(!)*** *Important: When you copy an NTFS file to a FAT volume, such as a floppy disk, data streams and other attributes not supported by FAT are lost.*

### NTFS Compressed Files

Windows NT/2000 supports compression on individual files, folders, and entire NTFS volumes. Files compressed on an NTFS volume can be read and written by any Windows-based application without first being decompressed by another program. Decompression occurs automatically when the file is read. The file is compressed again when it is closed or saved. Compressed files and folders have an attribute of C when viewed in Windows Explorer.

Only NTFS can read the compressed form of the data. When an application such as Microsoft® Word or an operating system command such as **copy** requests access to the file, the compression filter driver decompresses the file before making it available. For example, if you copy a compressed file from another Windows NT/2000–based computer to a compressed folder on your hard disk, the file is decompressed when read, copied, and then recompressed when saved.

This compression algorithm is similar to that used by the Windows 98 application DriveSpace 3, with one important difference — the limited functionality compresses the entire primary volume or logical volume. NTFS allows for the compression of an entire volume, of one or more folders within a volume, or even one or more files within a folder of an NTFS volume.

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

Each NTFS data stream contains information that indicates whether any part of the stream is compressed. Individual compressed buffers are identified by “holes” following them in the information stored for that stream. If there is a hole, NTFS automatically decompresses the preceding buffer to fill the hole.

NTFS provides real-time access to a compressed file, decompressing the file when it is opened and compressing it when it is closed. When writing a compressed file, the system reserves disk space for the uncompressed size. The system gets back unused space as each individual compression buffer is compressed.

### NTFS Encrypted Files (Windows 2000 only)

The Encrypting File System (EFS) provides the core file encryption technology used to store encrypted files on NTFS volumes. EFS keeps files safe from intruders who might gain unauthorized physical access to sensitive, stored data (for example, by stealing a portable computer or external disk drive).

EFS uses symmetric key encryption in conjunction with public key technology to protect files and ensure that only the owner of a file can access it. Users of EFS are issued a digital certificate with a public key and a private key pair. EFS

uses the key set for the user who is logged on to the local computer where the private key is stored.

Users work with encrypted files and folders just as they do with any other files and folders. Encryption is transparent to the user who encrypted the file; the system automatically decrypts the file or folder when the user accesses. When the file is saved, encryption is reapplied. However, intruders who try to access the encrypted files or folders receive an “Access denied” message if they try to open, copy, move, or rename the encrypted file or folder.

To encrypt or decrypt a folder or file, set the encryption attribute for folders and files just as you set any other attribute. If you encrypt a folder, all files and subfolders created in the encrypted folder are automatically encrypted. It is recommended that you encrypt at the folder level.

**NTFS Sparse Files (Windows 2000 only)**

A sparse file has an attribute that causes the I/O subsystem to allocate only meaningful (nonzero) data. Nonzero data is allocated on disk, and non-meaningful data (large strings of data composed of zeros) is not. When a sparse file is read, allocated data is returned as it was stored; non-allocated data is returned, by default, as zeros.

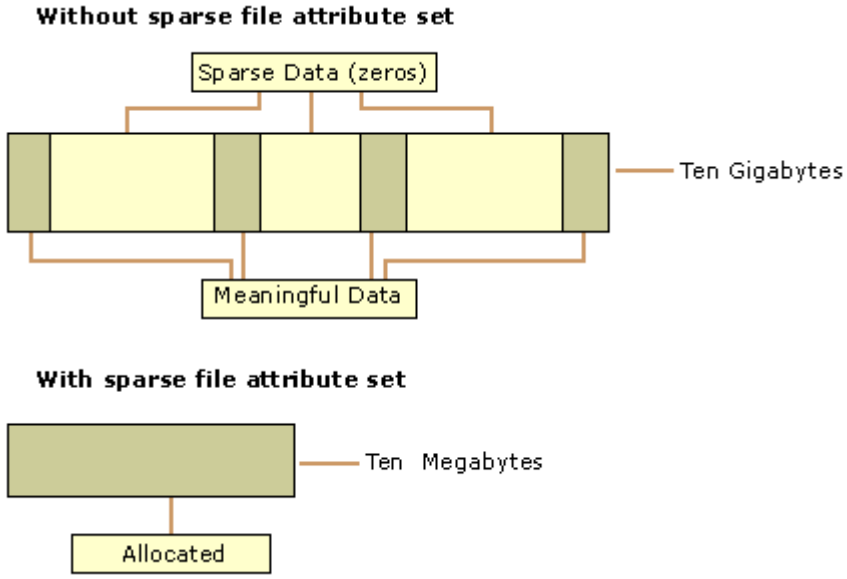
NTFS deallocates sparse data streams and only maintains other data as allocated. When a program accesses a sparse file, the file system yields allocated data as actual data and deallocated data as zeros.

NTFS includes full sparse file support for both compressed and uncompressed files. NTFS handles read operations on sparse files by returning allocated data and sparse data. It is possible to read a sparse file as allocated data and a range of data without retrieving the entire data set, although NTFS returns the entire data set by default.

With the sparse file attribute set, the file system can deallocate data from anywhere in the file and, when an application calls, yield the zero data by range instead of storing and returning the actual data. File system application programming interfaces (APIs) allow for the file to be copied or backed as actual bits and sparse stream ranges. The net result is efficient file system

storage and access. Next figure shows how data is stored with and without the sparse file attribute set.

**Figure 6-8** Windows 2000 Data Storage




***(!)*** *Important: If you copy or move a sparse file to a FAT or a non-Windows 2000 NTFS volume, the file is built to its originally specified size. If the required space is not available, the operation does not complete.*

### Data Integrity and Recoverability with NTFS

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

NTFS ensures the integrity of all NTFS volumes by automatically performing disk recovery operations the first time a program accesses an NTFS volume after the computer is restarted following a failure.

NTFS also uses a technique called cluster remapping to minimize the effects of a bad sector on an NTFS volume.

***(!)*** *Important: If either the master boot record (MBR) or boot sector is corrupted, you might not be able to access data on the volume.*

### Recovering Data with NTFS

NTFS views each I/O operation that modifies a system file on the NTFS volume as a transaction, and manages each one as an integral unit. Once started, the transaction is either completed or, in the event of a disk failure, rolled back

(such as when the NTFS volume is returned to the state it was in before the transaction was initiated).

To ensure that a transaction can be completed or rolled back, NTFS records the suboperations of a transaction in a log file before they are written to the disk. When a complete transaction is recorded in the log file, NTFS performs the suboperations of the transaction on the volume cache. After NTFS updates the cache, it commits the transaction by recording in the log file that the entire transaction is complete.

Once a transaction is committed, NTFS ensures that the entire transaction appears on the volume, even if the disk fails. During recovery operations, NTFS redoes each committed transaction found in the log file. Then NTFS locates the transactions in the log file that were not committed at the time of the system failure and undoes each transaction suboperation recorded in the log file. Incomplete modifications to the volume are prohibited.

NTFS uses the Log File service to log all redo and undo information for a transaction. NTFS uses the redo information to repeat the transaction. The undo information enables NTFS to undo transactions that are not complete or that have an error.

