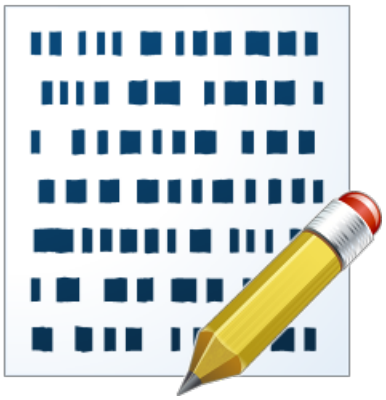


Active@ Disk Editor

USER MANUAL

ver. 25

Published: 16 Jan 2025

Contents

Overview 4

Getting started with Disk Editor 5
- Disk Explorer 6
- File Browser 9

Using Disk Editor 11
- Working with editor 11
  - Edit physical disks 13
  - Edit logical drives (volumes) 13
  - Edit files 14
  - Navigation and information 17
  - Filling a block 21
- Using Templates 22
- Disk Editor tools 25
  - Data Inspector 25
  - File cluster chain 26
  - File preview 27
  - Active Bookmarks 28
- Searching in Disk Editor 31
- Disk Management 33
  - Initialize new disk 34
  - Partition management 35
  - Disk editing 41

Appendix 42
- Preferences 42
  - General Settings 42
  - Disk Editor preferences 44
  - Error Handling 45
- Searching patterns 47
- Hardware diagnostic file 48
- Knowledge base 48
  - Overview 48
  - Hardware and Disk Architecture 49
  - File Systems 60
  - Erase Disk Concept 100
  - Wipe Disk Concepts 103
  - Sanitization Types 109
  - Disk Erase performance 110
  - Disk Hidden Zones 110
  - Virtual Disks 112
  - Data Recovery Concept 113
  - File Recovery 113
  - Partition Recovery 120
- Glossary 129

Uninstall Active@ Disk Editor.....131

Legal Statement

Overview

General system requirements

- Windows 11, Windows 10, Windows 8, Windows 7, Windows Vista, Windows XP
- Windows Servers 2003, 2008, 2012, 2016, 2019, 2022
- WinPE recovery environment
- Administrators privileges required to install and run software
- Intel/AMD processor
- 100 MB available on a disk
- 1GB MB of RAM
- Internet Browser
- Mouse or other pointing device

Welcome to Active@ Disk Editor

When application starts the Welcome screen will appear where you can choose following actions:

- Open *Disk* or *Volume*;
- Open *File*;
- Open *Disk Image*;
- Continue with browsing you local disks and data storages;

Close Welcome screen or click **File Browser** button to start browsing your local devices, volumes and files.

The simplest way to open objects for editing is to select *Disk*, *Volume* or *File* in File Browser and use **Open in Disk Editor** command in toolbar or in context menu.

Tip:

You can use the **Ctrl+H** shortcut to open any selected item in **Disk Editor**.

Overview

Active@ Disk Editor is advanced tool for viewing and editing raw data of *physical disks*, *partitions* and *volumes*, contents of any *file* type and *file records*. Disk Editor uses a simple, low-level disk viewer which displays information in binary and text modes at the same time. You can use this view to analyze the contents of data storage structure elements such as:

- Hard disk drives (disks);
- Partitions;
- Volumes (Logical drives);
- File records on volume;
- Files;

Warning: As with any advanced tool, use extreme caution with the **Disk Editor**. Changes that you make may affect disk structure integrity. You must be certain that the changes you make are in line with correct data structures before you save changes.

Disk Editor Preferences

Disk Editor memorize its state and when closed those settings are preserved. The settings saved are view options and geometry of windows.

Read Disk Editor preferences on page 44 for detailed information.

Saving Changes

Unless stated otherwise, all modifications made in the Disk Editor are stored in memory. Changes are written to the drive when you click **Save**. Read Working with editor on page 11 article for more information.

Looking for big picture?

Active@ Disk Editor is self-contained separate module developed as part of **Active @ UNDELETE - Data Recovery Toolkit**. For more features, like:

- **Recover deleted files** or folders from live, deleted or damaged volumes (partitions);
- **Restore** deleted or damaged **partitions**;
- **Create**, **Format** and **Resize** partitions;
- Create and use **Disk Images**;
- **Recover** data from damaged **RAID**'s.

Please visit Active@ UNDELETE website and download DEMO version.

Getting started with Disk Editor

To get started and get familiar with main features of **Active@ Disk Editor** read following articles:

- Working with editor on page 11
- Navigation and information on page 17

To be able to run **Active@ Disk Editor** with command line parameters read Command line article.

Active@ Disk Editor can automatically detect plugged removable devices and shows them in File Browser. However, if plugged device does not appear in click **Refresh** button in toolbar to update File Browser view or press and hold **Ctrl** button on keyboard and click **Refresh** button in toolbar to completely rescan and refresh all connected local data storages.

Warning: As with any advanced tool, use extreme caution with the Disk Editor. Changes that you make may affect disk structure integrity. You must be certain that the changes you make are in line with correct data structures before you save changes.

Related information

Disk Explorer on page 6
File Browser on page 9

Disk Explorer

Disk Explorer is a default workspace for the **Active@ Disk Editor** application. All attached HDD/SSD/USB disks are visualized here and can be selected for different actions. Majority of commands can be initiated from here as well as progress displayed for actions performed with disks.

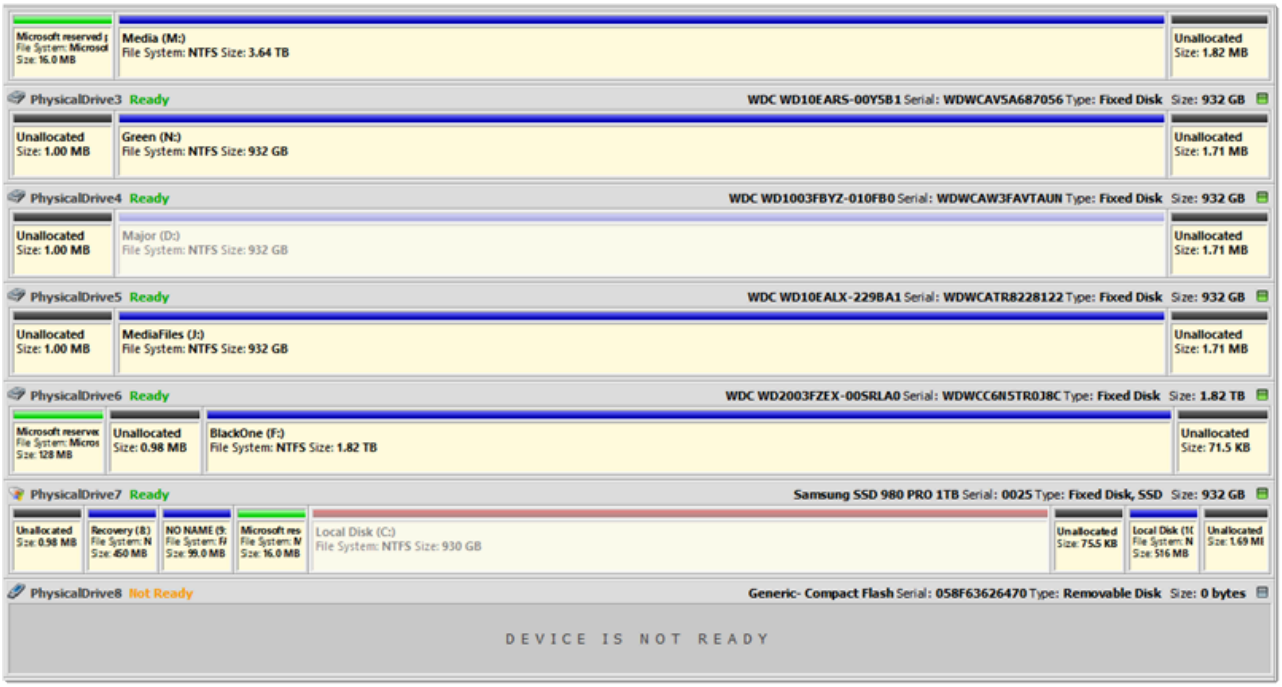


Figure 1: Disk Explorer View

An additional toolbar helps to execute frequently performed tasks. It contains the following buttons with drop-down menus:

View

The disk explorer supports a range of different views to use when performing **Active@ Disk Editor** actions, each with their own customizable settings for different use cases.

Customize

These settings (different for each View) let you customize appearance for better experience with each View.

Select disk partition volume or other object in Disk Explorer and use available command from views toolbar or **Action** main menu to perform an action.

Active@ Disk Editor can automatically detect plugged removable devices and shows them in Disk Explorer. However, if plugged device does not appear in click **Refresh** button in toolbar to update Disk

Explorer view or press and hold **Ctrl** button on keyboard and click **Refresh** button in toolbar to completely rescan and refresh all connected local data storage's.

Additional tool views such as Output view or Property View available through **View** > **Windows** main menu. Use Property or SMART info panes to view detailed information about selected item attributes and device SMART info (if available). Read more in: Property views.

Device view mode

At this view mode Disk Explorer shows all disks recognized by the Operating System as a flat list. This is default view mode.

Device view mode has some customization options to adjust:
Customize options

Show System Devices

Displays the disk where Operating System is installed. This is off by default to prevent accidental erasure of the system

Show Not Ready Devices

Displays devices not yet initialized and used by Operating System

Show Removable Devices

Displays all removable and externally connected disks (such as USB Flash Drives and External USB Disks)

Show Virtual Disks

Displays virtual disk storages, such as RAID's, Google drives etc.

Compact View

Changes the layout of the Disk View from display block to inline block orientation

Show partitions

Hides or shows additional partition items.

My Computer view mode

Additional view mode, similar to File Browser view that shows additional system elements in a standard tree-view form, much like the disks in Windows Explorer. Information for the currently selected object such as disk status, serial number, partitioning displayed in Properties window at the right side.

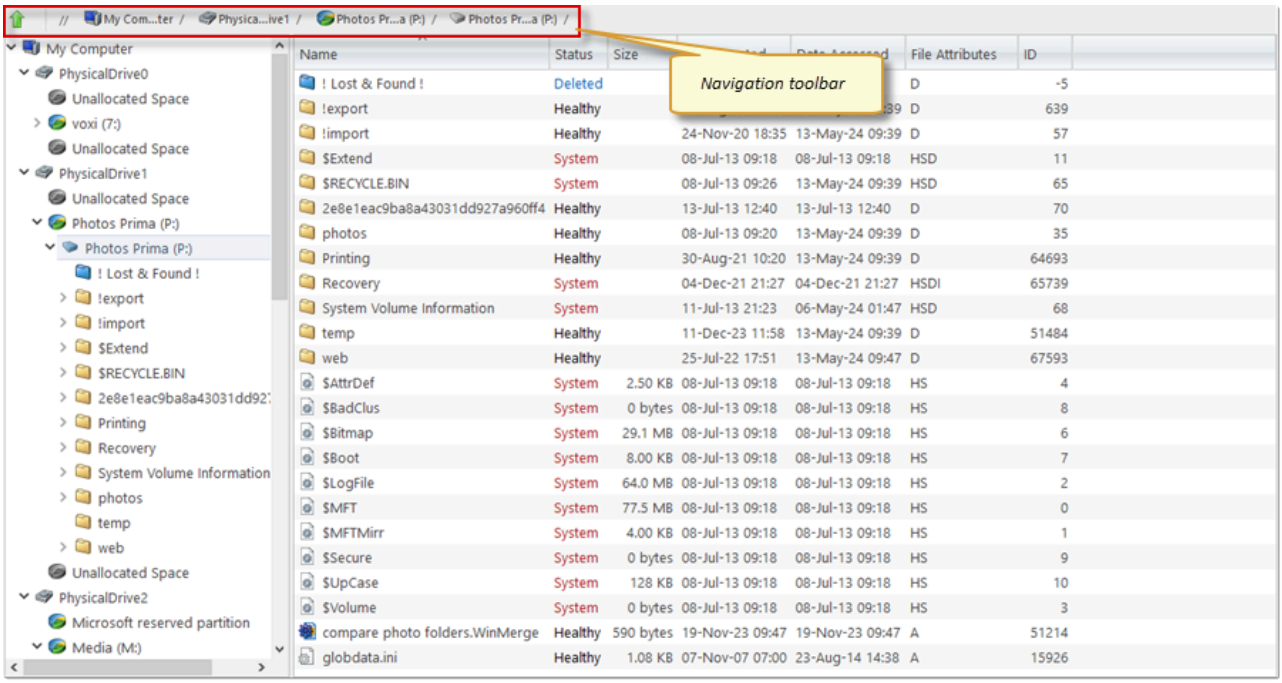


Figure 2: My Computer view mode

This view mode is also adjustable through **Customize** drop down menu, similar to Device view mode:

Customize menu

Show My Computer

Displays root **My Computer** node;

Show System Disk

Displays the disk containing the Operating System. This is off by default to prevent accidental erasure of the system

Show Unallocated Partitions

Displays disk's unallocated space - partitions where no volumes created yet

Show Devices

Switches between display of Devices (physical disks containing volumes) and Volumes only display

Show Removable Disks

Displays removable media storage devices (USB Flash Disk, External USB etc.)

Show Not Ready Devices

Displays devices that may not yet been initialized and accessed by the Operating System

Show Virtual Disks

Displays virtual disk storages

Show Local Network

Shows/hides **Navigator Pane** on the right side of the View

Navigator Pane

Shows/hides **Navigator Pane** (tree) on the right side of the View

Navigation toolbar control shows current path and helps to navigate directly to any folder at this path.

Tip:

You can use the **Ctrl+H** shortcut to open any selected item in **Disk Editor**.

Related tasks

Map Network Shares

Related information

Preferences on page 42

File Browser on page 9

Application Log

Output view

File Browser

File Browser is subset of Disk Explorer that allows to review content of physical disk, partition of volume individually in tree-view layout to file level on all major file systems used in Windows, Linux, Unix or Mac OS.

Note:

Active@ Disk Editor shows existing files as well as files that have been deleted but NOT sanitized. They appear in Gray color and indicate deleted files with a high probability of being recovered with a special file recovery tools.

Active@ Disk Editor can automatically detect plugged removable devices and shows them in File Browser. However, if plugged device does not appear in click **Refresh** button in toolbar to update File Browser view or press and hold **Ctrl** button on keyboard and click **Refresh** button in toolbar to completely rescan and refresh all connected local data storage's.

Use Property or SMART Info panes to view detailed information about selected item attributes and device SMART info (if available). Read more in: Property views.

Browse Disk View

To browse the contents of a specific disk simply select the disk, partition or volume in Disk Explorer and click **Open in File Browser** from the **Action** menu or select the related command from the context menu.

Another way is to use a keyboard shortcut which is **Ctrl-B**. This will open the File Browser window:

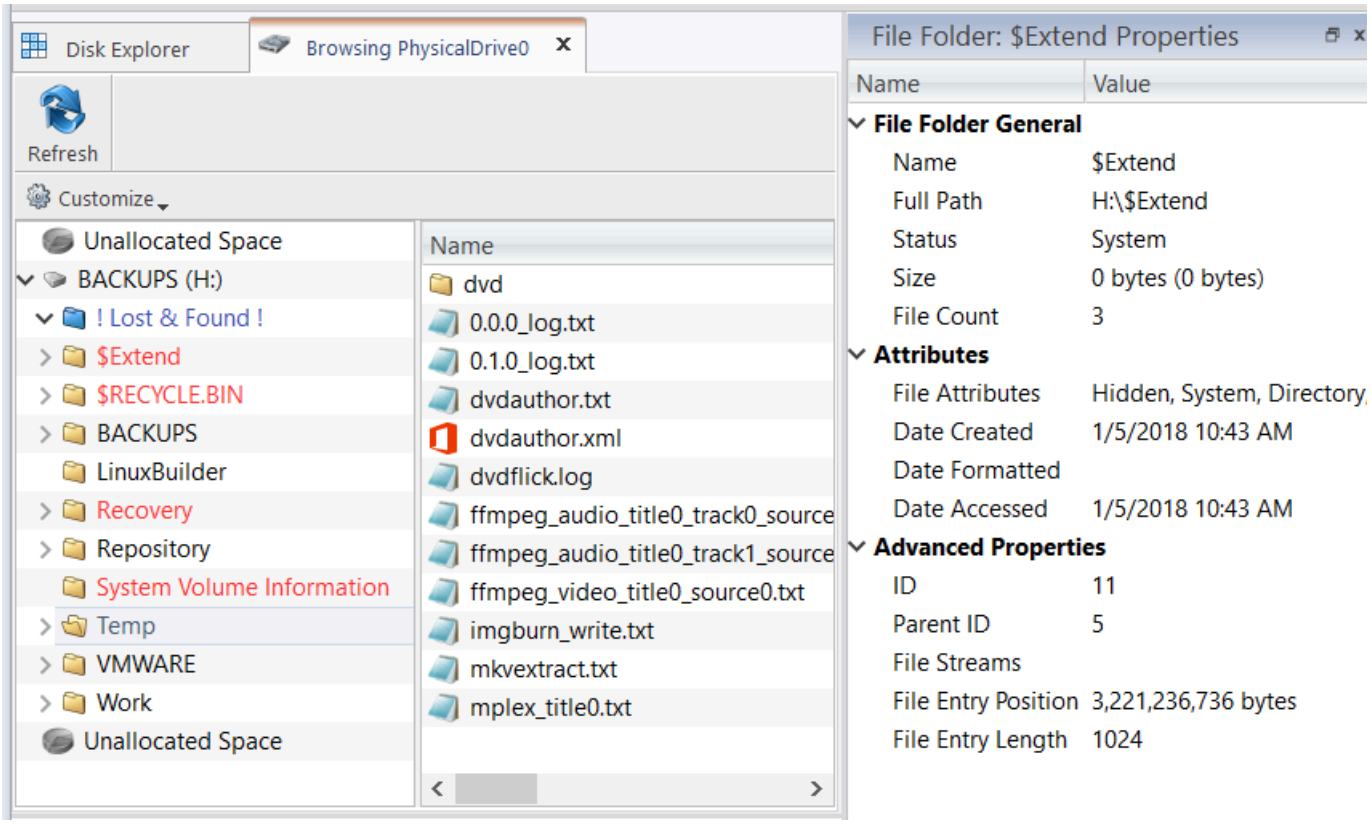


Figure 3: File Browser Window

The File Browser tab displays files and folders on the disk being selected. Browsing over the folders tree performed the same way as default OS Explorer. The File Browser tabbed view may also be adjusted by **Customize** menu. Here you have options to adjust:

Show System Files
Toggles display of advanced disk information (system files)
Show Unallocated Partitions
Toggles display of the unallocated disk partitions when browsing physical disks
Navigator Pane
Toggles display of the Navigator Pane

Files of folders in gray color indicates deleted files have not been sanitized. These files are recoverable.

Note:

Found deleted files appear in their original directory (before they were deleted). The **! Lost & Found !** folder is a virtual directory created for deleted files which are found without directory information.

Related tasks

Map Network Shares

Using Disk Editor

Working with editor

The **Active@ Disk Editor** allows you to edit the content of a selected part of an opened object. By default, the Disk Editor shows the content of an object in a *Read Only* mode that prevents accidental modifications. In *Edit* mode, you can change the content of the opened file or disk and all modifications are stored in memory. Changes are written to the drive when you click **Save**.