***(!)*** *Important: NTFS uses transaction logging and recovery to guarantee that the volume structure is not corrupted. For this reason, all system files remain accessible after a system failure. However, user data can be lost because of a system failure or a bad sector.*

### Cluster Remapping

In the event of a bad-sector error, NTFS implements a recovery technique called cluster remapping. When Windows 2000 detects a bad-sector, NTFS dynamically remaps the cluster containing the bad sector and allocates a new cluster for the data. If the error occurred during a read, NTFS returns a read error to the calling program, and the data is lost. If the error occurs during a write, NTFS writes the data to the new cluster, and no data is lost.

NTFS puts the address of the cluster containing the bad sector in its bad cluster file so the bad sector is not reused.

***(!)*** *Important: Cluster remapping is not a backup alternative. Once errors are detected, the disk should be monitored closely and replaced if the defect list grows. This type of error is displayed in the Event Log.*

## The File Recovery Process

The file recovery process can be briefly described as drive or folder scanning to find deleted entries in Root Folder (FAT) or Master File Table (NTFS) then for the particular deleted entry, defining clusters chain to be recovered and then copying contents of these clusters to the newly created file.

Different file systems maintain their own specific logical data structures, however basically each file system:

- Has a list or catalog of file entries, so we can iterate through this list and entries, marked as deleted
- Keeps for each entry a list of data clusters, so we can try to find out set of clusters composing the file

After finding out the proper file entry and assembling set of clusters, composing the file, read and copy these clusters to another location.

Step by Step with examples:

- Disk Scanning
- Defining the Chain of Clusters
- Recovering the Chain of Clusters

Not every deleted file can be recovered, however there are some assumptions that are common to all deleted files:

- First, we assume that the file entry still exists (it has not been overwritten with other data). The fewer files that have been created on the drive where the deleted file was resided, increases the chances that space for the deleted file entry has not been used for other entries.
- Second, we assume that the file entry is more-or-less safe to point to the proper place where file clusters are located. In some cases (it has been noticed in Windows XP, on large FAT32 volumes) the operating system damages file entries right after deletion so that the first data cluster becomes invalid and further entry restoration is not possible.
- Third, we assume that the file data clusters are safe (not overwritten with other data). The fewer write operations events on the drive where deleted file resided, the more chances that the space occupied by data clusters of the deleted file has not been used for other data storage.

**General Advice After Data Loss**

1 DO NOT WRITE ANYTHING ONTO THE DRIVE CONTAINING YOUR IMPORTANT DATA THAT YOU HAVE JUST DELETED ACCIDENTALLY!

Even data recovery software installation can spoil your sensitive data. If the data is really important to you and you do not have another logical drive to install software to, take the whole hard drive out of the computer and plug it into another computer where data recovery software has been already installed or use recovery software that does not require installation, for example recovery software which is capable to run from bootable floppy.

2 DO NOT TRY TO SAVE ONTO THE SAME DRIVE DATA THAT YOU FOUND AND TRYING TO RECOVER!

When saving recovered data onto the same drive where sensitive data is located, you can intrude in process of recovering by overwriting FAT/MFT records for this and other deleted entries. It is better to save data onto another logical, removable, network or floppy drive.

## Disk Scanning for Deleted Entries

Disk Scanning is a process of low-level enumeration of all entries in the Root Folders on FAT12, FAT16, FAT32 or in Master File Table (MFT) on NTFS, NTFS5. The goal is to find and display deleted entries.

In spite of different file/folder entry structure for the different file systems, all of them contain basic file attributes like name, size, creation and modification date/time, file attributes, existing/deleted status, etc....

Given that a drive contains root file table and any file table (MFT, root folder of the drive, regular folder, or even deleted folder) has location, size and predefined structure, we can scan it from the beginning to the end checking each entry, if it's deleted or not and then display information for all found deleted entries.

***(i)*** *Note: Deleted entries are marked differently depending on the file system. For example, in FAT any deleted entry, file or folder has been marked with ASCII symbol 229 (OxE5) that becomes first symbol of the structure entry. On NTFS deleted entry has a special attribute in file header that points whether the file has been deleted or not.*

Example of scanning a folder on FAT16:

1 Existing folder MyFolder entry (long entry and short entry)

```
0003EE20   41 4D 00 79 00 46 00 6F  00 6C 00 0F 00 09 64 00   AM.y.F.o.l....d.
0003EE30   65 00 72 00 00 00 FF FF  FF FF 00 00 FF FF FF FF   e.r...yyyy..yyyy
0003EE40   4D 59 46 4F 4C 44 45 52  20 20 20 10 00 4A C4 93   MYFOLDER   ..JA“
0003EE50   56 2B 56 2B 00 00 C5 93  56 2B 02 00 00 00 00 00   V+V+..A“V+......
```

2 Deleted file MyFile.txt entry (long entry and short entry)

```
0003EE60   E5 4D 00 79 00 46 00 69  00 6C 00 0F 00 BA 65 00   aM.y.F.i.l...?e.
0003EE70   2E 00 74 00 78 00 74 00  00 00 00 00 FF FF FF FF   ..t.x.t.....yyyy
0003EE80   E5 59 46 49 4C 45 20 20  54 58 54 20 00 C3 D6 93   aYFILE  TXT .AO“
0003EE90   56 2B 56 2B 00 00 EE 93  56 2B 03 00 33 B7 01 00   V+V+..i“V+..3·..
```

3 Existing file Setuplog.txt entry (the only short entry)

```
0003EEA0   53 45 54 55 50 4C 4F 47  54 58 54 20 18 8C F7 93   SETUPLOGTXT .??“
0003EEB0   56 2B 56 2B 00 00 03 14  47 2B 07 00 8D 33 03 00   V+V+....G+..?3..
0003EEC0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
0003EED0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................

Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
```

This folder contains 3 entries, one of them is deleted. First entry is an existing folder MyFolder. Second one is a deleted file MyFile.txt Third one is an existing file Setuplog.txt.

First symbol of the deleted file entry is marked with E5 symbol, so Disk Scanner can assume that this entry has been deleted.

Example of scanning folder on NTFS5 (Windows 2000):

For our drive we have input parameters:

- Total Sectors 610406
- Cluster size 512 bytes
- One Sector per Cluster
- MFT starts from offset 0x4000, non-fragmented
- MFT record size 1024 bytes
- MFT Size 1968 records

Thus we can iterate through all 1968 MFT records, starting from the absolute offset 0x4000 on the volume looking for the deleted entries. We are interested in MFT entry 57 having offset 0x4000 + 57 * 1024 = 74752 = 0x12400 because it contains our recently deleted file “My Presentation.ppt”

Below MFT record number 57 is displayed:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00012400   46 49 4C 45 2A 00 03 00  9C 74 21 03 00 00 00 00   FILE*...?t!.....
00012410   47 00 02 00 30 00 00 00  D8 01 00 00 00 04 00 00   G...0...O.......
00012420   00 00 00 00 00 00 00 00  05 00 03 00 00 00 00 00   ................
00012430   10 00 00 00 60 00 00 00  00 00 00 00 00 00 00 00   ....`...........
00012440   48 00 00 00 18 00 00 00  20 53 DD A3 18 F1 C1 01   H....... SY?.nA.
00012450   00 30 2B D8 48 E9 C0 01  C0 BF 20 A0 18 F1 C1 01   .0+OHeA.A?  .nA.
00012460   20 53 DD A3 18 F1 C1 01  20 00 00 00 00 00 00 00    SY?.nA. .......
00012470   00 00 00 00 00 00 00 00  00 00 00 00 02 01 00 00   ................
00012480   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00012490   30 00 00 00 78 00 00 00  00 00 00 00 00 00 03 00   0...x...........
000124A0   5A 00 00 00 18 00 01 00  05 00 00 00 00 00 05 00   Z...............
000124B0   20 53 DD A3 18 F1 C1 01  20 53 DD A3 18 F1 C1 01    SY?.nA. SY?.nA.
000124C0   20 53 DD A3 18 F1 C1 01  20 53 DD A3 18 F1 C1 01    SY?.nA. SY?.nA.
000124D0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000124E0   20 00 00 00 00 00 00 00  0C 02 4D 00 59 00 50 00    .........M.Y.P.
000124F0   52 00 45 00 53 00 7E 00  31 00 2E 00 50 00 50 00   R.E.S.~.1...P.P.
00012500   54 00 69 00 6F 00 6E 00  30 00 00 00 80 00 00 00   T.i.o.n.0...ˆ...
00012510   00 00 00 00 00 00 02 00  68 00 00 00 18 00 01 00   ........h.......
00012520   05 00 00 00 00 00 05 00  20 53 DD A3 18 F1 C1 01   ........ SY?.nA.
00012530   20 53 DD A3 18 F1 C1 01  20 53 DD A3 18 F1 C1 01    SY?.nA. SY?.nA.
00012540   20 53 DD A3 18 F1 C1 01  00 00 00 00 00 00 00 00    SY?.nA.........
00012550   00 00 00 00 00 00 00 00  20 00 00 00 00 00 00 00   ........ .......
00012560   13 01 4D 00 79 00 20 00  50 00 72 00 65 00 73 00   ..M.y. .P.r.e.s.
00012570   65 00 6E 00 74 00 61 00  74 00 69 00 6F 00 6E 00   e.n.t.a.t.i.o.n.
00012580   2E 00 70 00 70 00 74 00  80 00 00 00 48 00 00 00   ..p.p.t.ˆ...H...
00012590   01 00 00 00 00 00 04 00  00 00 00 00 00 00 00 00   ................
000125A0   6D 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00   m.......@.......
000125B0   00 DC 00 00 00 00 00 00  00 DC 00 00 00 00 00 00   .U.......U......
000125C0   00 DC 00 00 00 00 00 00  31 6E EB C4 04 00 00 00   .U......1neA....
000125D0   FF FF FF FF 82 79 47 11  00 00 00 00 00 00 00 00   yyyy,yG.........
000125E0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000125F0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 03 00   ................
................
00012600   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

MFT Record has pre-defined structure. It has a set of attributes defining any file of folder parameters.

MFT Record begins with standard File Record Header (first bold section, offset 0x00):

- “FILE” identifier (4 bytes)

- Offset to update sequence (2 bytes)
- Size of update sequence (2 bytes)
- $LogFile Sequence Number (LSN) (8 bytes)
- Sequence Number (2 bytes)
- Reference Count (2 bytes)
- Offset to Update Sequence Array (2 bytes)
- Flags (2 bytes)
- Real size of the FILE record (4 bytes)
- Allocated size of the FILE record (4 bytes)
- File reference to the base FILE record (8 bytes)
- Next Attribute Id (2 bytes)

The most important information for us in this block is a file state: deleted or in-use. If Flags (in red color) field has bit 1 set, it means that file is in-use. In our example it is zero, i.e. file is deleted.

Starting from 0x48, we have **Standard Information Attribute** (second bold section):

- File Creation Time (8 bytes)
- File Last Modification Time (8 bytes)
- File Last Modification Time for File Record (8 bytes)
- File Access Time for File Record (8 bytes)
- DOS File Permissions (4 bytes) 0x20 in our case Archive Attribute

Following standard attribute header, we have **File Name Attribute** belonging to DOS name space, short file names, (third bold section, offset 0xA8) and again following standard attribute header, we have **File Name Attribute** belonging to Win32 name space, long file names, (third bold section, offset 0x120):

- File Reference to the Parent Directory (8 bytes)
- File Modification Times (32 bytes)
- Allocated Size of the File (8 bytes)
- Real Size of the File (8 bytes)
- Flags (8 bytes)
- Length of File Name (1 byte)
- File Name Space (1 byte)
- File Name (Length of File Name * 2 bytes)

In our case from this section we can extract file name, “My Presentation.ppt”, File Creation and Modification times, and Parent Directory Record number.

Starting from offset 0x188, there is a non-resident Data attribute (green section).

- Attribute Type (4 bytes) (e.g. 0x80)
- Length including header (4 bytes)
- Non-resident flag (1 byte)
- Name length (1 byte)
- Offset to the Name (2 bytes)
- Flags (2 bytes)
- Attribute Id (2 bytes)
- Starting VCN (8 bytes)
- Last VCN (8 bytes)
- Offset to the Data Runs (2 bytes)
- Compression Unit Size (2 bytes)
- Padding (4 bytes)
- Allocated size of the attribute (8 bytes)
- Real size of the attribute (8 bytes)
- Initialized data size of the stream (8 bytes)
- Data Runs ...

In this section we are interested in Compression Unit size (zero in our case means non-compressed), Allocated and Real size of attribute that is equal to our file size (0xDC00 = 56320 bytes), and Data Runs (see the next topic).

## Defining the Chain of Clusters

To reconstruct a file from a set of clusters, we need to define a chain of clusters. Here are the steps:

1. Scan the drive to locate and identify data.
2. One-by-one, go through each file cluster (NTFS) or each free cluster (FAT) that we presume belongs to the file
3. Continue chaining the clusters until the size of the cumulative total of clusters approximately equals the total size of the deleted file. If the file is fragmented, the chain of clusters will be composed of several extents (NTFS), or select probable contiguous clusters and bypass occupied clusters that appear to have random data (FAT).

   The location of these clusters can vary depending on file system. For example, a file deleted in a FAT volume has its first cluster in the Root entry; the other clusters can be found in the File Allocation Table. In NTFS each file has a _**DATA**_ attribute that describes “data runs”. Disassembling data runs reveals **extents**. For each extent there is a **start cluster offset** and a **number of clusters in extent**. By enumerating the extents, the file’s cluster chain can be assembled.

   The clusters chain can be assembled manually, using low-level disk editors, however it is much simpler using a data recovery utility, like **Active@ UNERASER**.

### Defining a Cluster Chain in FAT16

In the previous topic, we were examining a sample set of data with a deleted file named **MyFile.txt**. This example will continue with the same theme.

The folder we scanned before contains a record for this file:

```
0003EE60 E5 4D 00 79 00 46 00 69   00 6C 00 0F 00 BA 65 00  aM.y.F.i.l...?e.
0003EE70 2E 00 74 00 78 00 74 00   00 00 00 00 FF FF FF FF  ..t.x.t.....yyyy
0003EE80 E5 59 46 49 4C 45 20 20   54 58 54 20 00 C3 D6 93  aYFILE TXT .AO"
0003EE90 56 2B 56 2B 00 00 EE 93   56 2B 03 00 33 B7 01 00  V+V+..i"V+..3..
```

We can calculate size of the deleted file based on root entry structure. Last four bytes are 33 B7 01 00 and converting them to decimal value (changing bytes order), we get 112435 bytes. Previous 2 bytes (03 00) are the number of the first cluster of the deleted file. Repeating for them the conversion operation, we get number 03 - this is the start cluster of the file.