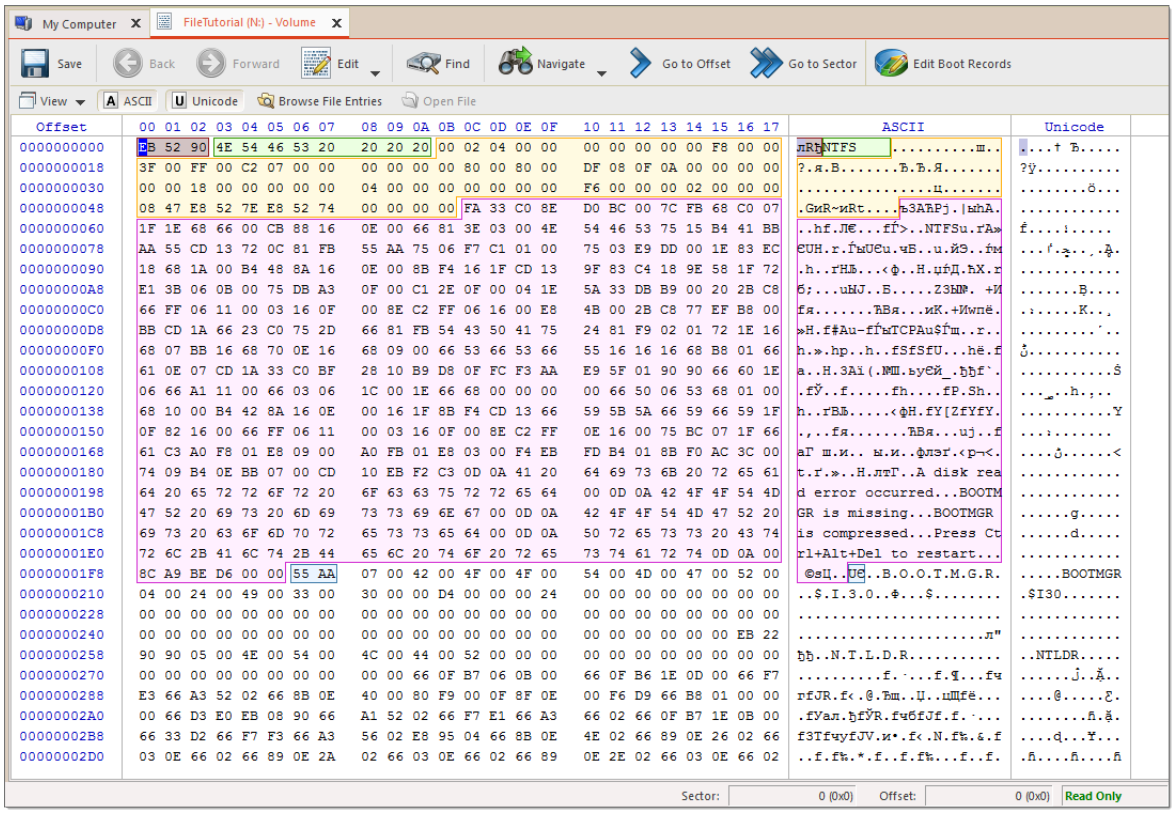


Figure 4: Disk Editor tab - editing volume example

To toggle between *Read Only* and *Edit* modes, do one of the following:

- From the Disk Editor toolbar, choose **Edit** > **Allow Edit** content.
- Right-click in the edit pane choose **Allow Edit** content from the context menu.

When you copy selected text from the edit pane to the clipboard, you may store it there in one of two formats using the following commands:

- **Copy** - selected data is copied into the clipboard as binary.
- **Copy Formatted** - selected data is copied as formatted text suitable to paste into a text editor.

Navigation

After you have opened an object with the Disk Editor, you may navigate by scrolling block by block, or by jumping directly to specific addresses. You may jump to disk system records such as the boot sector (primary and copy) or a partition table.

Read Navigation and information on page 17 articles for more.

Data selection

In order to select data in the **Disk Editor Area**, click and hold down the left mouse button and start dragging to select an area. The selected area background will be highlighted. Release the mouse to finish selecting. You can select an area bigger than will fit into the screen by dragging the mouse beyond the top or bottom edge of the hex editor window.

The alternative way to make a selection is to define a beginning and an end of the block. This method might be more convenient when a large area has to be selected in order to simply select data in a particular range. Move the cursor to the position where you want the selection to start and do one of the following:

- Select the menu command **Edit** > **Beginning of block** from the **Edit** menu in the toolbar.
- Right click and select **Edit** > **Beginning of block** from a context menu.
- Press **Ctrl+1**.

Move the cursor to the end of the desired selection and set the end of a selection in a similar way. If you need to select all the data, you can use the Select All command instead.

To apply massive changes to selection (block) use Filling a selection on page 21 feature.

Working with the clipboard

Select an area of data as described above and either select the command **Edit** > **Copy** or press **Ctrl+C**. The selected area will be copied into the clipboard in binary format. If you later want to insert it into a text editor, use the **Copy Formatted** command instead. It will copy data as a formatted text.

When you copy selected text from the edit pane to the clipboard, you may store it there in one of three formats:

- Binary – hexadecimal representation of selected data
- Text – text representation of selected data
- Display – formatted hexadecimal and text representation of selected data (as it appears in the editor)

Note: Please note that you can copy a maximum of 1MB of data into the clipboard.

Pasting data from the clipboard

If you copied data into the clipboard, you can paste it into a different place by moving the cursor to the position where you want the data to be copied. Use the command **Edit** > **Paste** or press **Ctrl+V**.

If you copied a text into the clipboard in a text editor, it will be pasted into the **Disk Editor** as text. Otherwise, the data will be copied as binaries.

Saving Changes

Unless stated otherwise, all modifications made in the Hex Editor are stored in memory. Changes are written to the drive when you click **Save**.

Related information

Navigation and information on page 17
Filling a selection on page 21
Edit physical disks on page 13
Edit logical drives on page 13
Edit files and file records on page 14

Edit physical disks

To navigate to the disk system records of a physical disk, click on the **Navigate** button in the toolbar. Depending on the partition scheme and contents of the physical disk you are editing, the **Navigate** menu will contain different options.

Navigating basic disks

After the **Go to Offset** and **Go to Sector** items there is a **Partition Table** menu item which allows jumping to sector 0 of a physical disk. As you jump to the partition table, a *Master Boot Record* template is automatically selected.

If the disk is not empty, the names of the partitions and their system areas will be in sub menus below the **Partition Table** menu item.

Navigating dynamic disks

For dynamic disks the following system areas are available for direct access:

- LDM Private Header
- LDM Primary TOC Block
- LDM Backup TOC Block
- LDM VMDB Block
- LDM KLog
- LDM First VBLK Block

After each access point a sector number is specified in the brackets.

Related information

Working with editor on page 11
Edit logical drives on page 13
Edit files and file records on page 14
Navigation and information on page 17

Edit logical drives

To navigate to the disk system records of a logical drive, click on the Navigate button in the toolbar.

Depending on the file system present in a logical drive, the navigation menu will have different access points.

FAT and FAT32 drives

- Boot Sector
- Boot Sector Copy (FAT32 only)
- FAT1
- FAT2
- Root Directory

NTFS drives

- Boot Sector
- Boot Sector Copy
- $MFT
- $MFT Mirror
- Arbitrary MFT record

HFS+ drives

- Volume Header
- Volume Header Copy

Ext2/Ext3 drives

- Superblock

Some of the access points when used automatically select a corresponding template. For example, if a boot sector access point is selected, a boot sector template is applied to the boot sector offset.

Browsing file records

When editing *volume* (*logical drive*) you also can navigate file records. To activate this feature toggle on **Browse File Entries** button in toolbar. By selecting file or folder in file's tree editor's pane will automatically repositions to beginning of file entry record. If recognized, file can be previewed in File Preview pane and Property pane will display file's most common attributes and properties.

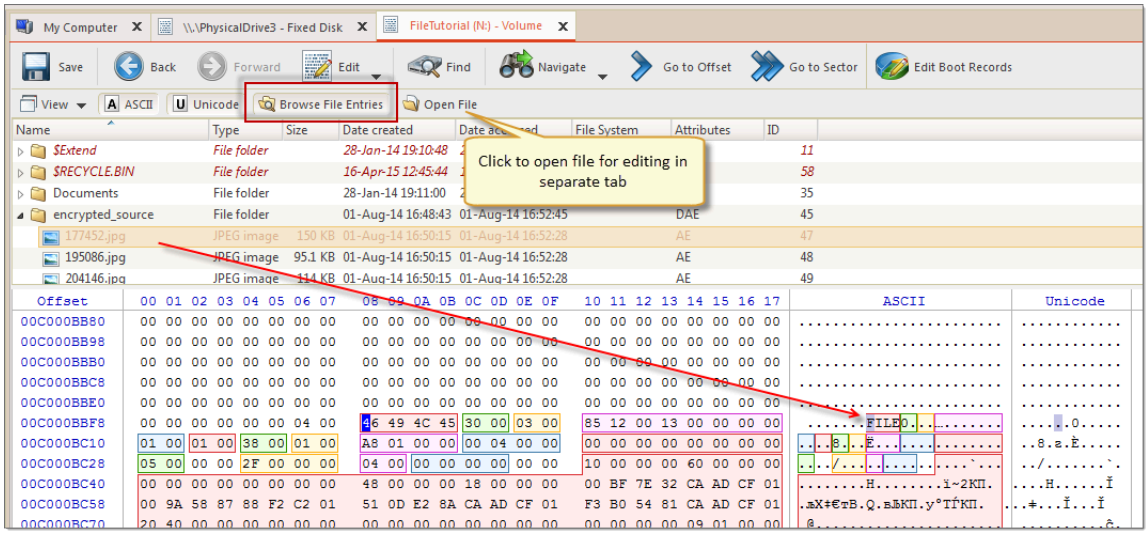


Figure 5: Browsing volume file entries

To open selected file in separate tab either click **Open File** button in toolbar or **Double click** on selected file for the same result.

Related information

Working with editor on page 11
Edit physical disks on page 13
Edit files and file records on page 14
Navigation and information on page 17

Edit files and file records

Open file in Disk Editor

To open file in Disk Editor use:

- Disk Explorer on page 6 in **My Computer** view mode or
- use File Browser on page 9 to navigate through drive contents

Select file and click **Open in Disk Editor** button to edit file's contents or use **Inspect File Record** command to edit file's record. You may use context menu for same result.

To open file in Disk Editor select it Scan Result view or Search Result view and click **Open in Disk Editor** button to edit file's contents or use **Inspect File Record** command to edit file's record. You may use context menu for same result.

Tip:

You can use the **Ctrl+H** shortcut to open any selected item in **Disk Editor**.

Editing file

Disk Editor allows to edit file in several view modes:

File view modes

Disk mode

File presented as raw data in context of physical data storage (disk)

Partition Mode

File presented as data on parent logical structure - partition or volume (logical drive)

File mode

Single file - seamless ray file contents.

File editing is the same as with any other editable object in Disk Editor. Read Working with editor on page 11 for more information.

Warning:

For safety reason, by default all objects are opened in Disk Editor as *Read Only* to prevent accidental modifications. In *Edit mode*, you can change content of the opened file or disk and all modifications are stored in memory. Changes are written to the drive when you click **Save**.

Warning:

As with any advanced tool, use extreme caution with the Disk Editor. Changes that you make may affect disk structure integrity. You must be certain that the changes you make are in line with correct data structures before you save changes.

Inspect file record

Information about file in File Table could be viewed for file by doing one of the following:

- Select file in browser and click **Inspect File Record** button in toolbar or use the same command from context menu;
- While editing file in Disk Editor click **Inspect File Record** button in view's toolbar
- While editing volume (logical drive) click **Browse File Records** in view's toolbar to open File Records navigation pane.

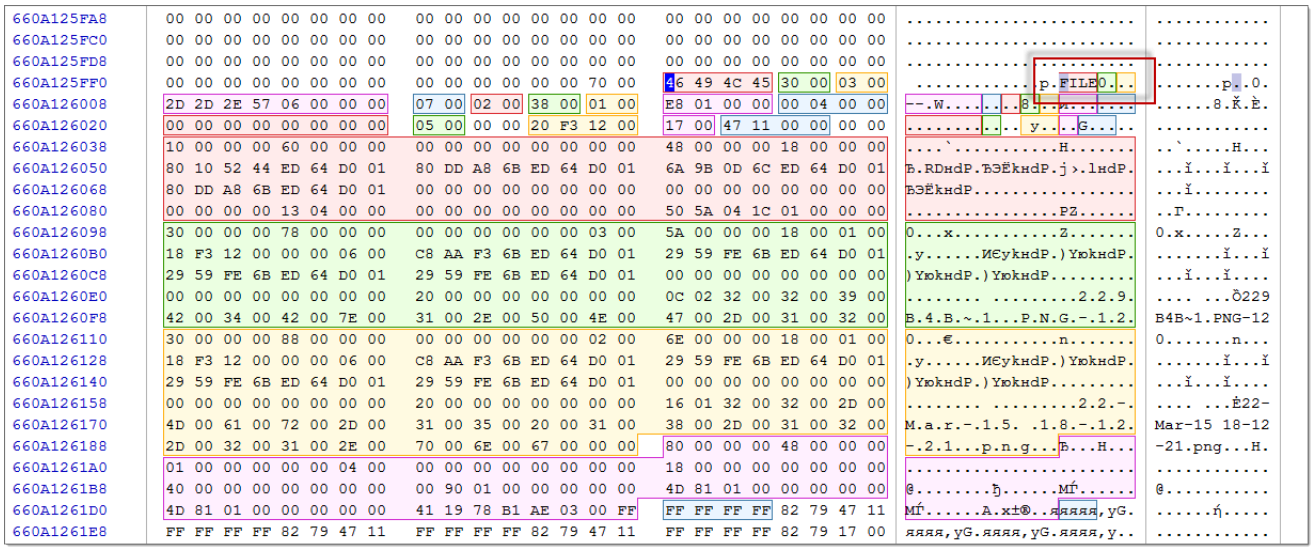


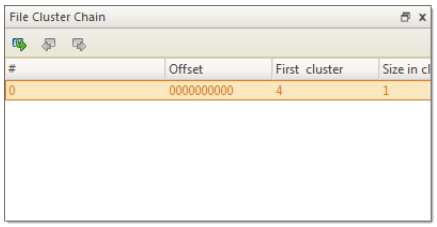
Figure 6: File record on NTFS

Use data templates to inspect file record. Depending on file system, they can be named as file record templates, directory entries or superblocks.

File Cluster Chain view

To open the File Cluster Chain View:

- from the Editor's toolbar, choose **View** > **File Cluster Chain**
- form main menu choose **View** > **Window** > **File Cluster Chain**



View options

Go to

Go to selected cluster of cluster chain. Same effect can by achieved by double-clicking on cluster entry in cluster chain list.

Go to Previous

Go to previous cluster chain in sequence

Go to Next

Go to next cluster chain in sequence.

Related information

File Browser on page 9
Working with editor on page 11
Edit logical drives on page 13
Edit physical disks on page 13
Navigation and information on page 17

Navigation and information

Basic Navigation

After you have opened an object with the Disk Editor, you may navigate by scrolling block by block, or by jumping directly to specific addresses. You may jump to disk system records such as the boot sector (primary and copy) or a partition table.

Use the **Navigate** button in the toolbar to jump to a specific area in the open object.

When you navigate to an access point through the **Navigate** menu or jump to a specific offset or sector, those addresses are stored in a stack. You can move backward and forward to the previous locations by using the **Back** and **Forward** commands located in the **Disk Editor Toolbar**.

The selections that appear depend on the type of object that you are editing.

Direct Navigation

No matter what object is opened for editing, the first two menu items in the **Navigate** menu will be **Go to Offset** and **Go to Sector**.

Read Move to offset on page 18 and Move to sector (cluster) on page 18 articles for more information.

Logical navigation

After you have opened an object with Hex Editor, you may navigate by scrolling block by block, or by "jumping" directly to specific addresses. You may jump to disk system records, such as the boot sector (primary and copy) or partition table. In a file's cluster chain list, you may jump to the first cluster of a continuous cluster chunk when working with a file.

To open the **Navigate** menu:

- In the Hex Editor toolbar, open the **Navigate** drop-down menu.
- Right-click in the editor pane and open the **Navigate** sub menu in the context menu.

The selections that appear depend on the type of object that you are editing.

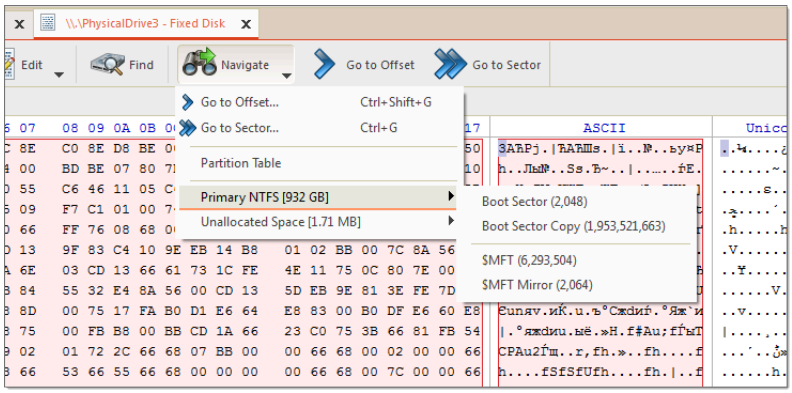


Figure 7: Example. Navigate Menu Selections

Use Property view and SMART Info for detailed information about subject attributes - Property views.

Related information

Working with editor on page 11
Move to offset on page 18
Move to sector (cluster) on page 18
Edit physical disks on page 13

Edit logical drives on page 13
Edit files and file records on page 14

Move to offset

The **Go to Offset** menu opens a dialog allowing specification of an exact location (offset) in the disk to jump to.

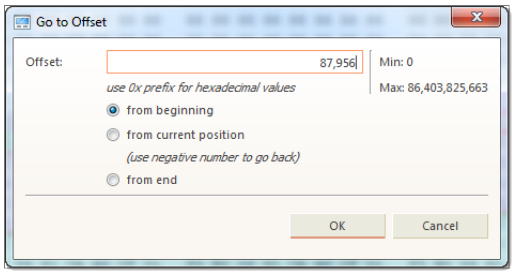


Figure 8: Go to Offset dialog

You can use both decimal and hexadecimal values, preceding hexadecimal values with 0x. For example, to specify location 512 as a hexadecimal number, enter 0x200. There are also options to specify an offset from the beginning, from the current position, or from the end.

Next to the offset edit field there are two labels specifying the minimum and maximum allowed vales for offsets displayed as decimal numbers.

You can also open this dialog directly by using the shortcut **Ctrl+Shift+G**.

Related information

Move to sector (cluster) on page 18
Navigation and information on page 17
Navigate a Physical Disk on page 19
Navigate a Logical Drive on page 20

Move to sector (cluster)

This command allows jumping to the beginning of a specified sector or cluster.

There are two edit fields in this dialog that allow entering a desired location either as a sector number or a cluster number.

The **Cluster edit field** is available only for logical disks and grayed out for all other objects.

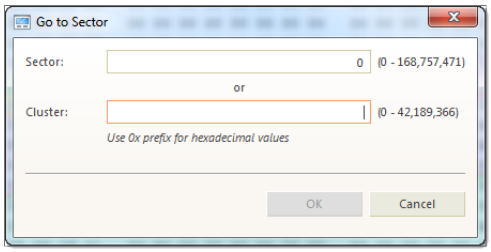


Figure 9: Go to Sector dialog

As with the offset dialog, you can also use both decimal and hexadecimal numbers.