What we can see in the File Allocation Table at this moment?

```
Offset   0  1  2  3  4  5  6  7     8  9  A  B  C  D  E  F
00000200 F8 FF FF FF FF FF 00 00   00 00 00 00 00 00 08 00  oyyyyy..........
00000210 09 00 0A 00 0B 00 0C 00   0D 00 FF FF 00 00 00 00  ..........yy....
00000220 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
```

Zeros! And it is good in our case - it means that these clusters are free, i.e. most likely our file was not overwritten by another file's data. Now we have chain of clusters 3, 4, 5, 6 and we are ready to recover it.

Some explanations:

- We started looking from offset 6 because each cluster entry in FAT16 takes 2 bytes, our file starts from 3rd cluster, i.e. 3*2=6.
- We considered 4 clusters because cluster size on our drive is 32 Kb, our file size is 112, 435 bytes, i.e. 3clusters*32Kb = 96Kb plus a little bit more.
- We assumed that this file was not fragmented, i.e. all clusters were located consecutively. We need 4 clusters, we found 4 free consecutive clusters, so this assumption sounds reasonable, although in real life it may be not true.

***(i)*** *Note: In many cases data cannot be successfully recovered, because the cluster chain cannot be defined. This will occur when another file or folder is written on the same drive as the one where the deleted file is located. Warning messages about this fact will be displayed while recovering data using Active@ UNDELETE.*

### Defining a Cluster Chain in NTFS

When recovering in NTFS, a part of DATA attributes called **Data Runs** provides the location of file clusters. In most cases, DATA attributes are stored in the Master File Table (MFT) record. Finding the MFT record for a deleted file will most likely lead to the location of the cluster's chain.

In example below the DATA attribute is marked with a green color. Data Runs inside the DATA attribute are marked as Bold.

```
Offset   0  1  2  3  4  5  6  7     8  9  A  B  C  D  E  F
00012580 2E 00 70 00 70 00 74 00   80 00 00 00 48 00 00 00  ..p.p.t._...H...
00012590 01 00 00 00 00 00 04 00   00 00 00 00 00 00 00 00  ................
```

```
000125A0 6D 00 00 00 00 00 00 00   40 00 00 00 00 00 00 00  m.......@.......
000125B0 00 DC 00 00 00 00 00 00   00 DC 00 00 00 00 00 00  .U.......U......
000125C0 00 DC 00 00 00 00 00 00   31 6E EB C4 04 00 00 00  .U......1neA....
000125D0 FF FF FF FF 82 79 47 11   00 00 00 00 00 00 00 00  yyyy,yG.........
```

### *Decrypting Data Runs*

Decrypting data runs can be accomplished using the following steps:

1. First byte (0x31) shows how many bytes are allocated for the length of the run (0x1 in the example case) and for the first cluster offset (0x3 in our case).
2. Take one byte (0x6E) that points to the length of the run.
3. Pick up 3 bytes pointing to the start cluster offset (0xEBC404).
4. Changing bytes order we get first cluster of the file 312555 (equals 0x04C4EB).
5. Starting from this cluster we need to pick up 110 clusters (equals 0x6E).
6. Next byte (0x00) tells us that no more data runs exist.
7. Our file is not fragmented, so we have the only one data run.
8. Lastly, check to see if there is enough information (size of the file). Cluster size is 512 bytes. There are 110 clusters, 110*512 = 56,320 bytes. Our file size was defined as 56,320 bytes, so we have enough information now to recover the file clusters.

## Recovering the Chain of Clusters

After the cluster chain is defined, the final task is to read and save the contents of the defined clusters to another place, verifying their contents. With a chain of clusters and standard formulae, it is possible to calculate each **cluster offset** from the beginning of the drive. Formulae for calculating cluster offset vary, depending on file system. Starting from the calculated offset, copy a volume of data equal to the size of the chain of clusters into a newly-created file.

To calculate the cluster offset in a FAT drive, we need to know:

- Boot sector size
- Number of FAT-supported copies
- Size of one copy of FAT
- Size of main root folder
- Number of sectors per cluster
- Number of bytes per sector

NTFS format defines a linear space and calculating the cluster offset is simply a matter of multiplying the cluster number by the cluster size.

### Recovering Cluster Chain in FAT16

This section continues the examination of the deleted file **MyFile.txt** from previous topics. By now we have chain of clusters numbered 3, 4, 5 and 6 identified for recovering. Our cluster consists of 64 sectors, sector size is 512 bytes, so cluster size is: 64*512 = 32,768 bytes = 32 Kb.

The first data sector is 535 (we have 1 boot sector, plus 2 copies of FAT times 251 sectors each, plus root folder 32 sectors, total 534 occupied by system data sectors).

Clusters 0 and 1 do not exist, so the first data cluster is 2.

Cluster number 3 is next to cluster 2, i.e. it is located 64 sectors behind the first data sector (535 + 64 = 599).

Equal offset of 306,668 byte from the beginning of the drive (0x4AE00).

With a help of low-level disk editor on the disk we can see our data starting with offset 0x4AE00, or cluster 3, or sector 599:

```
Offset   0  1  2  3  4  5  6  7     8  9  A  B  C  D  E  F
0004AE00 47 55 49 20 6D 6F 64 65  20 53 65 74 75 70 20 68  GUI mode Setup h
0004AE10 61 73 20 73 74 61 72 74  65 64 2E 0D 0A 43 3A 5C  as started...C:\
0004AE20 57 49 4E 4E 54 5C 44 72  69 76 65 72 20 43 61 63  WINNT\Driver Cac
```

Because the cluster chain is consecutive, all we need to do is copy 112,435 bytes starting from this place. If the cluster chain was not consecutive, we would need to re-calculate the offset for each cluster and copy 3 times the value of 64*512 = 32768 bytes starting from each cluster offset. The last cluster copy remainder, 14,131 bytes is calculated as 112,435 bytes - (3 * 32,768 bytes).

### Recovering Cluster Chain in NTFS

In our example we just need to pick up 110 clusters starting from the cluster 312555.