Next to the edit field is the range of allowed values in brackets. Notice that not all sectors correspond to clusters, but every cluster corresponds to a particular sector.

You can enter either a sector value or a cluster value. Depending on which field is active, the dialog will use a sector or cluster. If you enter a number in the cluster edit field, a corresponding sector is displayed automatically.

You can also open this dialog directly using the shortcut **Ctrl+G**.

Related information

Move to offset on page 18
Navigation and information on page 17
Navigate a Physical Disk on page 19
Navigate a Logical Drive on page 20

Navigate a Physical Disk

To navigate to the disk system records of a physical disk, click on the **Navigate** button in the toolbar. Depending on the partition scheme and contents of the physical disk you are editing, the **Navigate** menu will contain different options.

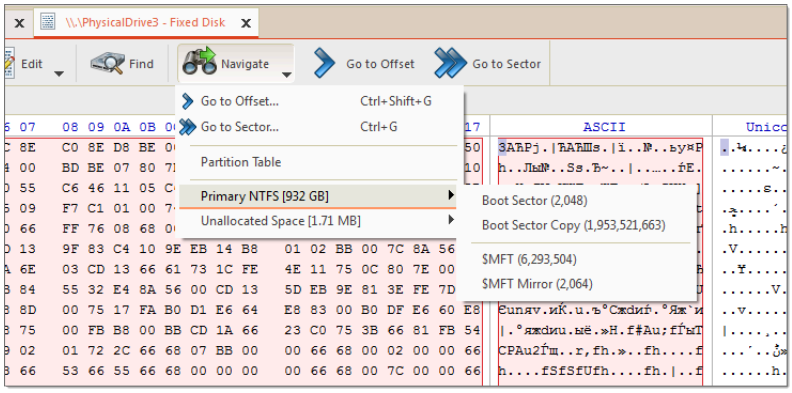


Figure 10: Example. Navigate Menu Selections

Navigating basic disks

After the **Go to Offset** and **Go to Sector** items there is a **Partition Table** menu item which allows jumping to sector 0 of a physical disk. As you jump to the partition table, a *Master Boot Record* template is automatically selected.

If the disk is not empty, the names of the partitions and their system areas will be in sub menus below the **Partition Table** menu item.

Navigating dynamic disks

For dynamic disks the following system areas are available for direct access:

- LDM Private Header
- LDM Primary TOC Block
- LDM Backup TOC Block
- LDM VMDB Block
- LDM KLog
- LDM First VBLK Block

After each access point a sector number is specified in the brackets.

Related information

Navigation and information on page 17
Move to offset on page 18
Move to sector (cluster) on page 18
Navigate a Logical Drive on page 20

Navigate a Logical Drive

To navigate to the disk system records of a logical drive, click on the Navigate button in the toolbar.

Depending on the file system present in a logical drive, the navigation menu will have different access points.

FAT and FAT32 drives

- Boot Sector
- Boot Sector Copy (FAT32 only)
- FAT1
- FAT2
- Root Directory

NTFS drives

- Boot Sector
- Boot Sector Copy
- $MFT
- $MFT Mirror
- Arbitrary MFT record

HFS+ drives

- Volume Header
- Volume Header Copy

Ext2/Ext3 drives

- Superblock

Some of the access points when used automatically select a corresponding template. For example, if a boot sector access point is selected, a boot sector template is applied to the boot sector offset.

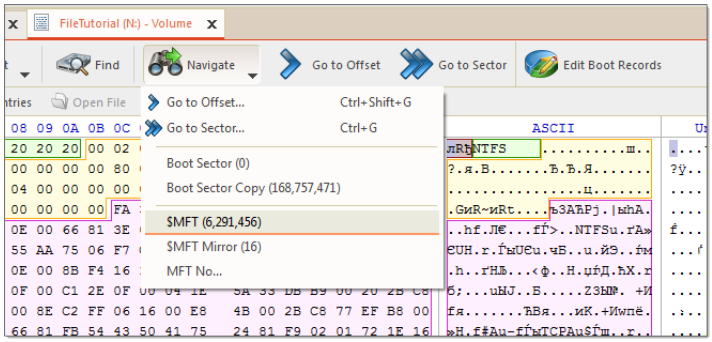


Figure 11: Example. Navigate Menu Selections

Browsing File Entries

When editing *volume* (*logical drive*) you also can navigate file records. To activate this feature toggle on **Browse File Entries** button in toolbar. By selecting file or folder in file's tree editor's pane will automatically repositions to beginning of file entry record. If recognized, file can be previewed in File Preview pane and Property pane will display file's most common attributes and properties.

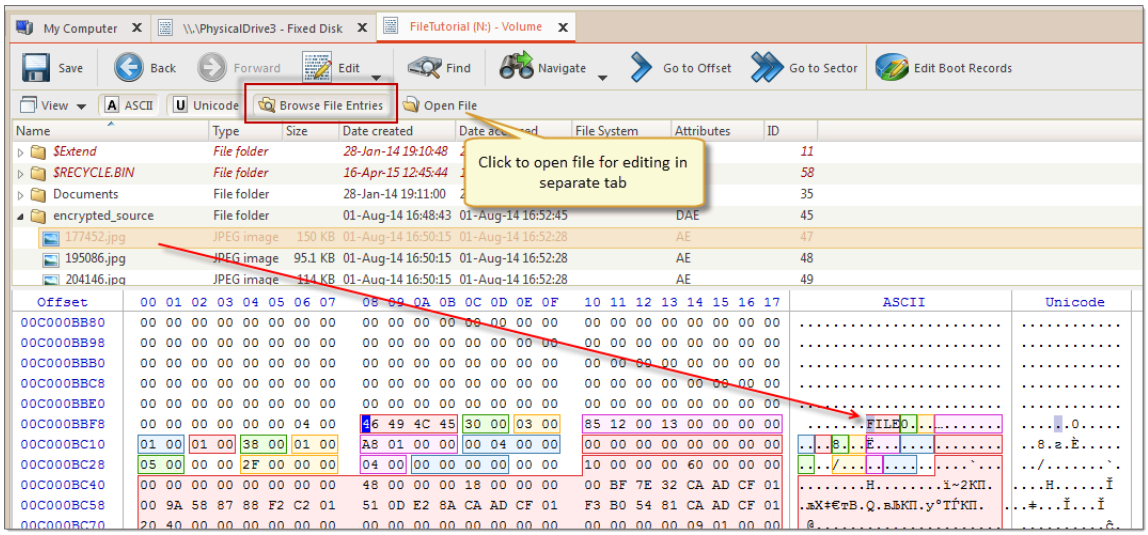


Figure 12: Browsing volume file entries

To open selected file in separate **Disk Editor** either click **Open File** button in toolbar or **Double click** on selected file for the same result.

Related information

Navigation and information on page 17
Move to offset on page 18
Move to sector (cluster) on page 18
Navigate a Physical Disk on page 19

Filling a selection

You can fill a selection with an arbitrary text or binary data. Make a selection first, then right click **Edit** > **Fill block**. The Fill Block dialog allows entering either text or hex value patterns which will be used to fill the selection. Patterns are used in a loop until the whole selection is filled. For example, if you need to fill a selection with 0 bytes, just enter 00 into the Hex values edit field. If you want fill it with an '**ERASED**' pattern, enter it as a text and it will be repeated as many times as necessary to fill the block.

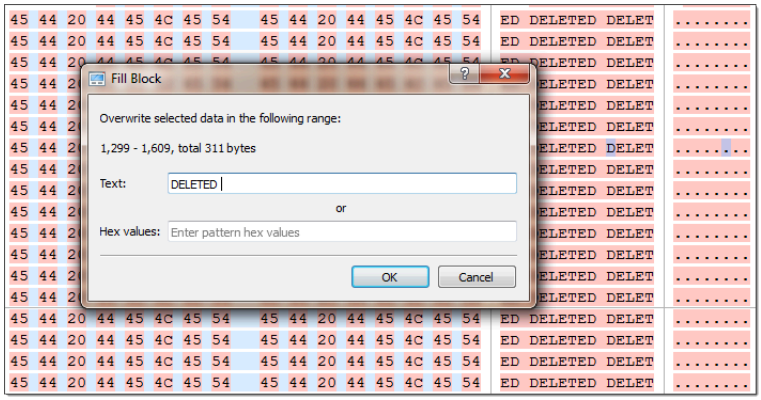


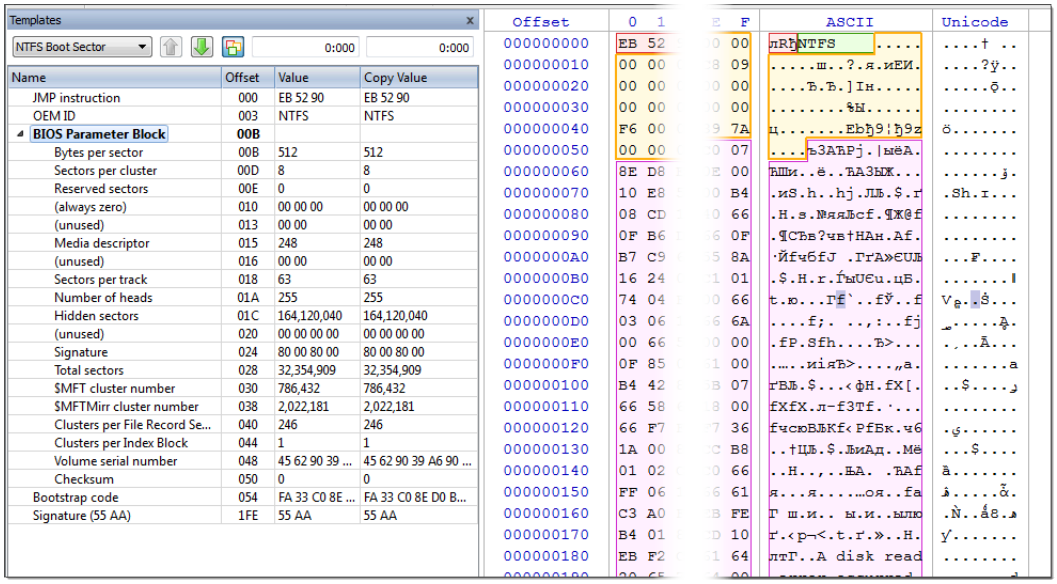
Figure 13: Fill Block dialog

Related information

Edit physical disks on page 13
Edit logical drives on page 13
Edit files and file records on page 14
Navigation and information on page 17

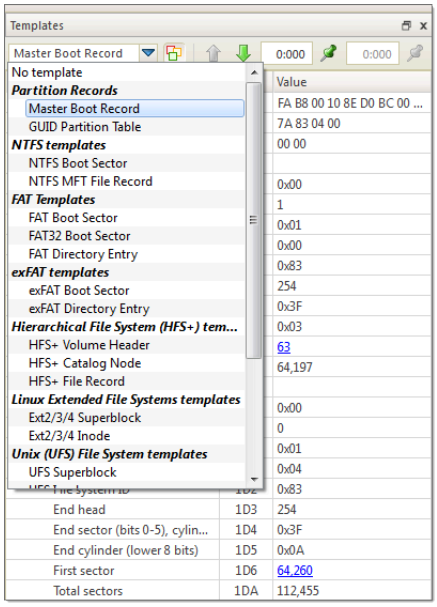
Using Templates

You can edit system records (like boot sectors, MBR, MFT etc.) by using a template tool window. Template window is a small dockable window normally located to the left from main Disk Editor editing area. If it is not visible, you can turn it on by selecting toolbar menu **View** > **Templates**.



Applying a template

In order to apply a template to the desired offset, move the cursor to the location and use **Edit** menu command **Set Template position**. You can select this command either from Edit toolbar menu or from a context menu. The next step select a required template from the list box with template names in the toolbar of templates window.



When you are jumping to particular system areas using **Navigate** menu, the corresponding template might be applied automatically. This is true for templates like boot sectors, MBR or MFT record but not all access points have a template associated with them.

The following templates are supported:

Partition records

- Master Boot Record (MBR)
- GUID Partition table

NTFS templates

- NTFS Boot Sector
- NTFS MFT File Record

FAT templates

- FAT Boot Sector
- FAT32 Boot Sector
- FAT Directory Entry

exFAT templates

- exFAT Boot Sector
- exFAT Directory Entry

Hierarchical File System (HFS+) templates

- HFS+ Volume Header
- HFS+ Catalog Node
- HFS+ File Record

Linux Extended File System templates

- Ext2/Ext3/Ext4 Boot Sector
- Ext2/Ext3/Ext4 Inode

Unix File System (UFS) templates

- UFS Superblock
- UFS Inode

B-tree (BtrFS) File System templates

- BtrFS Superblock

Logical Disk Manager (LDM) templates

- LDM Private Header
- LDM TOC
- LDM VMDB
- LDM Klog
- LDM VBLK

As you edit data in Hex, ASCII or Unicode pane or in Templates window, modified data is fully synchronized between views. After each modification a template view is recalculated giving you an up-to-date interpretation of data.

Template Copy

The following templates have their copy:

- NTFS Boot Sector
- FAT32 Boot Sector
- HFS+ Volume Header
- Ext2/Ext3 super block
- LDM Private Header
- LDM TOC Block

In this case template window will have an additional column named *Copy Value* which contains the data from the copy record. Template copies are useful to compare record located in different locations using the same pattern, for example to compare a boot record with its copy.

In case of *Copy* template its location is set separately from a main record using the same pattern. If the main template and its copy are intersecting, the copy template data will be shown in template window but not highlighted in the main edit area.

Setting template position

In order to set a template position or change an existing one move the cursor to desired location and use **Edit** menu command **Set Template position** (or **Set Template Copy Position** for its copy).

Navigating to a system area which has an attached template using **Navigate** menu also changes template position.

In order to facilitate the movement between records located in sequence, use arrow buttons located in the template window toolbar next to the templates list. For example, if you are editing or viewing an MFT record you can easily move to the next or previous record using those buttons.

Another way to set a template position is to enter new offset directly into template offset edit field in the template window toolbar. One of those fields are used for entering an offset of the main record and another is for its copy. The format of offset used in offset field is <sector:>:<sector offset>. You don't need to specify sector offset if you want to move to the beginning of the sector. For example, you can simply enter 100 to go to sector 100 and template offset will be shown as 100:0, but if you need to specify 128 byte in sector 100, you have to enter 100:128.

Highlighting template fields

By default all individual fields of template record are highlighted in Disk Editor main area (in hexadecimal and ASCII columns only). This coloring highlighting can be disabled by clicking Toggle template fields coloring button in template window toolbar next to arrow buttons.

The colors used by template coloring are arbitrary and have no specific meaning, their main purpose is to make separate fields visible and distinguish from each other. Actually, a palette of several colors is chosen and colors are used in a circle. When you select a field in the template window, the current field is also highlighted in hex editing area with bold field frame.

When you move a mouse cursor above colored field in editing area, the name and value of the corresponding field is also shown in a tooltip.

Navigating around template fields

You can set the cursor (current position) to a particular field in a template by double clicking it. If you double click in *Name*, *Offset* or *Value* column, the position inside the main record is selected, but if you click inside *Copy Value* column, the navigation is performed to the field in template copy.

Please note, that in Edit mode double clicking inside of *Value* or *Copy Value* starts editing of the field instead of navigating to that field.

Editing using template

Double click in the *Value* or *Copy Value* column to start editing the field (make sure that **Allow Edit Content** is enabled).

Some of the fields are edited according to the mask and will not allow to enter invalid values. For example, you cannot enter the number bigger than 65535 when editing a 2-byte field or invalid date when editing a date.

To exit the editing of the field with saving the result of edit, press Enter or click to another field. To exit editing without saving the result and revert to original value, press **Esc**.

Some of the templates fields depend on other fields. When a template is selected, an initial parsing occurs. If some of the fields contain invalid values, the further parsing of the record might be not possible and parsing will be stopped at this point, resulting in incomplete record. As an example lets take an MFT record. The record header is always parsed, but if it contains invalid fields or update sequence, attributes will not be

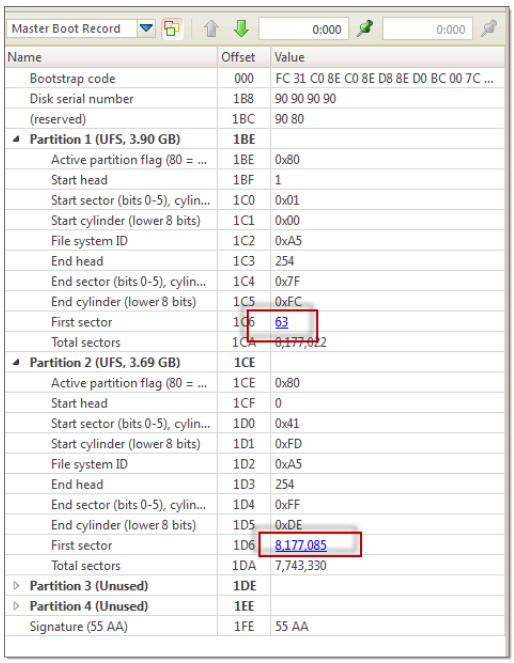
parsed. The same is true when parsing an attribute - if an error occurs, the further parse is canceled and no subsequent attributes are added to the record.

Furthermore, the whole set of fields for the template might depend on some field values. For example, FAT Directory Entry template will show a Short File Name Entry fields or Long File Name depending on the value of the flags.

Hyperlinks in templates

Many templates contain hyperlinks allowing navigate easily to important data points.

For example, MFT records contain links to first cluster in data runs and MBR provides links to partitions.



Disk Editor tools and views

Active@ Disk Editor delivers several tools for advanced users:

Data Inspector on page 25
Tool-view interpret currently selected data to several most used data types.

File preview on page 27
Allows to preview content of a file. Supports basic image formats and registered document types, such as MS Office, PDF's etc.

File cluster chain on page 26
Provides advanced navigation through file structure.

Active Bookmarks on page 28
Provides ability to mark certain locations on edited subject to faster access and navigation.

Searching in Disk Editor on page 31
Enhanced search within edited content.

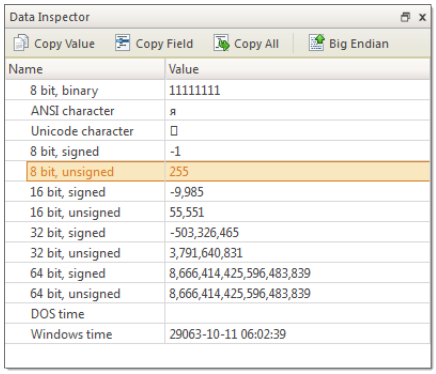
Related information

Property views

Data Inspector

The Data Inspector is a small viewing tool that provides the service of “inspecting” (or interpreting) data currently selected in the edit pane. The Data Inspector lets you view the type of data you have selected. This can help you interpret data as displayed in Disk Editor.

To open the Data Inspector, from the **Disk Editor toolbar**, choose **View** > **Data Inspector**;



View options

Copy Value

Copy value of selected field to clipboard.

Copy Field

Copy entire selected field (value and field name) to clipboard.

Copy All

Copy all name and value fields in a view to clipboard.

Big Endian

Toggle between *little endian* and *big endian* value representation.

Use view context menu to execute these commands for selected item (field).