Cluster size is 512 byte, so the offset of the first cluster would be 512 * 312555 = 160028160 = 0x0989D600

```
Offset   0  1  2  3  4  5  6  7     8  9  A  B  C  D  E  F

0989D600 D0 CF 11 E0 A1 B1 1A E1  00 00 00 00 00 00 00 00  ÐÏ.à¡±.á........
0989D610 00 00 00 00 00 00 00 00  3E 00 03 00 FE FF 09 00  ........>...þÿ..
0989D620 06 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  ................
0989D630 69 00 00 00 00 00 00 00  00 10 00 00 6B 00 00 00  i...........k...
0989D640 01 00 00 00 FE FF FF FF  00 00 00 00 6A 00 00 00  ....þÿÿÿ....j...
0989D650 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

In the above data, data recovery is complete when data has been read from this point through 110 clusters (56320 bytes). This data is copied to another location.

# The Partition Recovery Process

## System Boot Process

In some cases, the first indication of a problem with hard drive data is a refusal of the machine to perform a bootstrap startup. For the machine to be able to start properly, the following conditions must apply:

- Master Boot Record (MBR) exists and is safe
- Partition Table exists and contains at least one active partition

If the above is in place, executable code in the MBR selects an active partition and passes control there, so it can start loading the standard files (COMMAND.COM, NTLDR,...) depending on the file system type on that partition.

If these files are missing or corrupted it will be impossible for the operating system to boot - if you have ever seen the famous “NTLDR is missing...” error, you understand the situation.

When using Active@ File Recovery, the recovery software accesses the damaged drive at a low level, bypassing the standard system boot process (this is the same as if you instructed the computer to boot from another hard drive). Once the computer is running in this recovery environment, it will help you to see all other files and directories on the drive and allow you to copy data to a safe place on another drive.

### Partition Visibility

A more serious situation exists if your computer will start and cannot see a drive partition or physical drive (see Note below). For the partition or physical drive to be visible to the Operating System the following conditions must apply:

- Partition/Drive can be found via Partition Table
- Partition/Drive boot sector is safe

If the above conditions are true, the OS can read the partition or physical drive parameters and display the drive in the list of the available drives.

If the file system is damaged (Root, FAT area on FAT12/FAT16/FAT32, or system MFT records on NTFS) the drive's content might not be displayed and we might see errors like “MFT is corrupted”, or “Drive is invalid”... If this is the case it is less likely that you will be able to restore your data. Do not despair, as there may be some tricks or tips to display some of the residual entries that are still safe, allowing you to recover your data to another location.

Partition recovery describes two things:

***Physical partition recovery.*** The goal is to identify the problem and write information to the proper place on the hard drive so that the partition becomes visible to the OS again. This can be done using manual Disk Editors along with proper guidelines or using recovery software, designed specifically for this purpose.

Active@ Partition Recovery software implements this approach.

***Virtual partition recovery.*** The goal is to determine the critical parameters of the deleted/damaged/overwritten partition and render it open to scanning in order to display its content. This approach can be applied in some cases when physical partition recovery is not possible (for example, partition boot sector is dead) and is commonly used by recovery software. This process is almost impossible to implement it manually.

Active@ File Recovery, Active@ UNERASER for DOS software both implement this approach.

***(i)*** *Note: If your computer has two operating systems and you choose to start in Windows 95/98 or ME, these operating systems cannot see partitions that are formatted for NTFS. This is normal operation for these operating systems. To view NTFS partitions, you must be in a Windows NT/2000/XP environment.*

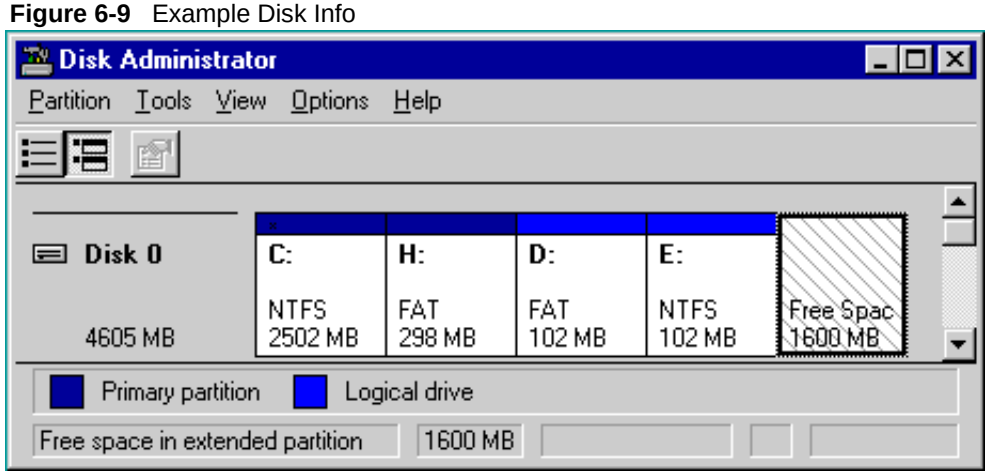
### Other Partition Recovery Topics

These topics related to the recovery of partitions apply to any file system:

- MBR is Damaged
- Partition is Deleted or Partition Table is Damaged
- Partition Boot Sector is Damaged
- Missing or Corrupted System Files

For these topics the following disk layout will be used:

**Figure 6-9** Example Disk Info




The figure shows a system with two primary partitions (C:(NTFS) and H:(FAT)) and one extended partition having two logical drives (D: (FAT) and E:(NTFS))

### MBR is Damaged

The Master Boot Record (MBR) will be created when you create the first partition on the hard disk. It is very important data structure on the disk. The Master Boot Record contains the Partition Table for the disk and a small

amount of executable code for the boot start. The location is always the first sector on the disk.

The first 446 (0x1BE) bytes are MBR itself, the next 64 bytes are the Partition Table, the last two bytes in the sector are a signature word for the sector and are always 0x55AA.

**Blank Screen on Startup**

For our disk layout we have MBR:

```
Physical Sector: Cyl 0, Side 0, Sector 1
000000000   33 C0 8E D0 BC 00 7C FB  50 07 50 1F FC BE 1B 7C   3AZ??.|uP.P.u?.|
000000010   BF 1B 06 50 57 B9 E5 01  F3 A4 CB BE BE 07 B1 04   ?..PW?a.o¤E??.±.
000000020   38 2C 7C 09 75 15 83 C6  10 E2 F5 CD 18 8B 14 8B   8,|.u.??.aoI.‹.‹
000000030   EE 83 C6 10 49 74 16 38  2C 74 F6 BE 10 07 4E AC   i??.It.8,to?..N¬
000000040   3C 00 74 FA BB 07 00 B4  0E CD 10 EB F2 89 46 25   <.tu»..?.I.eo‰F%
000000050   96 8A 46 04 B4 06 3C 0E  74 11 B4 0B 3C 0C 74 05   –SF.?.<.t.?.<.t.
000000060   3A C4 75 2B 40 C6 46 25  06 75 24 BB AA 55 50 B4   :Au+@?F%.u$»?UP?
000000070   41 CD 13 58 72 16 81 FB  55 AA 75 10 F6 C1 01 74   AI.Xr.?uU?u.oA.t
000000080   0B 8A E0 88 56 24 C7 06  A1 06 EB 1E 88 66 04 BF   .Sa?V$C.?.e.?f.?
000000090   0A 00 B8 01 02 8B DC 33  C9 83 FF 05 7F 03 8B 4E   ..?..‹U3E?y..‹N
0000000A0   25 03 4E 02 CD 13 72 29  BE 46 07 81 3E FE 7D 55   %.N.I.r)?F.?>?}U
0000000B0   AA 74 5A 83 EF 05 7F DA  85 F6 75 83 BE 27 07 EB   ?tZ?i.U…ou??'.e
0000000C0   8A 98 91 52 99 03 46 08  13 56 0A E8 12 00 5A EB   S?‘R™.F..V.e..Ze
0000000D0   D5 4F 74 E4 33 C0 CD 13  EB B8 00 00 00 00 00 00   OOta3AI.e?......
0000000E0   56 33 F6 56 56 52 50 06  53 51 BE 10 00 56 8B F4   V3oVVRP.SQ?..V‹o
0000000F0   50 52 B8 00 42 8A 56 24  CD 13 5A 58 8D 64 10 72   PR?.BSV$I.ZX?d.r
000000100   0A 40 75 01 42 80 C7 02  E2 F7 F8 5E C3 EB 74 49   .@u.B^C.a?o^AetI
000000110   6E 76 61 6C 69 64 20 70  61 72 74 69 74 69 6F 6E   nvalid partition
000000120   20 74 61 62 6C 65 00 45  72 72 6F 72 20 6C 6F 61    table.Error loa
000000130   64 69 6E 67 20 6F 70 65  72 61 74 69 6E 67 20 73   ding operating s
000000140   79 73 74 65 6D 00 4D 69  73 73 69 6E 67 20 6F 70   ystem.Missing op
000000150   65 72 61 74 69 6E 67 20  73 79 73 74 65 6D 00 00   erating system..
000000160   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000000170   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000000180   00 00 00 8B FC 1E 57 8B  F5 CB 00 00 00 00 00 00   ...‹u.W‹oE......
000000190   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
0000001A0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
0000001B0   00 00 00 00 00 00 00 00  A6 34 1F BA 00 00 80 01   ........¦4.?..^.
0000001C0   01 00 07 FE 7F 3E 3F 00  00 00 40 32 4E 00 00 00   ...?>?...@2N...
0000001D0   41 3F 06 FE 7F 64 7F 32  4E 00 A6 50 09 00 00 00   A?.?d2N.¦P....
0000001E0   41 65 0F FE BF 4A 25 83  57 00 66 61 38 00 00 00   Ae.??J%?W.fa8...
0000001F0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 AA   ..............U?
```

To simulate what will happen if the first sector has been damaged (by a virus, for example), we will overwrite the first 16 bytes with zeros, as shown below:

```
000000000   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000000010   BF 1B 06 50 57 B9 E5 01  F3 A4 CB BE BE 07 B1 04   ?..PW?a.o¤E??.±.
```

We have effectively destroyed the MBR at this point. When we try to restart the computer, we see the hardware testing procedures, and then a blank screen without any messages. This blank screen confirms that the piece of code at the beginning of the MBR could not be executed properly. Error messages cannot be displayed because the MBR cannot be run.

If we boot from a system floppy, however, we can see a hard drive FAT partition and the files on it. We are able to perform standard operations like file copy, program execution and so on. This is possible because only the first part of the MBR has been damaged. The partition table is safe and we can access our drives when we boot from the operating system installed on the other drive.

### Operating System Not Found

In this next scenario, we explore what will happen if the **sector signature** (last word 0x55AA) has been removed or damaged?

To explore this scenario, we write zeros to the location of sector signature, as shown below:

```
Physical Sector: Cyl 0, Side 0, Sector 1
0000001E0   41 65 0F FE BF 4A 25 83  57 00 66 61 38 00 00 00   Ae.??J%?W.fa8...
0000001F0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

When we try to boot now, we see the “Operating System not found” error message.

When encountering this message on system boot, run **Disk Viewer** and check the first physical sector on the hard drive to see whether it looks like a valid MBR or not. Here are things to check:

- See if it is filled up with zeros or any other single character.
- Check whether error messages (like you can see above “Invalid partition table”...) are present or not.
- Check whether the disk signature (0x55AA) is present.

The simplest way to repair or re-create the MBR is to run Microsoft's standard utility called **FDISK** with a parameter **/MBR**. The command looks like the sample below:

```
A:\> FDISK.EXE  /MBR
```

FDISK is a standard utility included in MS-DOS, Windows 95, 98, ME.

If you have Windows NT / 2000 / XP, you can boot from startup floppy disks or CD-ROM, choose **Repair** option during setup, and run **Recovery Console**. When you are logged on, you can run **FIXMBR** command to repair the MBR.

Another alternative is to use a third party MBR recovery utility or if you've created an MBR backup, repair the damaged MBR by restoring the backup (Active@ Partition Recovery has such capabilities).

### Recovering Data if the First Sector is Bad or Unreadable

In the **Blank Screen** simulation, above, we simulated the destroyed first sector scenario. When you try to read the first sector using Disk Viewer/Editor you should get an error message saying that the sector is unreadable. In this case recovery software is unable to help you to bring the hard drive back to the working condition, i.e. physical partition recovery is not possible.

The only thing that can be done is to scan and search for partitions (i.e. perform virtual partition recovery). When something is found - display the data save it to another location. Software, like Active@ File Recovery, Active@ UNERASER for DOS will help you here.

### Partition is Deleted or Partition Table is Damaged

The information about primary partitions and extended partition is contained in the Partition Table, a 64-byte data structure, located in the same sector as the Master Boot Record (cylinder 0, head 0, sector 1). The Partition Table conforms to a standard layout, which is independent of the operating system. The last two bytes in the sector are a signature word for the sector and are always 0x55AA.

For our disk layout we have Partition Table:

```
 Physical Sector: Cyl 0, Side 0, Sector 1
0000001B0                                                80 01   ..............€.
0000001C0   01 00 07 FE 7F 3E 3F  00 00 00 40 32 4E 00 00 00   ...?>?...@2N...
0000001D0   41 3F 06 FE 7F 64 7F 32  4E 00 A6 50 09 00 00 00   A?.?d2N.¦P....
0000001E0   41 65 0F FE BF 4A 25 83  57 00 66 61 38 00 00 00   Ae.??J%?W.fa8...
0000001F0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 AA   ..............U?
```

We can see three existing entries and one empty entry:

- Partition 1, offset 0x01BE (446)
- Partition 2, offset 0x01CE (462)
- Partition 3, offset 0x01DE (478)
- Partition 4 - empty, offset 0x01EE (494)

Each Partition Table entry is 16 bytes long, making a maximum of four entries available. Each partition entry has fields for Boot Indicator (BYTE), Starting Head (BYTE), Starting Sector (6 bits), Starting Cylinder (10 bits), System ID (BYTE), Ending Head (BYTE), Ending Sector (6 bits), Ending Cylinder (10 bits), Relative Sector (DWORD), Total Sectors (DWORD).

Thus the MBR loader can assume the location and size of partitions. MBR loader looks for the “active” partition, i.e. partition that has Boot Indicator equals 0x80 (the first one in our case) and passes control to the partition boot sector for further loading.

Below, a number of situations are simulated demonstrating events which cause a computer to hang while booting or in a data loss scenario:

1. No disk partition has been set to the Active state (Boot Indicator=0x80).

   To simulate this scenario, remove the Boot Indicator from the first partition as below:

   ```
   0000001B0                                               00 01  ................
   0000001C0 01 00 07 FE 7F 3E 3F  00 00 00 40 32 4E 00 00 00  ...?>?...@2N...
   ```

   When we try to boot now, we see an error message like “Operating System not found”. This demonstrates a situation where the loader wants to pass control to the active system, and cannot determine which partition is active and contains the system.

2. A partition has been set to the Active state (Boot Indicator=0x80) but there are no system files on that partition.

   (This situation is possible if we had used FDISK and not selected the correct active partition).