The Data Inspector window is dockable and its location can be changed by clicking on the window title and dragging it to the new one. If the Data Inspector window is sharing its space with other tool views, you can change its relative position by left clicking and dragging the window tab. You can close the window by clicking on the [**X**] button in the top right corner of the window and reopen it again using the **View** menu in the **Disk Editor Toolbar**.

Related information

Disk Editor tools and views on page 25

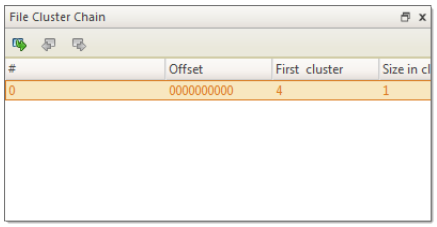
File cluster chain

File cluster chain is one of the essential approach to analyze file data integrity and file recovery/ To help navigate through the content of an open file, file's cluster chain, shown sequence number, offset and size of each chain, is displayed in File Cluster Chain view.

File Cluster Chain view

To open the File Cluster Chain View:

- from the Editor's toolbar, choose **View** > **File Cluster Chain**
- form main menu choose **View** > **Window** > **File Cluster Chain**



View options

Go to

Go to selected cluster of cluster chain. Same effect can by achieved by double-clicking on cluster entry in cluster chain list.

Go to Previous

Go to previous cluster chain in sequence

Go to Next

Go to next cluster chain in sequence.

Related information

Disk Editor tools and views on page 25

File preview

In **Active@ Disk Editor** provides ability to preview files along with editing of its content and explore volume entry records.

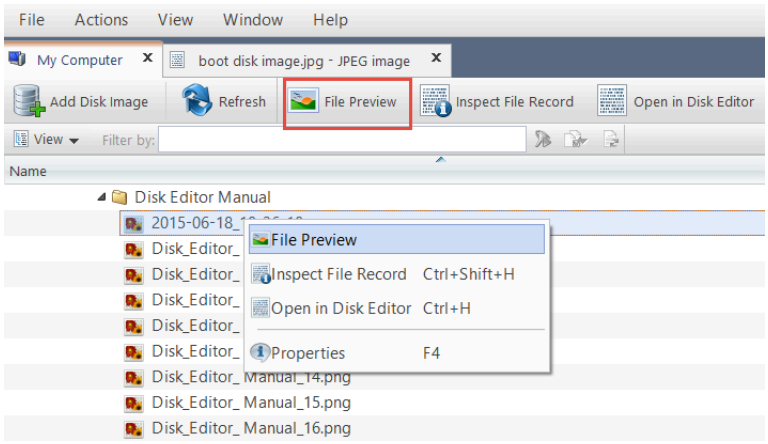


Figure 14: Preview selected file in Disk Browser

To open the File Preview panel from any view, do one of the following:

- Hold **Ctrl** key and double-click on file.
- Right-click on file and click **File Preview** from the context menu.
- Select a file and click **File Preview** from the main toolbar.

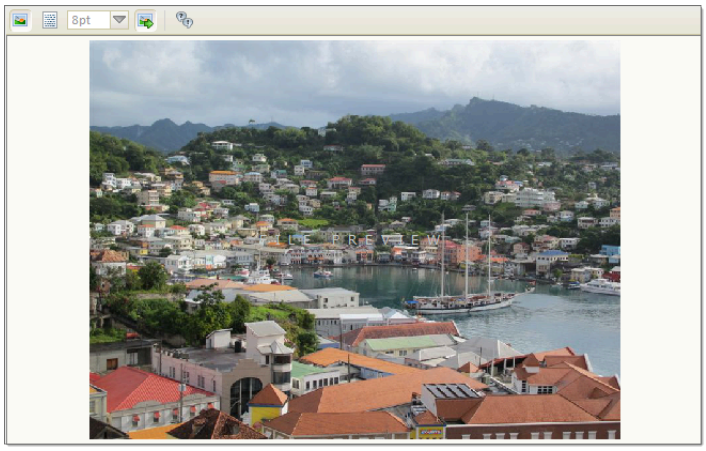


Figure 15: Preview selected file in Disk Browser

View options

Preview mode

Default preview mode can be selected either as *Hexadecimal* or *Rendered*, in which case file will be shown as an image (for graphics files) or rendered by one of the registered file previewers.

Font size

Select size of the font for hexadecimal mode;

Auto-follow

With this option **on** files, selected in context source, will be previewed automatically. Toggle this option **off** if for any reason file preview causes delays in file navigation.

Info

In this mode, all registered previewers and supported graphics formats in current system will be shown.

Note: If the preview file is not available then it appears in hexadecimal or text mode.

Related information

File Browser on page 9

Active Bookmarks

Bookmarks allow you to save the current cursor location and quickly return to it later on. You may also give a name to a bookmark to make orientation easier.

Bookmarks are shown in the tool window called Bookmarks. If the Bookmarks window is closed you can open it using the menu **View** > **Bookmarks**.

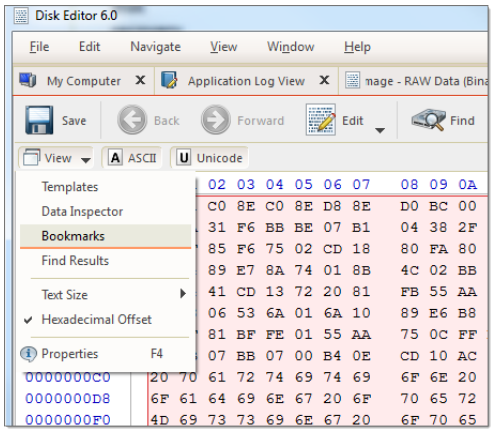


Figure 16: Open bookmarks view

Bookmark view

All bookmark for currently edited object are listed in Bookmark view. Bookmark will be saved for next session use if edited object is saved or left open before application exit.

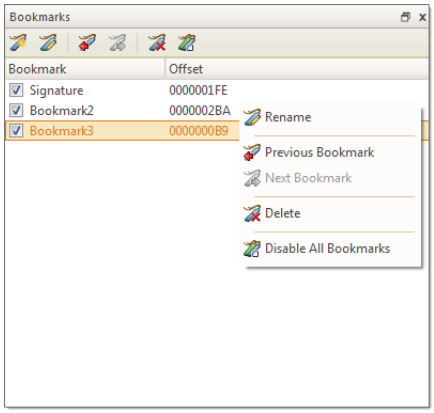


Figure 17: Bookmarks and Bookmarks view

View options

Toggle bookmark

Add or remove bookmark at current cursor position

Rename

Rename selected bookmark

Go to previous

Move (jump) cursor to previous bookmark

Go to next

Move (jump) cursor to next bookmark

Delete

Delete selected bookmark

Disable all bookmarks

Disable bookmarks, thus they will be ignored in bookmark's shortcut navigation.

Tip: Use context menu for selected bookmark for the same set of commands as in view's toolbar.

Placing and removing a bookmark

Move cursor to position of interest in the Editor View and press **Ctrl+F2** in order to add a bookmark or toggle a **Toggle a Bookmark** button in the Bookmark view toolbar. Alternatively, you can right click in the hex editor and select a command from a context menu. The bookmark position is shown with a light blue box in the Disk Editor and also added to the list of bookmarks in the Bookmarks view.

To remove a bookmark, press **Ctrl+F2** while having the cursor over the position of that bookmark. You can also remove a bookmark from the Bookmarks view by selecting a bookmark in the list and clicking **Delete** button in a toolbar. The delete function can also be selected from a context menu.

Going to a bookmark

If you have defined bookmarks, pressing **F2** will move your current position to the next enabled bookmark in the list.

You can also right click a bookmark and select the **Next** bookmark command from a context menu. Another option is to double click a bookmark name in the Bookmarks window.

Editing bookmarks

Bookmarks are named automatically when they are placed. You can rename a bookmark in the Bookmarks window to give it some meaningful name. To do so make a single mouse click on the bookmark name and edit it. Press **Enter** to accept your changes or **Esc** to cancel editing and revert to the original name. You can also rename a bookmark by right-clicking on it and selecting the **Rename** command from a context menu.

All bookmarks are highlighted in Editor View view for easy navigation

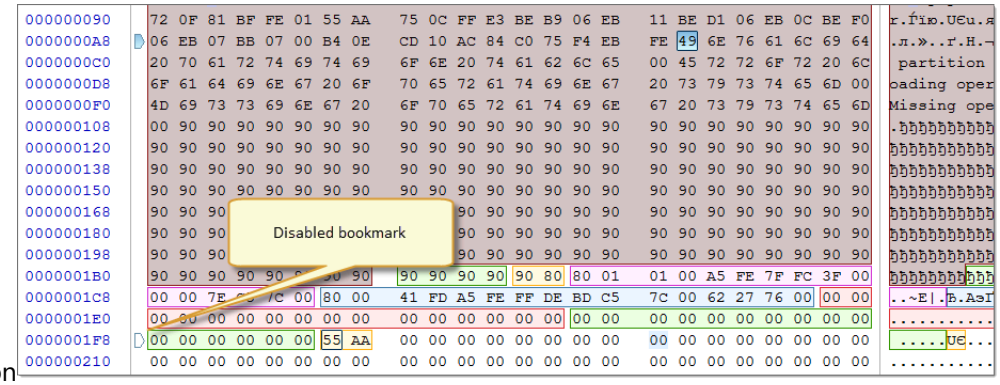


Figure 18: Bookmarks in Editor view

Sometimes instead of deleting a bookmark it is useful to temporarily disable it. A disabled bookmark will not be counted when moving to the next bookmark. Uncheck a bookmark in the Bookmarks window to disable it. To disable all bookmarks at once click **Disable** all bookmarks in a toolbar or select this command in a context menu.

Related information

Disk Editor tools and views on page 25

Searching in Disk Editor

To search text or byte sequence in Disk Editor view :

- Click **Ctrl+F** shortcut key or
- Use **Find** button in Disk Editor's toolbar then Find text dialog will appear.

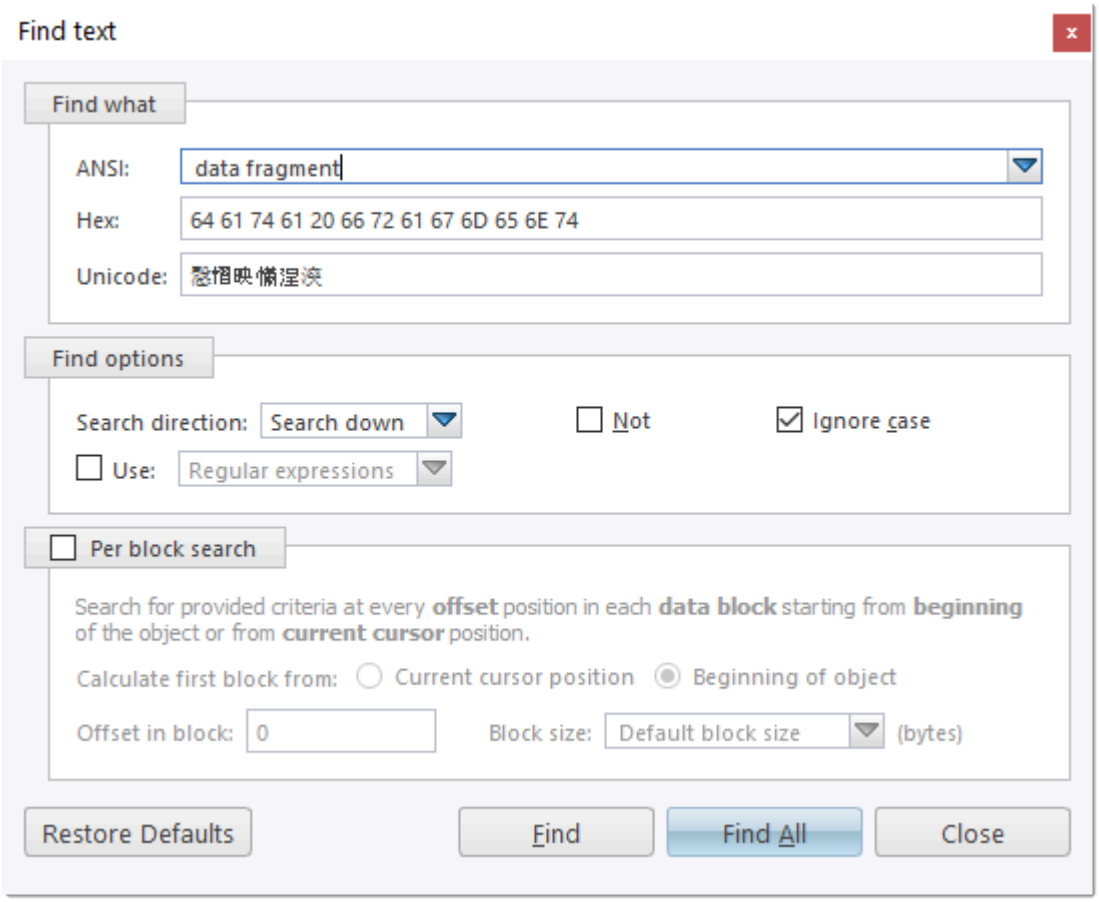


Figure 19: Dialog Find

Dialog options

Find what

Search pattern to find. Required. Can be set in one of the following formats:

- ANSI - text pattern, Regular expressions and wildcards can be used. History of ANSI search patters is preserved for next sessions and can be selected from drop-down list.
- HEX - search pattern in **hexadecimal** format.
- Unicode - search pattern in **Unicode** format.

Note: When search pattern is entered in one of the find what text fields, the other two related fields will interpret entered value in correspondent format.

Find options

Regular expressions and wildcards can provide even greater search capabilities.

Search direction will specify search direction from the current cursor position.

Not option search for characters that do not correspond to the **Find what** parameter.

Ignore case disables case-sensitivity in text search

Per block search

When this option is on, then search applies on per block fragments of context object. This method could be useful to search for repeated pattern, for instance at certain position (offset) in each and every sector (block).

Find command will initiate search process and will pause at first search result entry. Use **Next** button on dialog or **F3** keyboard shortcut to continue paused search.

When using **Find All** command, list of all search entries will appear in Find Results view. Use this list to navigate between search result entries (if any).

Browsing search results

After search completed, result entries (if any) are listed in Find Results view, grouped in subsequent search results, with offset (address) and short preview snippet. To focus on individual search result entry double-click on it in list or use **Goto** button in view's toolbar.

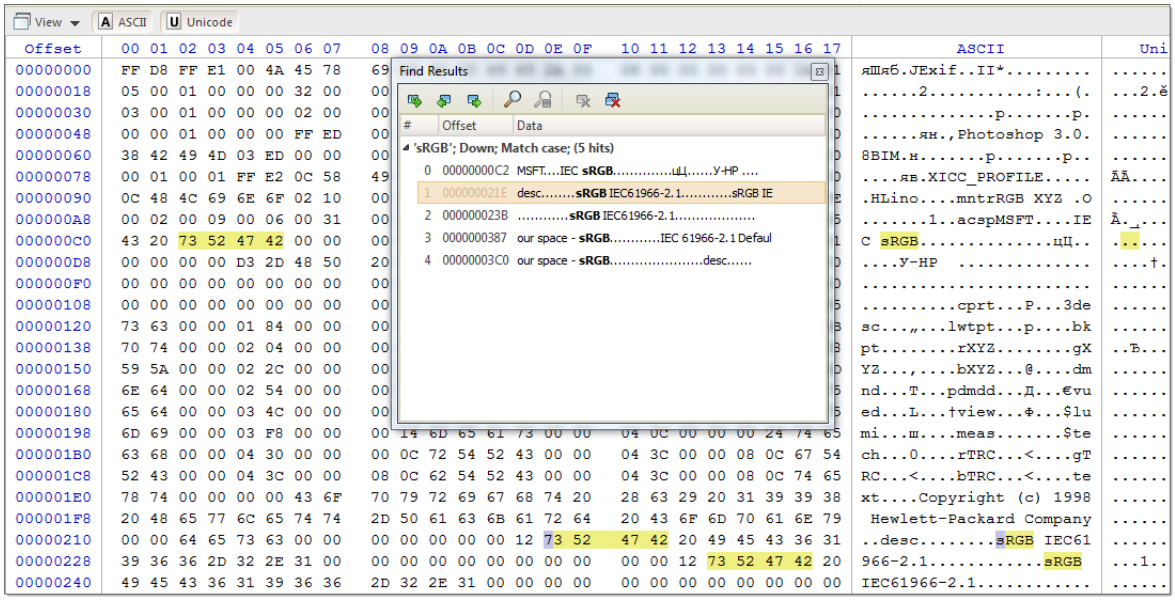


Figure 20: Find results in Disk Editor

All search results are highlighted in Disk Editor view.

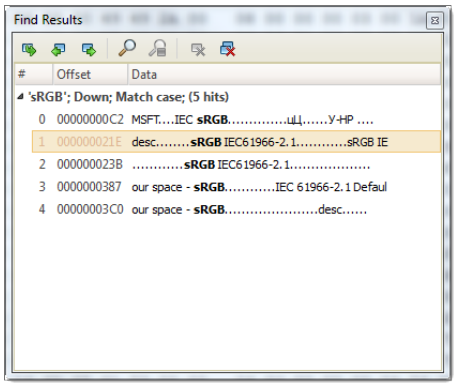


Figure 21: Find Results view

View actions

Goto location

Move cursor and position Disk Editor view on selected search result

Goto Previous

Move cursor to previous search result

Goto Next

Move cursor to next search result

Find

Open Find dialog for a new search

Stop

Terminate current search process

Remove

Remove selected search result or search group from list

Remove All

Clear search result list

Use context menu in Find result view to interact with each search entry individually.

Related information

Disk Editor tools and views on page 25
Active Bookmarks on page 28

Disk Management

Active@ Disk Editor is advanced disk utility and allows you to perform disk partitioning tasks, such as creating partitions and volumes, formatting them, and assigning drive letters. Initialize raw disk, edit partition tables and more.

Most of these changes to disk partitioning are recorded in dedicated backup files thus at any time these changes could be rollback at certain point. See Rollback partition changes on page 40 for more information.

The main features of Partition Manager are:

- Initialize new disk (physical device) on page 34
- Create partition on page 36
- Format partition on page 39
- Resize a partition or logical drive (volume) on page 38

Initialize new disk (physical device)

Physical Disks Initialization

To make disk accessible for application it needs to be initialized first by one of the following partition style:

- Master Boot Record (MBR);
- GUID Partition Table

To initialize physical disk proceed as follows:

1. Select disk to initialize

   In Disk Explorer select not-initialized Disk (Physical Disk).

2. Open the Initialize Disk dialog

   - From the Disk Explorer toolbar click **Initialize** button or use command **Actions** > **Initialize** ... from main menu;
   - Right-click the selected item and click **Initialize...** command from the context menu.

   Confirm disk selection and other options in opened dialog.

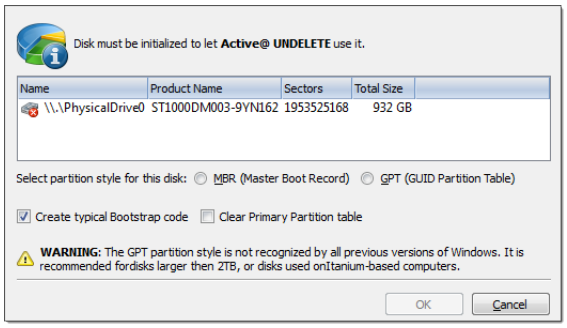


Figure 22: Initialize Disk dialog

Dialog options

Partition style

Select either *MBR* (Master Boot Record) or *GPT* (GUID Partition Table) partition style.

Note: GPT partition style is not supported by older versions of Windows. It is recommended for disks larger then 2TB. For all other purposes we recommend to use MBR partition style

Create typical bootstrap code

Default generic bootstrap code will be written if this option is on.

Clear primary partition table

Primary partition table records will be cleared.

Warning: It is highly recommended to not clear primary partition table in case of restoring deleted or damaged disk partitioning.

3. Click **OK** to complete disk initialization

After disk initialization it should be visible and accessible in Disk Explorer for other actions, such as Create partition on page 36 and more.

Related tasks

Convert MBR and GPT disks on page 41

Partition management

Active@ Disk Editor provides essential functionality to handle disk partitioning under windows environment, such as:

- Create partition on page 36
- Change partition attributes on page 38
- Resize a partition or logical drive (volume) on page 38
- Format partition on page 39

One of the unique feature of **Active@ Disk Editor** is Rollback partition changes on page 40 - ability to revert any if the actions mentioned above.

Related tasks

Initialize new disk (physical device) on page 34
Physical Disks Initialization

Related information

Disk editing on page 41

Create partition

To create new partition (*logical drive* or *volume*):

1. Select partition location

   In Disk Explorer select a disk (*physical device*) or *unallocated space* node.
2. Open the **Create New Partition** dialog
   - From the toolbar click **Create New Partition** button or use command **Actions** > **Create New Partition...** from main menu.
   - Right-click the selected item and click **Create New Partition** command from the context menu.

3. Adjust dialog options

Use sliders to specify partition boundaries - offset and size. Mouse click on unallocated space will select it to utilize all space available.

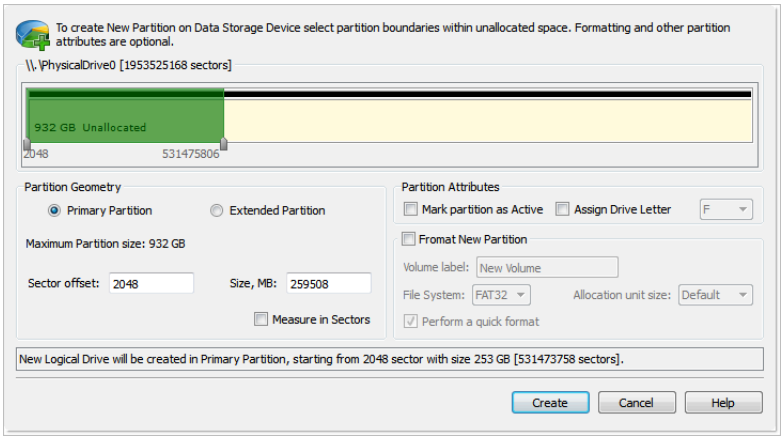


Figure 23: Create Partition dialog

Partition Geometry

Primary or Extended

Partition can be created as *Primary* partition (of number of available Primary partitions are not exceeded) or as *Extended* partition.

Sector Offset

First sector of created partition. It can be set exact by numerical value entered in text box or by moving left slider in **Device View** control;

Partition Size

Partition size can be set in megabytes or in sectors, depending on state of **Measure in Sectors** check box;

Partition Attributes

Mark Partition as Active

Newly created partition will be set as *Active Partition*;

Assign Drive letter

For Primary Partition or Logical Drive on extended partition drive letter can be assigned from the list of available in the system drive letters;

Format Partition [optional]

Volume label

Text label of partition (disk). This field can be blank

File System

Select file one of the supported file systems: FAT, FAT 32 or NTFS.

Unit Allocation Size

Depending on selected file system and total partition (disk) size available allocated unit size may be different. Default value of unit size is recommended.

4. Click **Create** button to create new partition

After partition created, it should appear in Disk Explorer available for other actions like formatting.

Related tasks

Format partition on page 39
Change partition attributes on page 38
Edit partition table

Change partition attributes

To change *logical drive* (*partition*) attributes:

1. Select volume

   In Disk Explorer, select a *logical drive* (*partition*) node.
2. Open the Partition Attributes dialog

   - From the Disk Explorer toolbar click **Change Attributes** button or use command **Actions** > **Change Attributes** from main menu;
   - Right-click the selected item and click **Change Attributes** from the context menu.

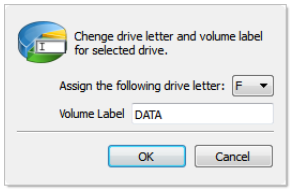


Figure 24: Create Partition dialog

   Select new drive letter from drop-down list of available drive letters and enter volume label if necessary.
3. Click **OK** to complete changes

After command is complete, volume item should appear in Partition Manager with new attributes.

Related tasks

Create partition on page 36
Format partition on page 39

Resize a partition or logical drive (volume)

Existing logical drive (volume) can be extended to use unallocated space available right after that partition or shrunk to utilize unused space. To resize *Logical Drive* (*Partition*):

1. In Disk Explorer select a *Logical Drive* (volume) node.
2. Open the **Resize Volume** dialog:

   - From the toolbar click **Resize** button or use command **Actions** > **Resize...** from main menu.
   - Right-click the selected item and click **Resize...** command from the context menu.

3. Define new partition size

   Using Resize Volume dialog to define new partition (volume) size

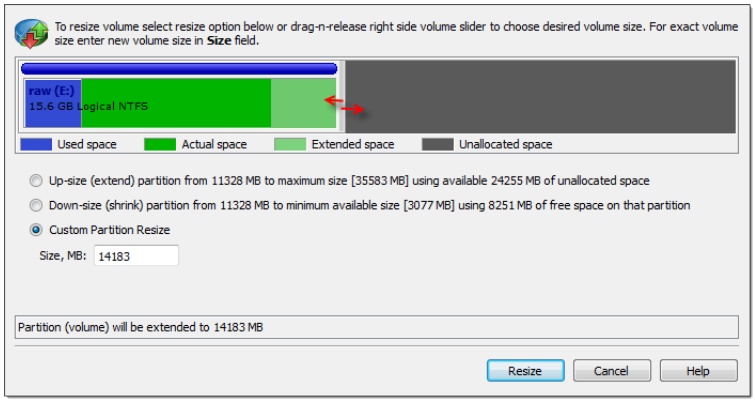


Figure 25: Resize Volume dialog

Dialog options

Resize options

Use radio buttons to expand to use maximum space available or shrink to last used cluster. Use **custom** option to define exact new size of partition.

Note:

Use device control drug'n'release feature to set approximate partition size.

Warning:

Logical drive (volume) resize is not part of Rollback feature - all changes are final and can not be undone.

4. Click **Resize** to complete changes

Related tasks

Change partition attributes on page 38

Related information

Partition management on page 35

Format partition

To format *volume* (*partition*):

1. Select volume

   In Disk Explorer select a *Logical Drive* (Partition) node.

2. Open the Format Partition dialog

   - From the toolbar click **Format** button or use command **Actions** > **Format...** from main menu.
   - Right-click the selected item and click **Format...** command from the context menu.

3. Adjust dialog options

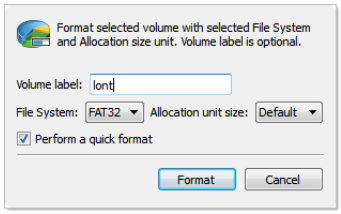


Figure 26: Format Partition dialog

Dialog options

Volume label

Text label of partition (disk). This field can be blank

File System

Select file one of the supported file systems: FAT, FAT 32 or NTFS.

Unit Allocation Size

Depending on selected file system and total partition (disk) size available allocated unit size may be different. **Default** value of unit size is recommended.

4. Click **Format** button to start formatting process

⚠ **DANGER:**

All data on formatting Logical Drive (partition) will be lost! Backup all your valuable data before formatting.

When formatting is complete, volume item should appear in Disk Explorer with new attributes and file system.

Related information

Partition management on page 35
Rollback partition changes on page 40

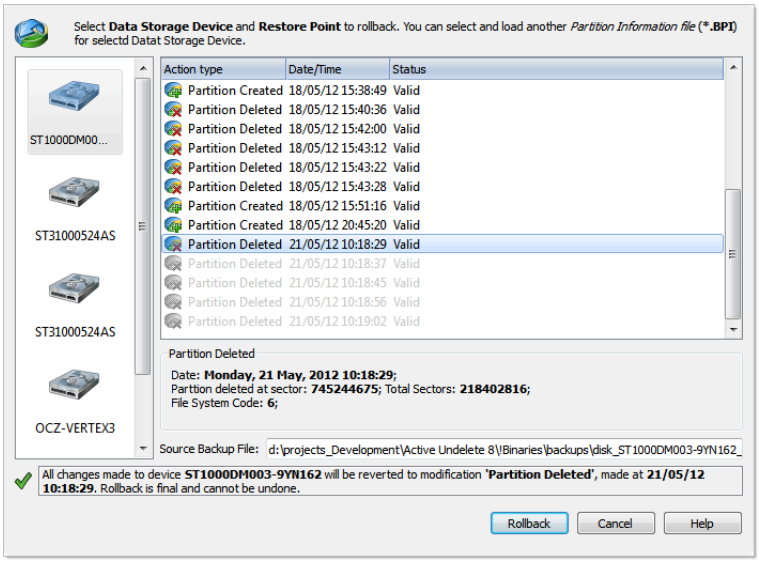
Rollback partition changes

Some critical partition layout changes made to a physical device are backed up by default. Users can rollback these changes at any point by using the **Rollback Partition Changes** tool. These changes are:

- Initialize disk
- Create partition
- Format partition
- Delete partition

To open the Rollback Partition Changes dialog, do one of the following:

- From the Tools menu, choose the **Rollback Partition Changes** command.
- From the Tools tab in Command Bar, choose the Rollback Partition Changes command.
- For a selected physical device (disk) node use the context menu **Rollback Partition Changes** command.



To rollback changes made to a physical device, select a restore point in the chronologically ordered list and click the **Roll Back** button to complete the changes.

Related information

Partition management on page 35
Disk editing on page 41

Disk editing

Disk editing in Disk Explorer includes:

- Convert MBR and GPT disks on page 41

These features available not only from Disk Explorer itself, but also from any other view that uses partition of hard drives in a same manner as in Disk Explorer view.

Related tasks

Initialize new disk (physical device) on page 34
Physical Disks Initialization

Related information

Partition management on page 35

Convert MBR and GPT disks

For freshly initialized (empty) physical disk partition style can be changed at any time from *MBR* to *GPT* or from *GPT* to *MBR*.

To convert partition style:

1. Select disk in Disk Explorer
2. Change partition style
   - Use **Actions** > **Convert to MBR [GPT]** command from main menu or
   - Use **Convert to MBR [GPT]** command from context menu

If conversion is successful, then device partitioning will be changed and property label will indicate new attribute.

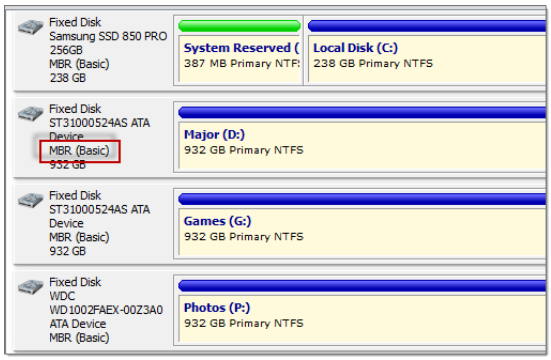


Figure 27: Disk partitioning style

Related tasks

Initialize new disk (physical device) on page 34
Physical Disks Initialization

Related information

Disk Explorer on page 6
Disk editing on page 41

Appendix

Preferences

Active@ Disk Editor Preferences dialog is the central location where **Active@ Disk Editor** features and settings can be configured.

To open **Preferences** dialog:

- From main menu choose **Tools** > **Preferences...**

  or
- Press **F10** **F2** keyboard shortcut at any time

Preferences dialog divided into several **Active@ Disk Editor** sections:

Preferences allow to configure all the settings needed for the application proper operation.

General Settings

The General Settings section allows to configure general preferences as well as the application's visual and sound representation.

Device Control Layout

These settings control visual disk behavior in Disk Explorer on page 6 and allow to Show or Hide a System Disk and devices which are not ready (offline).

Default serial number detection method

Select how **Active@ Disk Editor** retrieves the disk serial number by default. Values are: **SMART**, **IOControl** & **WMI**.

Local devices initialization

Select which types of devices appear in **Active@ Disk Editor** by default: **Dynamic Disks, Fixed disks**, **Removable disks**, **CD/DVD/BD** and **Floppies**.

Computer ID

Configure how the **Active@ Disk Editor** workstation is identified in logs & reports. Values are: **None**, **BIOS Serial Number**, **Motherboard Serial Number**.

Application Log File Settings

These settings apply to the log file generated by the application. All operations performed in a **Active@ Disk Editor** session will be saved in this log.

Log file location

Allows the user to specify where the application log file is saved. By default this is set to a **Active@ Disk Editor** installation directory.

Application log detail level

Manipulate the amount of details included in the logs. Options are: **Minimum** and **Maximum**.

Initialize application log when application starts

This setting configures whether **Active@ Disk Editor** generates a new log file for every session (erasing the log of the previous session) or appends new sessions to one log file. Moreover, logs can be placed to the files being named using naming pattern specified.

Environment

These are configurable options pertaining to the applications user interface and user experience.

Application style

Configures the color scheme used in the application. Values are: **Blue**, **Olive**, **None (Use OS default)**, **Silver** and **Dark**.

Default toolbar style

Configures how icons are shown in the toolbar. Values are: **Large icons, no text**; **Large icons, with text beside icon**; **Large icons, with text under icon**; **Small icons, with text beside icon; Small icons, no text.**

Default help source

If available, user can select help documentation source to be addressed when requested. Values are: **PDF**, **Context Help** and **On-line web help.**

Show notification dialog after process complete

Show or hides final process confirmation dialog.

Reset all dialogs

Resets all the settings to the default state.

Sound Notifications

These are configurable options related to application sounds: you can use either predefined values or assign your own sounds (User defined sound file).

Use Sound Notifications

Toggles sound tones being used for notifying the user of the completion of a task, errors and notification during an operation: **Success**, **With Warnings**, **With Errors**, **Failure**.

Action Triggers

Configure actions performed while application is running.

Automatically check for software updates

If this option set, application will check for a new update after every start up.

Action after all processes complete

Select either **None**, **Hibernate**, **Shutdown** or **Restart** system after all running processes completed.

CAUTION:

You will have 30 seconds to abort system hibernation, restart or shutdown.

Export erase certificates and application log to all detected removable media

Upon erase completion all certificates and logs will be automatically exported to attached USB disks (all detected media of removable type).

Disk Editor preferences

Attributes

Auto load objects

Load (open) edited objects in Disk Editor at each application start if they present in system.

Open as Read Only

Open objects in Read Only mode by default.

Show Data Inspector pane

Show\Hide Data Inspector pane by default

Show Bookmark pane

Show\Hide Bookmark pane by default

Show Cluster Chain pane

Show\Hide Cluster Chain pane for edited files by default.

File view mode

Toggles default file view mode - files can be viewed as data

Auto apply template

If this option is ON, then most suitable data structure template for opened object will be applied and set visible.

Use template coloring

Toggle between template coloring or transparent template fields presentation.

Editor View attributes

Hexadecimal offset

Toggle between decimal and hexadecimal offset format

Show ASCII

Show\Hide ASCII decoding column

Show UNICODE

Show\Hide UNICODE decoding column

Bytes per line

Defines bytes per line representation. Minimum 8 bytes and maximum 255 bytes per line.

Lines per wheel scroll

Number of lines on each single mouse wheel scroll action.

Pages per scroll

Pages to scroll on each **PageUp** or **PageDown** keyboard button action.

Font name

List of mono-space font faces available in system to use in Disk Editor view.

Font size

Toggle between relative font size.

Error Handling

Error Handling section has the advanced settings to configure error handling while erasing or cloning the data.

Error handling attributes

Active@ Disk Editor allows to select one of ways to handle Read/Write Errors:

Abort entire group processing
If erase Batch is in progress and one of the disks has errors, the erase process for ALL the disks in the Batch will be terminated.

Abort only failed disk from group processing
This is the default setting. Failed disks return an error and terminate the erase process. Other disks in the Batch will continue current operation.

Ignore error for group processing
Ignores the read/write error and continues erasing whatever is possible on the disk. None active or forth going operations will be terminated.

Terminate process after number of errors
Sets the error threshold to a certain amount before the disk operation is terminated and deemed unsuccessful.

Number of read/write attempts
Sets the number of attempts **Active@ Disk Editor** makes to perform an operation when an error is encountered before it stops execution.

Ignore preceding results
Errors (if any) on previous steps (i.e. Examination) are ignored and following steps (i.e. Erase, Clone) will be executed. If turned off the errors on previous steps will stop all further actions.

Use disk lock
Locks disks from being used by any other applications while operation is in progress.

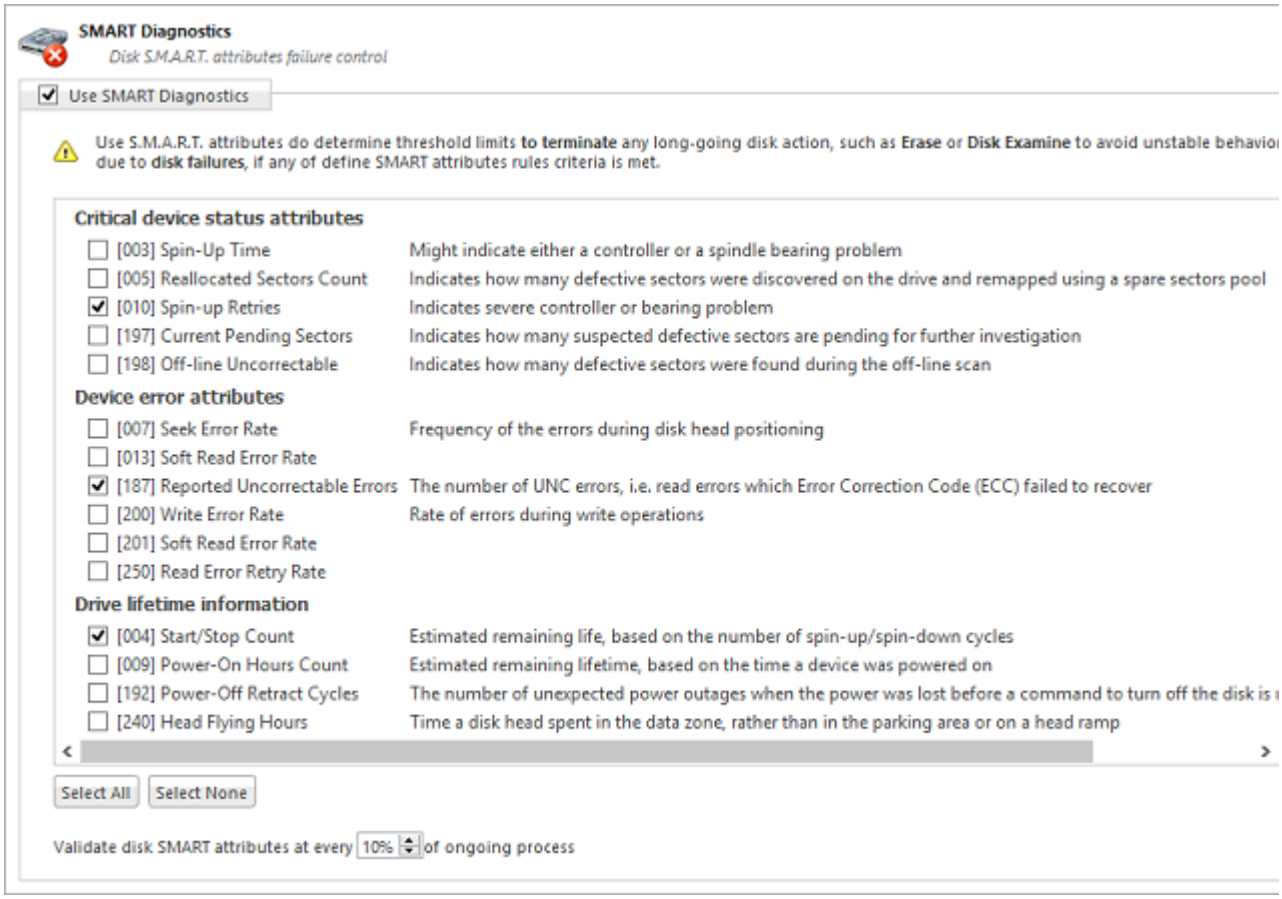
Ignore disk lock errors
Errors encountered with **Active@ Disk Editor** not being able to access locked disks will be ignored.