The Loader tries to pass control to the partition, fails, tries to boot again from other devices like the floppy. If it fails to boot again, an error message like “Non-System Disk or Disk Error” appears.

3 Partition entry has been deleted.

If the partition entry has been deleted, the next two partitions will move one line up in the partition table, as below:

```
Physical Sector: Cyl 0, Side 0, Sector 1
0000001B0                                                 80 00   ..............€.
0000001C0   41 3F 06 FE 7F 64 7F 32  4E 00 A6 50 09 00 00 00   A?.?d2N.¦P....
0000001D0   41 65 0F FE BF 4A 25 83  57 00 66 61 38 00 00 00   Ae.??J%?W.fa8...
0000001E0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
0000001F0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 AA   ..............U?
```

If we try to boot now, the partition previous identified as “second” (FAT) partition becomes the “first” and the loader will try to boot from it. If the operating system does not exist within the partition, the same error messages appear.

4 Partition entry has been damaged.

To simulate this situation, write zeros to the location of the first partition entry.

```
Physical Sector: Cyl 0, Side 0, Sector 1
0000001B0                                                 80 00   ..............€.
0000001C0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
0000001D0   41 3F 06 FE 7F 64 7F 32  4E 00 A6 50 09 00 00 00   A?.?d2N.¦P....
0000001E0   41 65 0F FE BF 4A 25 83  57 00 66 61 38 00 00 00   Ae.??J%?W.fa8...
0000001F0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 AA   ..............U?
```

If we try to boot now, the MBR loader will try to read and interpret zeros (or other garbage) as partition parameters. The error message will read “Missing Operating System”.

Thus, the second step in partition recovery is to run Disk Viewer and to make sure that the proper partition exists in the partition table and has been set as active.

### Can Recovery Software Help in the Above Scenarios?

Recovery Software can help in the following ways:

1 Discover and suggest you to choose the partition to be active (even FDISK does so).

2 Discover and suggest you to choose the partition to be active.

3 Perform a free disk space scan to look for partition boot sector or remaining of the deleted partition information in order to try to reconstruct Partition Table entry for the deleted partition.

4 Perform all disk space scan to look for partition boot sector or remaining of the damaged partition information in order to try to reconstruct Partition Table entry for the damaged partition entry.

### Why is the Partition Boot Sector so Important?

If recovery software finds it, all necessary parameters to reconstruct partition entry in the Partition Table are there. (see Partition Boot Sector topic for details).

### What if a Partition Entry was Deleted Then Recreated and Re-formatted?

In this case, instead of the original partition entry we would have a new one and everything would work fine except that later on we could recall that we had

some important data on the original partition. If you've created MBR, Partition Table, Volume Sectors backup before the problem (for example, Active@ Partition Recovery and Active@ UNERASER can do this), you can virtually restore it back and look for your data (in case if it has not been overwritten with new data yet). Some advanced recovery tools also have an ability to scan the disk surface and try to reconstruct previously deleted partition information from the remnants of information (i.e. perform virtual partition recovery). However there is no guarantee that you can recover anything.

### Partition Boot Sector is Damaged

The Partition Boot Sector contains information, which the file system uses to access the volume. On personal computers, the Master Boot Record uses the Partition Boot Sector on the system partition to load the operating system kernel files. Partition Boot Sector is the first sector of the Partition.

For our first NTFS partition we have boot sector:

```
Physical Sector: Cyl 0, Side 1, Sector 1
000000000   EB 5B 90 4E 54 46 53 20  20 20 20 00 02 01 00 00   e[?NTFS    .....
000000010   00 00 00 00 00 F8 00 00  3F 00 FF 00 3F 00 00 00   .....o..?.y.?...
000000020   00 00 00 00 80 00 80 00  3F 32 4E 00 00 00 00 00   ....€.€.?2N.....
000000030   5B 43 01 00 00 00 00 00  1F 19 27 00 00 00 00 00   [C........'.....
000000040   02 00 00 00 08 00 00 00  10 EC 46 C4 00 47 C4 0C   .........iFA.GA.
000000050   00 00 00 00 00 00 00 00  00 00 00 00 00 FA 33 C0   ..............u3A
000000060   8E D0 BC 00 7C FB B8 C0  07 8E D8 C7 06 54 00 00   Z??.|u?A.ZOC.T..
000000070   00 C7 06 56 00 00 00 C7  06 5B 00 10 00 B8 00 0D   .C.V...C.[...?..
000000080   8E C0 2B DB E8 07 00 68  00 0D 68 66 02 CB 50 53   ZA+Ue..h..hf.EPS
000000090   51 52 06 66 A1 54 00 66  03 06 1C 00 66 33 D2 66   QR.f?T.f....f3Of
0000000A0   0F B7 0E 18 00 66 F7 F1  FE C2 88 16 5A 00 66 8B   .·...f?n?A?.Z.f‹
0000000B0   D0 66 C1 EA 10 F7 36 1A  00 88 16 25 00 A3 58 00   ?fAe.?6..?.%.?X.
0000000C0   A1 18 00 2A 06 5A 00 40  3B 06 5B 00 76 03 A1 5B   ?..*.Z.@;.[.v.?[
0000000D0   00 50 B4 02 8B 16 58 00  B1 06 D2 E6 0A 36 5A 00   .P?.‹.X.±.O?.6Z.
0000000E0   8B CA 86 E9 8A 36 25 00  B2 80 CD 13 58 72 2A 01   ‹E†eS6%.?€I.Xr*.
0000000F0   06 54 00 83 16 56 00 00  29 06 5B 00 76 0B C1 E0   .T.?.V..).[.v.Aa
000000100   05 8C C2 03 D0 8E C2 EB  8A 07 5A 59 5B 58 C3 BE   .?A.?ZAeS.ZY[XA?
000000110   59 01 EB 08 BE E3 01 EB  03 BE 39 01 E8 09 00 BE   Y.e.?a.e.?9.e..?
000000120   AD 01 E8 03 00 FB EB FE  AC 3C 00 74 09 B4 0E BB   -.e..ue?¬<.t.?.»
000000130   07 00 CD 10 EB F2 C3 1D  00 41 20 64 69 73 6B 20   ..I.eoA..A disk
000000140   72 65 61 64 20 65 72 72  6F 72 20 6F 63 63 75 72   read error occur
000000150   72 65 64 2E 0D 0A 00 29  00 41 20 6B 65 72 6E 65   red....).A kerne
000000160   6C 20 66 69 6C 65 20 69  73 20 6D 69 73 73 69 6E   l file is missin
000000170   67 20 66 72 6F 6D 20 74  68 65 20 64 69 73 6B 2E   g from the disk.
000000180   0D 0A 00 25 00 41 20 6B  65 72 6E 65 6C 20 66 69   ...%.A kernel fi
000000190   6C 65 20 69 73 20 74 6F  6F 20 64 69 73 63 6F 6E   le is too discon
0000001A0   74 69 67 75 6F 75 73 2E  0D 0A 00 33 00 49 6E 73   tiguous....3.Ins
0000001B0   65 72 74 20 61 20 73 79  73 74 65 6D 20 64 69 73   ert a system dis
0000001C0   6B 65 74 74 65 20 61 6E  64 20 72 65 73 74 61 72   kette and restar
0000001D0   74 0D 0A 74 68 65 20 73  79 73 74 65 6D 2E 0D 0A   t..the system...
0000001E0   00 17 00 5C 4E 54 4C 44  52 20 69 73 20 63 6F 6D   ...\NTLDR is com
0000001F0   70 72 65 73 73 65 64 2E  0D 0A 00 00 00 00 55 AA   pressed.......U?

Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
```

The printout is formatted in three sections:

- Bytes 0x00– 0x0A are the jump instruction and the OEM ID (shown in bold print).
- Bytes 0x0B–0x53 are the BIOS Parameter Block (BPB) and the extended BPB.