Ignore read/write errors
Toggle whether read errors or write errors will be just ignored.

Rely upon disk performance
Sets a minimum acceptable read/write speed in megabytes per second for disks to flag under-performing drives.

SMART Diagnostics

S.M.A.R.T attributes can be used in error handling. Threshold limits can be set for some disks or for all the disks based on S.M.A.R.T parameters. This can speed up processing by terminating operations with unusable drives immediately.



Note:

Query execution for S.M.A.R.T attributes is time consuming and resource consuming operation. Single query can interrupt disk erasure procedure for several seconds. Thus it is recommended to validate these attributes less frequently.

Related information

S.M.A.R.T Monitor

Searching patterns

Wildcards

A *wildcard* is a character that can be used as a substitute for any of a class of characters in a search. Wildcard characters are often used in place of one or more characters when you do not know what the real character is or you do not want to enter the entire name. In **Active@ Disk Editor** three types of wildcard are used: **star** or asterisk(*), **question mark** (?) and **number sign** (#).

Examples of using wildcards:

Wildcard character	Example	Description
Asterisk (*)	docum*	Use the asterisk as a substitute for zero or more characters if you are looking for a file that you know what it starts with and you cannot remember the rest of the file name. The example locates all files of any file type that begin with **"docum"** including *documents.txt*, *document_01.doc* and *documentum.doc*.
	docum*.doc	To narrow the search to a specific type of file, include the file extension. The example locates all files that begin with **"docum"** and have the file name extension **.doc**, such as *document_01.doc* and *documentum.doc*.
Question mark (?)	doc?.doc	Use the question mark as a substitute for a single character in a file name. In the example, you will locate the file *docs.doc* or *doc1.doc* but not *documents.doc*.
Number sign (#)	doc_###.doc	Use the number sign (also known as the pound or hash sign) as a substitute for a single number in a name. In the example, you will locate the file *doc_012.doc* or *doc_211.doc* but not *doc_ABS.doc*.

Regular expressions

Regular expressions are special search patterns, more capable then wildcards to define search criteria.

Examples of using regular expressions:

^\d\d?$ - match integers 0 to 99

^\S+$ - match strings without white space

\b(mail|letter|correspondence)\b - match strings containing 'mail' or 'letter' or 'correspondence' but only match whole words i. e. not 'email'

&(?!amp;) - match ampersands but not &

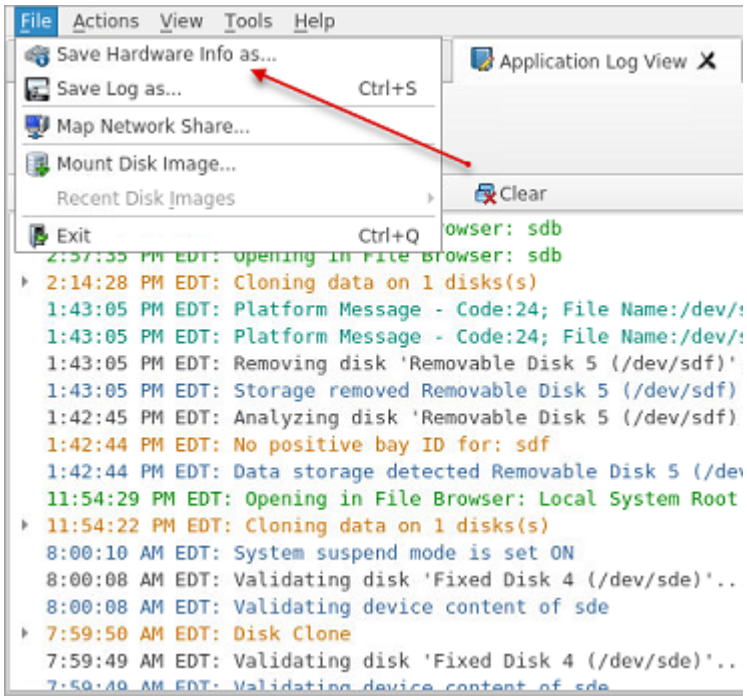
\b(Eric|Eirik)\b - match Eric or Eirik

Hardware diagnostic file

If you want to contact our technical support a file that contains a summary of your local devices and hardware configuration is very helpful and it is required to submit it for the proper problem investigation.

Application allows you to create a hardware summary file in XML format. This data format is "human-readable" and can help our technical support staff to analyze your computer configuration or point out disk failures or abnormal behavior.

To create a hardware diagnostic file, click **Save Hardware Info as** from the **File** menu .



Note:

To save time on initial contact with our technical support staff we highly recommend that you submit a hardware diagnostic file, otherwise, most likely, it will be requested from you by our support team later on.

Related information

Application Log

Knowledge base

Knowledge Base overview

To understand underlying mechanisms of data storage and logical organization, data recovery and analysis, the following topics will give essential concepts:

Understanding Hardware and Disk Organization
Basic information about Hard Disk Drives (HDD) and low-level disk organization.

Understanding File System (FAT)
The 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.

Understanding File System (NTFS)

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.

Hardware and Disk Architecture

Hardware and Disk Organization

Understanding of underlying mechanisms of data storage, organization and data recovery.

Here you can get some information about Hard Disk Drives (HDD) and low-level disk organization:

- Hard Disk Drive Basics on page 49
- Master Boot Record (MBR) on page 51
- Partition Table on page 52

Hard Disk Drive Basics

Understanding of underlying mechanisms of data storage, organization and data recovery.

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.

Each disk consists of platters, rings on each side of each platter called tracks, and sections within each track called sectors. A sector is the smallest physical storage unit on a disk, almost always 512 bytes in size.

Figure below illustrates a hard disk with two platters. The remainder of this section describes the terms used on the figure.

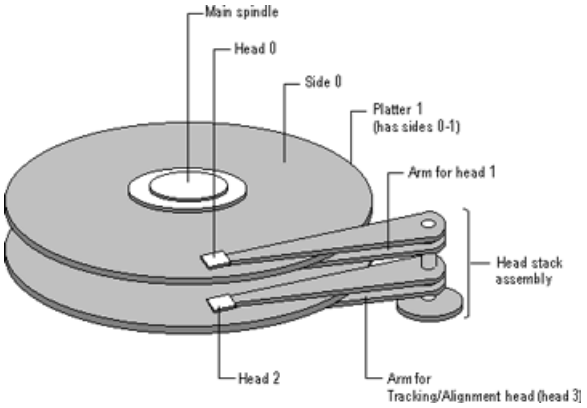


Figure 28: Two plated hard disk

The cylinder/head/sector notation scheme described in this section is slowly being eliminated. All new disks use some kind of translation factor to make their actual hardware layout appear as something else, mostly to work with MS-DOS and Windows 95.

Tracks and Cylinders

On hard disks, the data are stored on the disk in thin, concentric bands called *tracks*. There can be more than a thousand tracks on a 3½ inch hard disk. Tracks are a logical rather than physical structure, and are established when the disk is low-level formatted. Track numbers start at 0, and track 0 is the outermost track of the disk. The highest numbered track is next to the spindle. If the disk geometry is being translated,

the highest numbered track would typically be 1023. Next figure shows track 0, a track in the middle of the disk, and track 1023.

A *cylinder* consists of the set of tracks that are at the same head position on the disk. In a figure below, cylinder 0 is the four tracks at the outermost edge of the sides of the platters. If the disk has 1024 cylinders (which would be numbered 0-1023), cylinder 1023 consists of all of the tracks at the innermost edge of each side.

Most disks used in personal computers today rotate at a constant angular velocity. The tracks near the outside of the disk are less densely populated with data than the tracks near the center of the disk. Thus, a fixed amount of data can be read in a constant period of time, even though the speed of the disk surface is faster on the tracks located further away from the center of the disk.

Modern disks reserve one side of one platter for track positioning information, which is written to the disk at the factory during disk assembly. It is not available to the operating system. The disk controller uses this information to fine tune the head locations when the heads move to another location on the disk. When a side contains the track position information, that side cannot be used for data. Thus, a disk assembly containing two platters has three sides that are available for data.

Sectors and Clusters

Each track is divided into sections called *sectors*. A sector is the smallest physical storage unit on the disk. The data size of a sector is always a power of two, and is almost always 512 bytes.

Each track has the same number of sectors, which means that the sectors are packed much closer together on tracks near the center of the disk. Next figure shows sectors on a track. You can see that sectors closer to the spindle are closer together than those on the outside edge of the disk. The disk controller uses the sector identification information stored in the area immediately before the data in the sector to determine where the sector itself begins.

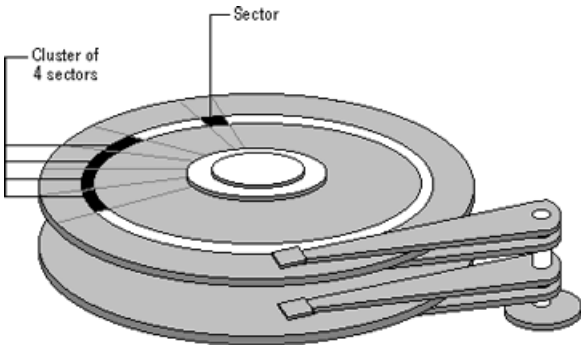


Figure 29: Clusters and sectors

As a file is written to the disk, the file system allocates the appropriate number of *clusters* to store the file's data. For example, if each cluster is 512 bytes and the file is 800 bytes, two clusters are allocated for the file. Later, if you update the file to, for example, twice its size (1600 bytes), another two clusters are allocated.

If contiguous clusters (clusters that are next to each other on the disk) are not available, the data are written elsewhere on the disk, and the file is considered to be *fragmented*. Fragmentation is a problem when the file system must search several different locations to find all the pieces of the file you want to read. The search causes a delay before the file is retrieved. 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 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.

Related concepts

Hardware and Disk Organization on page 49
Understanding of underlying mechanisms of data storage, organization and data recovery.

Master Boot Record (MBR) on page 51
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition Table on page 52
Understanding of underlying mechanisms of data storage, organization and data recovery.

Master Boot Record (MBR)

Understanding of underlying mechanisms of data storage, organization and data recovery.

The *Master Boot Record*, created when you create the first partition on the hard disk, is probably the most important data structure on the disk. It is the first sector on every disk. The location is always track (cylinder) 0, side (head) 0, and sector 1.

The Master Boot Record contains the Partition Table on page 52 for the disk and a small amount of executable code. On x86-based computers, the executable code examines the Partition Table, and identifies the system partition. The Master Boot Record then finds the system partition's starting location on the disk, and loads an copy of its Partition Boot Sector into memory. The Master Boot Record then transfers execution to executable code in the Partition Boot Sector.

Note:

Although there is a Master Boot Record on every hard disk, the executable code in the sector is used only if the disk is connected to an x86-based computer and the disk contains the system partition.

Figure below shows a hex dump of the sector containing the Master Boot Record. The figure shows the sector in two parts. The first part is the Master Boot Record, which occupies the first 446 bytes of the sector. The disk signature (FD 4E F2 14) is at the end of the Master Boot Record code. The second part is the Partition Table on page 52.

```
Physical Sector: Cyl 0, Side 0, Sector 1

00000000: 00 33 C0 8E D0 BC 00 7C  -  8B F4 50 07 50 1F FB FC   .3.....|..P.P..
00000010: BF 00 06 B9 00 01 F2 A5  -  EA 1D 06 00 00 BE BE 07   ................
00000020: B3 04 80 3C 80 74 0E 80  -  3C 00 75 1C 83 C6 10 FE   ...<.t..<.u.....
00000030: CB 75 EF CD 18 8B 14 8B  -  4C 02 8B EE 83 C6 10 FE   .u......L.......
00000040: CB 74 1A 80 3C 00 74 F4  -  BE 8B 06 AC 3C 00 74 0B   .t..<.t.....<.t.
00000050: 56 BB 07 00 B4 0E CD 10  -  5E EB F0 EB FE BF 05 00   V.......^.......
00000060: BB 00 7C B8 01 02 57 CD  -  13 5F 73 0C 33 C0 CD 13   ..|...W.._s.3...
00000070: 4F 75 ED BE A3 06 EB D3  -  BE C2 06 BF FE 7D 81 3D   Ou...........}.=
00000080: 55 AA 75 C7 8B F5 EA 00  -  7C 00 00 49 6E 76 61 6C   U.u.....|..Inval
00000090: 69 64 20 70 61 72 74 69  -  74 69 6F 6E 20 74 61 62   id partition tab
000000A0: 6C 65 00 45 72 72 6F 72  -  20 6C 6F 61 64 69 6E 67   le.Error loading
000000B0: 20 6F 70 65 72 61 74 69  -  6E 67 20 73 79 73 74 65    operating syste
000000C0: 6D 00 4D 69 73 73 69 6E  -  67 20 6F 70 65 72 61 74   m.Missing operat
000000D0: 69 6E 67 20 73 79 73 74  -  65 6D 00 00 80 45 14 15   ing system...E..
000000E0: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
000000F0: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
00000100: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
00000110: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
00000120: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
00000130: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
00000140: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
00000150: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
00000160: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
00000170: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
00000180: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
```

```
000001B0: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
000001A0: 00 00 00 00 00 00 00 00  -  00 00 00 00 00 00 00 00   ................
000001B0: 00 00 00 00 00 00 00 00  -  FD 4E F2 14 00 00 80 01   .........N......
000001C0: 01 00 06 0F 7F 96 3F 00  -  00 00 51 42 06 00 00 00   ....#.?...QB....
000001D0: 41 97 07 0F FF 2C 90 42  -  06 00 A0 3E 06 00 00 00   A....,.B...>....
000001E0: C1 2D 05 0F FF 92 30 81  -  0C 00 A0 91 01 00 00 00   .-....0.........
000001F0: C1 93 01 0F FF A6 D0 12  -  0E 00 C0 4E 00 00 55 AA   ...........N..U.
```

Important: Viruses Can Infect the Master Boot Record

Many destructive viruses damage the Master Boot Record and make it impossible to start the computer from the hard disk. Because the code in the Master Boot Record executes before any operating system is started, no operating system can detect or recover from corruption of the Master Boot Record. You can use, for example, the DiskProbe program on *Windows NT Workstation Resource Kit* CD to display the Master Boot Record, and compare it to the Master Boot Record shown above. There are also utilities on the Microsoft Windows Resource Kits that enable you to save and restore the Master Boot Record.

Tip:

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

Related concepts

Hardware and Disk Organization on page 49
Understanding of underlying mechanisms of data storage, organization and data recovery.

Hard Disk Drive Basics on page 49
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition Table on page 52
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition Table

Understanding of underlying mechanisms of data storage, organization and data recovery.

The information about primary partitions and an extended partition is contained in the Partition Table, a 64-byte data structure located in the same sector as the Master Boot Record (MBR) on page 51 (cylinder 0, head 0, sector 1). The Partition Table conforms to a standard layout that is independent of the operating system. Each Partition Table entry is 16 bytes long, making a maximum of four entries available. Each entry starts at a predetermined offset from the beginning of the sector, as follows:

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

The last two bytes in the sector are a signature word for the sector and are always 0x55AA.

The next figure is a printout of the Partition Table for the disk shown in a Master Boot Record (MBR) on page 51 earlier in this chapter. When there are fewer than four partitions, the remaining fields are all zeros.

```
000001B0:                                       80 01              ..
000001C0: 01 00 06 0F 7F 96 3F 00 - 00 00 51 42 06 00 00 00   ....#.?...QB.....
000001D0: 41 97 07 0F FF 2C 90 42 - 06 00 A0 3E 06 00 00 00   A....,.B...>....
000001E0: C1 2D 05 0F FF 92 30 81 - 0C 00 A0 91 01 00 00 00   .-....0.........
000001F0: C1 93 01 0F FF A6 D0 12 - 0E 00 C0 4E 00 00 55 AA   ...........N..U.
```

The following table describes each entry in the Partition Table. The sample values correspond to the information for partition 1.

Table 1: Partition Table Fields

Byte Offset	Field Length	Sample Value	Meaning
00	BYTE	0x80	Boot Indicator. Indicates whether the partition is the system partition. Legal values are: 00 = Do not use for booting. 80 = System partition.
01	BYTE	0x01	Starting Head.
02	6 bits	0x01	Starting Sector. Only bits 0-5 are used. Bits 6-7 are the upper two bits for the Starting Cylinder field.
03	10 bits	0x00	Starting Cylinder. This field contains the lower 8 bits of the cylinder value. Starting cylinder is thus a 10-bit number, with a maximum value of 1023.
04	BYTE	0x06	System ID. This byte defines the volume type. In Windows NT, it also indicates that a partition is part of a volume that requires the use of the HKEY_LOCAL_MACHINE \SYSTEM\DISK Registry subkey.
05	BYTE	0x0F	Ending Head.
06	6 bits	0x3F	Ending Sector. Only bits 0-5 are used. Bits 6-7 are the upper two bits for the Ending Cylinder field.
07	10 bits	0x196	Ending Cylinder. This field contains the lower 8 bits of the cylinder value. Ending cylinder is thus a 10-bit number, with a maximum value of 1023.
08	DWORD	3F 00 00 00	Relative Sector.
12	DWORD	51 42 06 00	Total Sectors.

The remainder of this section describes the uses of these fields. Definitions of the fields in the Partition Table is the same for primary partitions, extended partitions, and logical drives in extended partitions.

Boot Indicator Field

The Boot Indicator field indicates whether the volume is the system partition. On x-86-based computers, only one primary partition on the disk should have this field set. This field is used only on x86-based computers. On RISC-based computers, the NVRAM contains the information for finding the files to load.

On x86-based computers, it is possible to have different operating systems and different file systems on different volumes. For example, a computer could have MS-DOS on the first primary partition and Windows 95, UNIX, OS/2, or Windows NT on the second. You control which primary partition (active partition in FDISK) to use to start the computer by setting the Boot Indicator field for that partition in the Partition Table.

System ID field

For primary partitions and logical drives, the System ID field describes the file system used to format the volume. Windows NT uses this field to determine what file system device drivers to load during startup. It also identifies the extended partition, if there is one defined.

Table 2: System ID field description

Value	Meaning
0x01	12-bit FAT primary partition or logical drive. The number of sectors in the volume is fewer than 32680.

Value	Meaning
0x04	16-bit FAT primary partition or logical drive. The number of sectors is between 32680 and 65535.
0x05	Extended partition. See section titled "Logical Drives and Extended Partitions," presented later in this chapter, for more information.
0x06	BIGDOS FAT primary partition or logical drive.
0x07	NTFS primary partition or logical drive.

Figure presented earlier in this section, has examples of a BIGDOS FAT partition, an NTFS partition, an extended partition, and a 12-bit FAT partition.

If you install Windows NT on a computer that has Windows 95 preinstalled, the FAT partitions might be shown as unknown. If you want to be able to use these partitions when running Windows NT, your only option is to delete the partitions.

OEM versions of Windows 95 support the following four partition types for FAT file systems that Windows NT cannot recognize.

Value	Meaning
0x0B	Primary Fat32 partition, using interrupt 13 (INT 13) extensions.
0x0C	Extended Fat32 partition, using INT 13 extensions.
0x0E	Extended Fat16 partition, using INT 13 extensions.
0x0F	Primary Fat16 partition, using INT 13 extensions.

When you create a volume set or a stripe set, Disk Administrator sets the high bit of the System ID field for each primary partition or logical drive that is a member of the volume. For example, a FAT primary partition or logical drive that is a member of a volume set or a stripe set has a System ID value of 0x86. An NTFS primary partition or logical drive has a System ID value of 0x87. This bit indicates that Windows NT needs to use the HKEY_LOCAL_MACHINE\SYSTEM\DISK Registry subkey to determine how the members of the volume set or stripe set relate to each other. Volumes that have the high bit set can only be accessed by Windows NT.

When a primary partition or logical drive that is a member of a volume set or a stripe set has failed due to write errors or cannot be accessed, the second most significant bit is set. The System ID byte is set to C6 in the case of a FAT volume, or C7 in the case of an NTFS volume.

Note:

If you start up MS-DOS, it can only access primary partitions or logical drives that have a value of 0x01, 0x04, 0x05, or 0x06 for the System ID. However, you should be able to delete volumes that have the other values. If you use a MS-DOS-based low-level disk editor, you can read and write any sector, including ones that are in NTFS volumes.

On Windows NT Server, mirror sets and stripe sets with parity also require the use of the Registry subkey HKEY_LOCAL_MACHINE\SYSTEM\DISK to determine how to access the disks.

Starting and Ending Head, Sector, and Cylinder Fields

On x86-based computers, the Starting and Ending Head, Cylinder, and Sector fields on the start-up disk are very important for starting up the computer. The code in the Master Boot Record uses these fields to find and load the Partition Boot Sector.

The Ending Cylinder field in the Partition Table is ten bits long, which limits the maximum number of cylinders that can be described in the Partition Table to 1024. The Starting and Ending Head fields are one byte long, which limits this field to the range 0 – 255. The Starting and Ending Sector field is 6 bits

long, limiting its range to 0 – 63. However, sectors start counting at 1 (versus 0 for the other fields), so the maximum number of sectors per track is 63.

Since current hard disks are low-level formatted with the industry standard 512-byte sector size, the maximum capacity disk that can be described by the Partition Table can be calculated as follows:

```
MaxCapacity = (sector size) x (sectors per track) x (cylinders) x (heads)
```

Substituting the maximum possible values yields:

```
512 x 63 x 1024 x 256 = 8,455,716,864 bytes or 7.8 GB
```

The maximum formatted capacity is slightly less than 8 GB.

However, the maximum cluster size that you can use for FAT volumes when running Windows NT is 64K, when using a 512 byte sector size. Therefore, the maximum size for a FAT volume is 4 GB.

If you have a dual-boot configuration with Windows 95 or MS-DOS, FAT volumes that might be accessed when using either of those operating systems are limited to 2 GB. In addition, Macintosh computers that are viewing volumes on a computer running Windows NT cannot see more than 2 GB. If you try to use a FAT volume larger than 2 GB when running MS-DOS or Windows 95, or access it from a Macintosh computer, you might get a message that there are 0 bytes available. The same limit applies to OS/2 system and boot partitions.

The maximum size of a FAT volume on a specific computer depends on the disk geometry, and the maximum values that can fit in the fields described in this section. The next table shows the typical size of a FAT volume when translation is enabled, and when it is disabled. The number of cylinders in both situations is 1024.

Translation mode	Numb of heads	Sector per track	Maximum size for system or boot partition
Disabled	64	32	1 GB
Enabled	255	63	4 GB

Note:

RISC-based computers do not have a limit on the size of the system or boot partitions.

If a primary partition or logical drive extends beyond cylinder 1023, all of these fields will contain the maximum values.

Relative Sectors and Number of Sectors Fields

For primary partitions, the Relative Sectors field represents the offset from the beginning of the disk to the beginning of the partition, counting by sectors. The Number of Sectors field represents the total number of sectors in the partition. For a description of these fields in extended partitions, see the section Logical Drives and Extended Partitions.

Windows NT uses these fields to access all partitions. When you format a partition when running Windows NT, it puts data into the Starting and Ending Cylinder, Head, and Sector fields only for backward compatibility with MS-DOS and Windows 95, and to maintain compatibility with the BIOS interrupt (INT) 13 for start-up purposes.

Logical Drives and Extended Partitions

When more than four logical disks are required on a single physical disk, the first partition should be a primary partition. The second partition can be created as an extended partition, which can contain all the remaining unpartitioned space on the disk.

Note:

A primary partition is one that can be used as the system partition. If the disk does not contain a system partition, you can configure the entire disk as a single, extended partition.

Some computers create an EISA configuration partition as the first partition on the hard disk.

Windows NT detects an extended partition because the System ID byte in the Partition Table entry is set to 5. There can be only one extended partition on a hard disk.

Within the extended partition, you can create any number of logical drives. As a practical matter, the number of available drive letters is the limiting factor in the number of logical drives that you can define.

When you have an extended partition on the hard disk, the entry for that partition in the Partition Table (at the end of the Master Boot Record) points to the first disk sector in the extended partition. The first sector of each logical drive in an extended partition also has a Partition Table, which is the last 66 bytes of the sector. (The last two bytes of the sector are the end-of-sector marker.)

These are the entries in an extended Partition Table:

- The first entry is for the current logical drive.
- The second entry contains information about the next logical drive in the extended partition.
- Entries three and four are all zeroes.

This format repeats for every logical drive. The last logical drive has only its own partition entry listed. The entries for partitions 2-4 are all zeroes.

The Partition Table entry is the only information on the first side of the first cylinder of each logical drive in the extended partition. The entry for partition 1 in each Partition Table contains the starting address for data on the current logical drive. And the entry for partition 2 is the address of the sector that contains the Partition Table for the next logical drive.

The use of the Relative Sector and Total Sectors fields for logical drives in an extended partition is different than for primary partitions. For the partition 1 entry of each logical drive, the Relative Sectors field is the sector from the beginning of the logical drive that contains the Partition Boot Sector. The Total Sectors field is the number of sectors from the Partition Boot Sector to the end of the logical drive.

For the partition 2 entry, the Relative Sectors field is the offset from the beginning of the extended partition to the sector containing the Partition Table for the logical drive defined in the Partition 2 entry. The Total Sectors field is the total size of the logical drive defined in the Partition 2 entry.

Note:

If a logical drive is part of a volume set, the Partition Boot Sector is at the beginning of the first member of the volume set. Other members of the volume set have data where the Partition Boot Sector would normally be located.

Tip:

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

Related concepts

Hardware and Disk Organization on page 49
Understanding of underlying mechanisms of data storage, organization and data recovery.

Hard Disk Drive Basics on page 49
Understanding of underlying mechanisms of data storage, organization and data recovery.

Master Boot Record (MBR) on page 51
Understanding of underlying mechanisms of data storage, organization and data recovery.

Disk arrays (RAID)
Redundant array of independent disks (RAID)

Redundant array of independent disks (RAID) is a storage technology that combines multiple disk drive components into a logical unit. Data is distributed across the drives in one of several ways called "RAID levels", depending on what level of redundancy and performance (via parallel communication) is required.

RAID types
RAID-0
This technique has striping but no redundancy of data. It offers the best performance but no fault-tolerance.

RAID-1
This type is also known as disk mirroring and consists of at least two drives that duplicate the storage of data. There is no striping. Read performance is improved since either disk can be read at the same time. Write performance is the same as for single disk storage. RAID-1 provides the best performance and the best fault-tolerance in a multi-user system.

RAID-2
This type uses striping across disks with some disks storing error checking and correcting (ECC) information. It has no advantage over RAID-3.

RAID-3
This type uses striping and dedicates one drive to storing parity information. The embedded error checking (ECC) information is used to detect errors. Data recovery is accomplished by calculating the exclusive OR (XOR) of the information recorded on the other drives. Since an I/O operation addresses all drives at the same time, RAID-3 cannot overlap I/O. For this reason, RAID-3 is best for single-user systems with long record applications.

RAID-4
This type uses large stripes, which means you can read records from any single drive. This allows you to take advantage of overlapped I/O for read operations. Since all write operations have to update the parity drive, no I/O overlapping is possible. RAID-4 offers no advantage over RAID-5.

RAID-5
This type includes a rotating parity array, thus addressing the write limitation in RAID-4. Thus, all read and write operations can be overlapped. RAID-5 stores parity information but not redundant data (but parity information can be used to reconstruct data). RAID-5 requires at least three and usually five disks for the array. It's best for multi-user systems in which performance is not critical or which do few write operations.

Parity tables

Left Synchronous			
0	5	6	P
1	4	P	11
2	P	7	10
P	3	8	9

Left Asynchronous			
0	3	6	P
1	4	P	9

Left Asynchronous			
2	P	7	10
P	5	8	11

Right Synchronous			
P	5	6	11
0	P	7	10
1	4	P	9
2	3	8	P

Right Asynchronous			
P	3	6	9
0	P	7	10
1	4	P	11
2	5	8	P

Logical Disk Manager

Understanding of underlying mechanisms of data storage, organization and data recovery.

Dynamic disks provide features that basic disks do not, such as the ability to create volumes that span multiple disks (spanned and striped volumes), and the ability to create fault tolerant volumes (mirrored and RAID-5 volumes). All volumes on dynamic disks are known as dynamic volumes.

There are five types of dynamic volumes:

Simple
A dynamic volume made up of disk space from a single dynamic disk. A simple volume can consist of a single region on a disk or multiple regions of the same disk that are linked together. If the simple volume is not a system volume or boot volume, you can extend it within the same disk or onto additional disks. If you extend a simple volume across multiple disks, it becomes a spanned volume. You can create simple volumes only on dynamic disks. Simple volumes are not fault tolerant, but you can mirror them to create mirrored volumes on computers running the Windows 2000 Server or Windows Server 2003 families of operating systems.

Spanned
A dynamic volume consisting of disk space on more than one physical disk. You can increase the size of a spanned volume by extending it onto additional dynamic disks. You can create spanned volumes only on dynamic disks. Spanned volumes are not fault tolerant and cannot be mirrored.

Striped
A dynamic volume that stores data in stripes on two or more physical disks. Data in a striped volume is allocated alternately and evenly (in stripes) across the disks. Striped volumes offer the best performance of all the volumes that are available in Windows, but they do not provide fault tolerance. If a disk in a striped volume fails, the data in the entire volume is lost. You can create striped volumes only on dynamic disks. Striped volumes cannot be mirrored or extended.

Mirrored
A fault-tolerant volume that duplicates data on two physical disks. A mirrored volume provides data redundancy by using two identical volumes, which are called mirrors, to duplicate the information contained on the volume. A mirror is always located on a different disk. If one of the physical disks fails, the data on the failed disk becomes unavailable, but the system continues to operate in the mirror on the remaining disk. You can create mirrored volumes only on dynamic disks on computers running the Windows 2000 Server or Windows Server 2003 families of operating systems. You cannot extend mirrored volumes.

RAID-5

A fault-tolerant volume with data and parity striped intermittently across three or more physical disks. Parity is a calculated value that is used to reconstruct data after a failure. If a portion of a physical disk fails, Windows recreates the data that was on the failed portion from the remaining data and parity. You can create RAID-5 volumes only on dynamic disks on computers running the Windows 2000 Server or Windows Server 2003 families of operating systems. You cannot mirror or extend RAID-5 volumes. In Windows NT 4.0, a RAID-5 volume was known as a striped set with parity.

Mirrored and RAID-5 volumes are fault tolerant and are available only on computers running Windows 2000 Server, Windows 2000 Advanced Server, Windows 2000 Datacenter Server, or the Windows Server 2003 family of operating systems. You can, however, use a computer running Windows XP Professional to remotely create mirrored and RAID-5 volumes on these operating systems.

Regardless of whether the dynamic disk uses the master boot record (MBR) or GUID partition table (GPT) partition style, you can create up to 2,000 dynamic volumes, although the recommended number of dynamic volumes is 32 or less.

For information about how to manage dynamic volumes, see Manage dynamic volumes.

Virtual Disks

KillDisk provides full support for Virtual Disks - dynamic disks created and managed by:

- **Logical Disk Manager** (LDM on Windows)
- **Logical Volume Manager** (LVM on Linux)
- **Windows Storage Spaces** (WSS on Windows)

Virtual Disks are virtual devices which look like regular physical disks to all applications. These virtual devices are stored on one or more physical disks and emulate different types of volumes and RAID disk arrays not on a hardware level (inside disk controller), but on Operating System level (software emulation). Virtual devices are fully supported by the **KillDisk**. These disks will appear in **Local Devices** view like any other regular disks. When you launch an erase for the virtual disk, the progress is displayed in the same color on all components of the composite virtual drive.

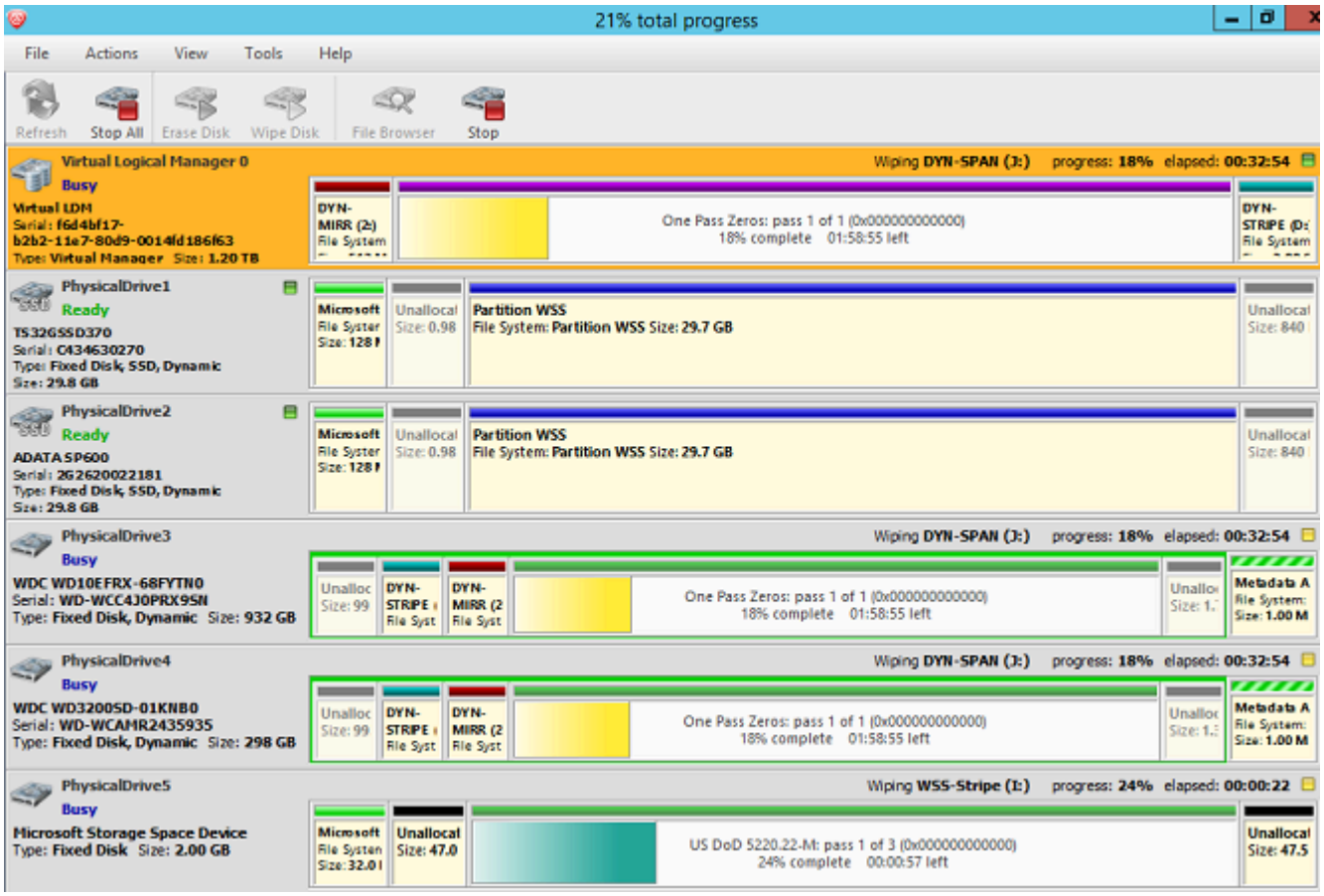


Figure 30: Erasing a Virtual Drive (Striped Disk Array)

Note: By default Virtual Disks are not being displayed in the list of devices. To display Virtual Disks go to **Preferences** > **General Settings** and turn on **Initialize virtual disks** option.

File Systems

Windows NT File System (NTFS)

Understanding of underlying mechanisms of data storage, organization and data recovery.

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).

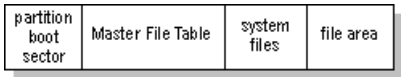


Figure 31: Layout of NTFS volume after formatting

See the next sections for more information about NTFS:

- NTFS Partition Boot Sector on page 61
- NTFS Master File Table (MFT) on page 63
- NTFS File Types on page 64

- Data Integrity and Recoverability with NTFS on page 69

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 behaviour 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.

Tip:

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

Understanding of underlying mechanisms of data storage, organization and data recovery.

Next table 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.

Byte Offset	Field Length	Field Name
0x00	3 bytes	Jump Instruction
0x03	LONGLONG	OEM ID
0x0B	25 bytes	BPB
0x24	48 bytes	Extended BPB

Byte Offset	Field Length	Field Name
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 start up. 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 3: BIOS Parameter Block and Extended BIOS Parameter Block Fields

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

Byte Offset	Field Length	Sample Value	Field Name
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	0x4AF57E0300000000	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	0x14A57B74C97B7411	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.

Tip:

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 Master File Table (MFT)

Understanding of underlying mechanisms of data storage, organization and data recovery.

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 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.