  This block contains such essential parameters as:

  - Bytes Per Sector (WORD, offset 0x0B),
  - Sectors Per Cluster (BYTE, offset 0x0D),

- Media Descriptor (BYTE, offset 0x15),
- Sectors Per Track (WORD, offset 0x18),
- Number of Heads (WORD, offset 0x1A),
- Hidden Sectors (DWORD, offset 0x1C),
- Total Sectors (LONGLONG, offset 0x28), etc....

- The remaining code is the bootstrap code (that is necessary for the proper system boot) and the end of sector marker (shown in bold print).

This sector is so important on NTFS, for example, that a duplicate of the boot sector is located on the disk.

Boot Sector for FAT looks different, however its BPB contains parameters similar to the above mentioned. There is no extra copy of this sector stored anywhere, so recovery on FAT is not as convenient as it is on NTFS.

### What Will Happen if Partition Boot Sector is Damaged or Bad/Unreadable?

To simulate this scenario, we fill up several lines of the Partition Boot Sector with zeros:

```
000000000   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000000010   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000000020   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000000030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000000040   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000000050   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000000060   8E D0 BC 00 7C FB B8 C0  07 8E D8 C7 06 54 00 00   Z??.|u?A.ZOC.T..
```

If we try to boot, we'll see “Non System Disk” or “Disk Error”. After we fail to load from it and from floppy, partition becomes unbootable.

Because a normally functioning system relies on the boot sector to access a volume, it is highly recommended that you run disk-scanning tools such as **Chkdsk** regularly, as well as back up all of your data files to protect against data loss in case you lose access to the volume.

Tools like Active@ Partition Recovery and Active@ UNERASER allow you to create a backup of the MBR, Partition Table and Volume Boot Sectors so that if for some reason the system fails to boot, you can restore your partition information and have access to files and folders on that partition.

### What if This Sector is Damaged?

- If we do have backup of the whole disk or MBR/Boot Sectors we can try to restore it from there.
- If we do not have backup, in case of NTFS we could try to locate a duplicate of Partition Boot Sector and get information from there.
- If duplicate boot sector is not found, only virtual partition recovery might be possible if we can determine critical partition parameters such as Sectors per Cluster, etc.

### Can I Fix NTFS Boot Sector Using Standard Windows NT/2000/XP Tools?

On NTFS a copy of the boot sector is stored in the middle or at the end of the Volume.

You can boot from startup floppy disks or CD-ROM, choose the **Repair** option during setup, and run **Recovery Console**. When you are logged on, you can run the **FIXBOOT** command to try to fix boot sector.

**Can Recovery Software Help in This Situation?**

It can backup MBR, Partition Table and Boot Sectors and restore them in case of damage.

It can try to find out duplicate boot sector on the drive and re-create the original one or perform virtual data recovery based on found partition parameters

Some advanced techniques allow assuming drive parameters even if duplicate boot sector is not found (i.e. perform virtual partition recovery) and give the user virtual access to the data on the drive to be able to copy them to the safer location.

## Missing or Corrupted System Files

For the operating system to boot properly, system files are required to be safe.

In case of Windows 95 / 98 / ME, these files are **msdos.sys, config.sys, autoexec.bat, system.ini, system.dat, user.dat,** etc.

In case of Windows NT / 2000 / XP these files are: **NTLDR, ntdetect.com, boot.ini,** located at the root folder of the bootable volume, **Registry files** (i.e., SAM, SECURITY, SYSTEM and SOFTWARE), etc.

If these files have been deleted, corrupted or damaged by a virus, Windows will be unable to boot. You'll see error messages like “NTLDR is missing ...“.

The next step in the recovery process is to check the existence and safety of system files (you won't able to check them all, but you must check at least **NTLDR**, **ntdetect.com**, **boot.ini** which cause most problems).

To do it in Windows 95 / 98 / ME, boot in **Command Prompt** mode, or from a bootable floppy and check the system files in the command line or with a help of third party recovery software.

To do it in Windows NT / 2000 / XP, use the **Emergency Repair Process**, **Recovery Console** or third party recovery software.

**Emergency Repair Process**

To proceed with Emergency Repair Process, you need an **Emergency Repair Disk** (ERD). It is recommended to create an ERD after you install and customize Windows. To create it, use the **Backup** utility from System Tools. You can use the ERD to repair a damaged boot sector, damaged MBR, repair or replace missing or damaged NT Loader (NTLDR) and ntdetect.com files.

If you do not have an ERD, the emergency repair process can attempt to locate your Windows installation and start repairing your system, but it may not be able to do so.

To run the process, boot from a Windows bootable disk or CD, and choose the **Repair** option when system suggests you to proceed with installation or

repairing. Then press **R** to run Emergency Repair Process and choose **Fast** or **Manual Repair** option. Fast Repair is recommended for most users, Manual Repair - for Administrators and advanced users only.

If the emergency repair process is successful, your computer will automatically restart and you should have a working system

### Recovery Console

Recovery Console is a command line utility similar to MS-DOS command line. You can list and display folder content, copy, delete, replace files, format drives and perform many other administrative tasks.

To run Recovery Console, boot from Windows bootable disks or CD and choose the **Repair** option. When the system suggests you to proceed with installation or repairing and then press **C** to run Recovery Console. You will be asked which system you want to log on to and then for the Administrator's password. After you logged on, you can display the drive's contents, check the existence and safety of critical files and, for example, copy them back to restore them if they have been accidentally deleted.

### Recovery Software

Third party recovery software in most cases does not allow you to deal with system files due to the risk of further damage to the system, however you can use it to check for the existence and safety of these files, or to perform virtual partition recovery.

# 7 Data Recovery Tips

The following tips are designed to help with data recovery.

**Treat Recovery Area With Care**

Once the drive with recoverable data is identified, do not save, write or install anything onto that drive. Even installing the data recovery software to the same drive could overwrite critical file location data and render your sensitive data irrecoverable.

**Save Recovered Files Onto a Different Drive**

Do not save the recovered files back onto the same drive from which they were recovered. This can interfere with the recovery process by overwriting table records for other deleted files or folders.

Please save your recovered data onto another logical, removable, floppy or network drive.

**Load UNERASER to a Floppy**

Download and save Active@ UNERASER for DOS onto a bootable floppy and boot your system using that disk.

# Active Data Recovery Software

2550 Argentia Road, Suite 218
Mississauga, Ontario
Canada  L5N 5R1

http://www.uneraser.com

Phone (416) 812-8434
Customer Service: sales@uneraser.com
Technical Support : support@uneraser.com