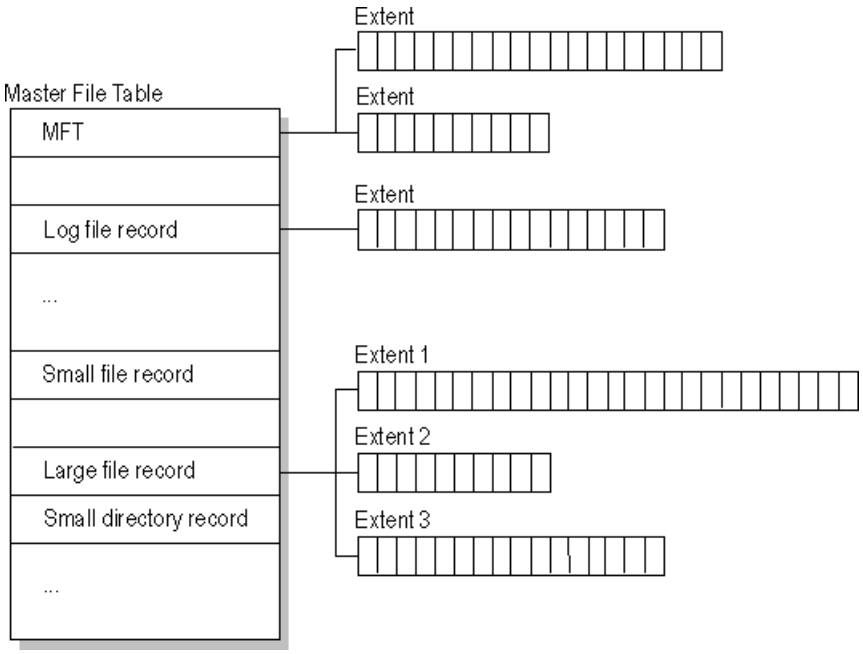


Figure 32: Simplified illustration of the MFT structure

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.

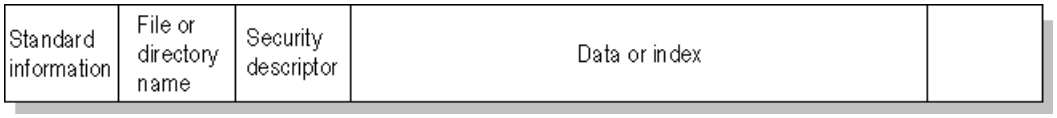


Figure 33: MFT Record for a Small File or Directory

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.

Tip:

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 File Types

Understanding of underlying mechanisms of data storage, organization and data recovery.

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 file name 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 non-resident. The non-resident 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.

Next table 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.

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.
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 meta data and to implement the file system. System files are placed on the volume by the Format utility.

Table 4: Meta-data Stored in the Master File Table

Syster File	File Name	MFT Recor	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.

System File	File Name	MFT Record	Purpose of the File
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	$LogFile	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.
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.

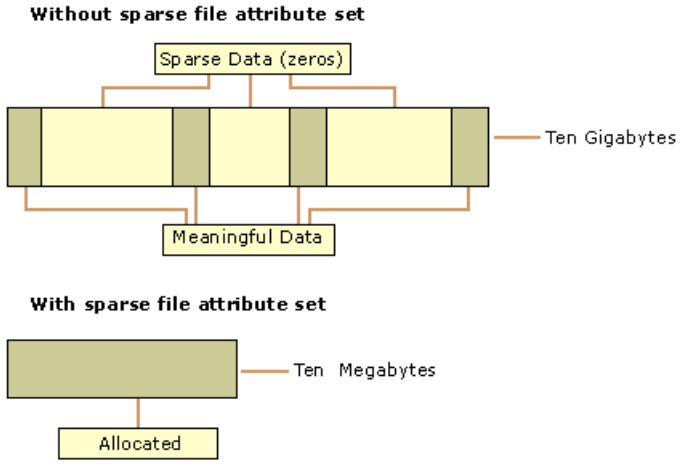
NTFS Sparse Files (Windows 2000 only)

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.



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.

Tip:

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

Data Integrity and Recoverability with NTFS

Understanding of underlying mechanisms of data storage, organization and data recovery.

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.

Tip:

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 System (FAT)

Understanding of underlying mechanisms of data storage, organization and data recovery.

The 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.

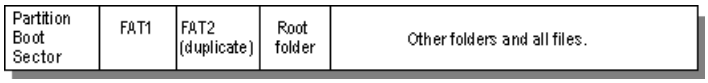


Figure 34: FAT file system volume organization

See the next sections for more information about FAT:

- FAT Partition Boot Sector on page 70
- FAT File Allocation Table on page 72
- FAT Root Folder on page 73
- FAT Folder Structure on page 73
- FAT32 Features on page 74

Main differences between FAT12, FAT16, FAT32

- FAT12 file system contains 1.5 bytes per cluster within the file allocation table.
- FAT16 file system contains 2 bytes per cluster within the file allocation table.
- FAT32 file system includes 4 bytes per cluster within the file allocation table.

Related concepts

Windows NT File System (NTFS) on page 60
Understanding of underlying mechanisms of data storage, organization and data recovery.

Extended File System (exFAT) on page 83
Understanding of underlying mechanisms of data storage, organization and data recovery.

FAT Partition Boot Sector

Understanding of underlying mechanisms of data storage, organization and data recovery.

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.

Next table describes the fields in the Partition Boot Sector for a volume formatted with the FAT file system.

Table 5: System ID field description

Byte Offset (in hex)	Field Length	Sample Value	Meaning
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: BIOS Parameter Block and Extended BIOS Parameter Block Fields

Byte Offset	Field Length	Sample Value	Meaning
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 file names 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 file names.
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.
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.

Byte Offset	Field Length	Sample Value	Meaning
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.

Tip:

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 File Allocation Table

Understanding of underlying mechanisms of data storage, organization and data recovery.

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 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.

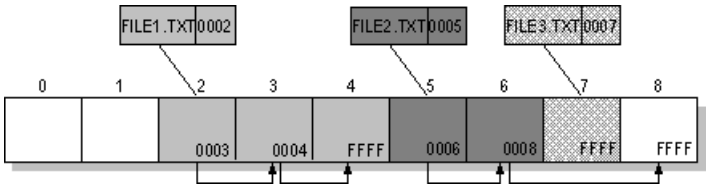


Figure 35: Example of File Allocation Table

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 entry (see folder entry for details) points to the first cluster of the file.

Tip:

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

Understanding of underlying mechanisms of data storage, organization and data recovery.

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 FAT Folder Structure on page 73 topic for details about folder organization.

Tip:

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

Understanding of underlying mechanisms of data storage, organization and data recovery.

Folders have set of 32-byte *Folder Entries* for each file and sub-folder 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 Table 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 sub folder, 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.

File names on FAT Volumes

Beginning with Windows NT 3.5, files created or renamed on FAT volumes use the attribute bits to support long file names 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 file name, 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 file name. Each of these secondary folder entries stores a corresponding part of the long file name 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 file name. 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 file name contained in the folder entry for the file.

Figure 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.

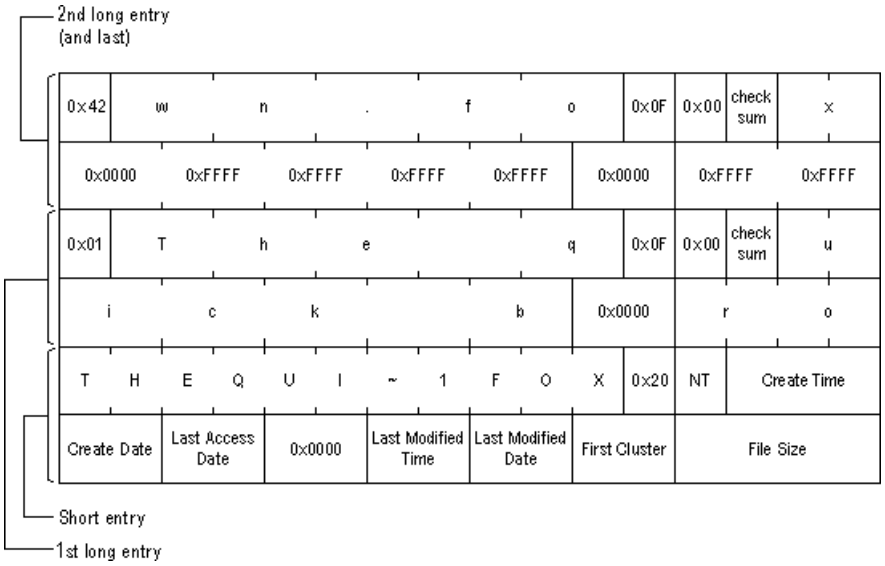


Figure 36: Example of Folder Entries for the long file name

Tip:

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

FAT32 Features

Understanding of underlying mechanisms of data storage, organization and data recovery.

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

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
```

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.

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.

Value	Description
BGBPB_F_ActiveFATMask (000Fh)	Mask for low four bits.
BGBPB_F_NoFATMirror (0080h)	Mask indicating FAT 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
```

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, **DPB**.

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. 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. 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**.

DPB (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
```

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.

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_ActiveFATMask (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.

Partition Types

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

Table 7: Partition Types

Value	Description
PART_UNKNOWN (00h)	Unknown
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.

Value	Description
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)

```
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
```

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.

PartFileSystem

Specifies the file system for the partition.

Table 8: 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.

Value	Description
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.

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.

Extended File System (exFAT)

Understanding of underlying mechanisms of data storage, organization and data recovery.

Extended File System (exFAT) is a successor of FAT family of file systems (FAT12/16/32). It has similar design though renders many significant improvements:

- Larger volume and file size limits
- Native Unicode file names
- Bigger boot area allowing a larger boot code
- Better performance
- Time zone offset support
- OEM parameters support

exFAT vs. FAT32 Comparison

Feature	FAT32	exFAT
Maximum Volume Size	8 TB*	128 PB
Maximum File Size	4 GB	16 EB
Maximum Cluster Size	32 KB **	32 MB
Maximum Cluster Count	228	232
Maximum File Name Length	255	255
Date/Time resolution	2 s	10 ms
MBR Partition Type Identifier	0x0B, 0x0C	0x07

Notice: Windows cannot format FAT32 volumes bigger than 32GB, though itsupports larger volumes created by third party implementations; 16 TB is the maximum volume size if formatted with 64KB cluster

Notice: According to Microsoft KB184006 clusters cannot be 64KB or larger, though some third party implementations support up to 64KB.

Related concepts

Volume Layout on page 84
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Directory Structure on page 89
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Defined Directory Entries on page 91
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Cluster Heap on page 98
Understanding of underlying mechanisms of data storage, organization and data recovery.

Volume Layout

Understanding of underlying mechanisms of data storage, organization and data recovery.

Offset, sectors	Size, sectors	Block	Comments
Main Boot Region			
0	1	Boot Sector	
1	8	Extended Boot Sectors	
9	1	OEM Parameters	
10	1	Reserved	
11	1	Boot Checksum	
Backup Boot Region			
12	1	Boot Sector	
13	8	Extended Boot Sectors	
21	1	OEM Parameters	
22	1	Reserved	
23	1	Boot Checksum	
FAT Region			

Offset, sectors	Size, sectors	Block	Comments
24	FatOffset - 24	FAT Alignment	Boot Sectors contain FatOffset
FatOffset	FatLength	First FAT	Boot Sectors contain FatOffset and FatLength
FatOffset + FatLength	FatLength	Second FAT	For TexFAT only
Data Region			
FatOffset + FatLength * NumberOfFats	ClusterHeapOffset – (FatOffset + FatLength * NumberOfFats)	Cluster Heap Alignment	
ClusterHeapOffset	ClusterCount * 2^SectorsPerClusterShift	Cluster Heap	
ClusterHeapOffset + ClusterCount * 2^SectorsPerClusterShift	VolumeLength – (ClusterHeapOffset + ClusterCount * 2^SectorsPerClusterShift)	Excess Space	

Navigate to detailed volume specification using following links:

- Boot Sector on page 85
- Extended Boot Sector on page 87
- OEM Parameters on page 87
- Boot Checksum on page 87
- File Allocation Table (FAT) on page 88

Boot Sector

Offset	Size	Description	Comments
0 (0x00)	3	JumpBoot	0xEB7690
3 (0x03)	8	FileSystemName	"EXFAT "
11 (0x0B)	53	MustBeZero	
64 (0x40)	8	PartitionOffset	In sectors; if 0, shall be ignored
72 (0x48)	8	VolumeLength	Size of exFAT volume in sectors
80 (0x50)	4	FatOffset	In sectors
84 (0x54)	4	FatLength	In sectors. May exceed the required space in order to align the second FAT
88 (0x58)	4	ClusterHeapOffset	In sectors
92 (0x5C)	4	ClusterCount	2^32-11 is the maximum number of clusters could be described
96 (0x60)	4	RootDirectoryCluster	
100 (0x64)	4	VolumeSerialNumber	

Offset	Size	Description	Comments
104 (0x68)	2	FileSystemRevision	as MAJOR.minor, major revision is high byte, minor is low byte; currently 01.00
106 (0x6A)	2	VolumeFlags (see below)	
108 (0x6C)	1	BytesPerSectorShift	Power of 2. Minimum 9 (512 bytes per sector), maximum 12 (4096 bytes per sector)
109 (0x6D)	1	SectorsPerCluster Shift	Power of 2. Minimum 0 (1 sector per cluster), maximum 25 – BytesPerSectorShift, so max cluster size is 32 MB
110 (0x6E)	1	NumberOfFats	2 is for TexFAT only
111 (0x6F)	1	DriveSelect	Extended INT 13h drive number; typically 0x80
112 (0x70)	1	PercentInUse	0..100 – percentage of allocated clusters rounded down to the integer 0xFF – percentage is not available
113 (0x71)	7	Reserved	
120 (0x78)	390	BootCode	
510 (0x1FE)	2	BootSignature	0xAA55
512 (0x200)	2^BytesPerSectorShift - 512	ExcessSpace	Not used

Table 9: Volume Flags

Offset	Size	Field
0	1	ActiveFat 0 - First FAT and Allocation Bitmap are active, 1 - Second .
1	1	VolumeDirty (0-clean, 1-dirty)
2	1	MediaFailure (0 – no failures reported or they already marked as BAD clusters) 1- some read/ write operations failed)
3	1	ClearToZero (no meaning)
4	12	Reserved

Extended Boot Sector

Offset	Size	Description	Comments
0 (0x00)	2^BytesPerSectorShift - 4	ExtendedBootCode	
2^BytesPerSectorShift - 4	4	ExtendedBootSignature	0xAA550000

Whole sector is used for boot code except last 4 bytes used for signature in each sector. If Extended Boot Sector is not used, it should be filled with 0x00. Extended signature must be preserved.

OEM Parameters

Offset	Size	Description	Comments
0 (0x00)	48	Parameters[0]	
…	…	…	
432 (0x1B0)	48	Parameters[9]	
480 (0x01E0)	2^BytesPerSectorShift - 480	Reserved	

OEM parameters are ignored by Windows but can be used by OEM implementations. OEMs can define their own parameters with unique GUIDs. All unused Parameters fields must be described as unused by GUID_NULL in ParameterType.

This structure must be preserved during exFAT formatting, except in the case of secure wipe.

Table 10: OEM Parameter Record

Offset	Size	Description	Comments
0x00	16	ParameterType	OEM defined GUID , GUID_NULL indicate that parameter value is not used
0x10	32	ParameterValue	OEM specific

```
#define OEM_FLASH_PARAMETER_GUID 0A0C7E46-3399-4021-90C8-FA6D389C4BA2
struct
{
 GUID OemParameterType; //Value is OEM_FLASH_PARAMETER_GUID
 UINT32 EraseBlockSize; //Erase block size in bytes
 UINT32 PageSize;
 UINT32 NumberOfSpareBlocks;
 UINT32 tRandomAccess;  //Random Access Time in nanoseconds
 UINT32 tProgram;            //Program time in nanoseconds
 UINT32 tReadCycle;     //Serial read cycle time in nanoseconds
 UINT32 tWriteCycle;    //Write Cycle time in nanoseconds
 UCHAR Reserved[4];
}
FlashParameters;
```

Boot Checksum

This sector contains a repeating 32-bit checksum of the previous 11 sectors. The checksum calculation excludes VolumeFlags and PercentInUse fields in Boot Sector (bytes 106, 107, 112). The checksum is repeated until the end of the sector. The number of repetitions depends on the size of the sector.

```
UINT32 BootChecksum(const unsigned char data[], int bytes)
{
 UINT32 checksum = 0;

 for (int i = 0; i < bytes; i++)
 {
  if (i == 106 || i == 107 || i == 112)
   continue;
  checksum = (checksum << 31) | (checksum >> 1) + data[i];
 }
 return checksum;
}
```

File Allocation Table (FAT)

File Allocation Table (FAT) may contain 1 or 2 FATs, as defined in NumberOfFats field. ActiveFat field in VolumeFlags in the Main Boot Sector determines which FAT is active.

The first cluster is cluster 2, as in FAT32. Each FatEntry represents one cluster

In exFAT, FAT is not used for tracking an allocation; an Allocation Bitmap is used for this purpose. FAT is only used for keeping chains of clusters of fragmented files. If a file is not fragmented, FAT table does not need to be updated. A Stream Extensions Directory Entry should be consulted to determine if the FAT chain is valid or not. If FAT chain is not valid, it does not need to be zeroed.

Offset	Size	Description	Comments
0 (0x00)	4	FatEntry[0]	Media type (should be 0xFFFFFFF8)
4 (0x04)	4	FatEntry[1]	Must be 0xFFFFFFFF
8 (0x08)	4	FatEntry[2]	First cluster
…	…	…	…
(ClusterCount + 1) * 4	4	FatEntry[ClusterCount + 1]	Last cluster
(ClusterCount + 2) * 4	Remainder of sector	ExcessSpace	

Valid values of FAT entries:

0x00000002

ClusterCount +1 (max 0xFFFFFFF6) – next cluster in the chain

0xFFFFFFF7

bad cluster

0xFFFFFFF8

media descriptor

0xFFFFFFFF

end of file (EOF mark)

Value 0x00000000 does not mean the cluster is free, it is an undefined value.

The second FAT table (presents only in TexFAT) is located immediately after the first one and has the same size.

Related concepts

Extended File System (exFAT) on page 83

Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Directory Structure on page 89
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Defined Directory Entries on page 91
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Cluster Heap on page 98
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Directory Structure

Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT uses tree structure to describe relationship between files and directories. The root of the directory tree is defined by directory located at RootDirectoryCluster. Subdirectories are single-linked to there parents. There is no special (.) and (..) directories pointing to itself and to parent like in FAT16/FAT32.

Each directory consists of a series of directory entries. Directory entries are classified as critical/benign and primary/secondary as follows:

- Primary Directory Entries
- Critical Primary Entries
- Benign Primary Entries
- Secondary Directory Entries
- Critical Secondary Entries
- Benign Secondary Entries

Critical entries are required while benign entries are optional. Primary directory entries correspond to the entries in file system and describe main characteristics. Secondary directory entries extend the metadata associated with a primary directory entry end follow it. A group of primary/secondary entries make up a directory entry set describing a file or directory. The first directory entry in the set is a primary directory entry. All subsequent entries, if any, must be secondary directory entries.

Each directory entry derives from Generic Directory Entry template. Size of directory entry is 32 bytes.

Table 11: Generic Directory Entry Template

Offset	Size	Description	Comments
0 (0x00)	1	EntryType (see below)	
1 (0x01)	19	CustomDefined	
20 (0x14)	4	FirstCluster	0 – no cluster allocation 2..ClusterCount+1 – cluster index
24 (0x18)	8	DataLength	In bytes

Table 12: Entry Types description

Bits	Size	Description	Comments
0-4	5	Code	
5	1	Importance	0 – Critical entry, 1 – Benign entry
6	1	Category	0 – Primary entry, 1 – Secondary entry
7	1	In use status	0 – Not in use, 1 – In use

Entry Type can have the following values:

- **0x00** – End Of Directory marker. All other fields in directory entry are invalid. All subsequent directory entries are also End Of Directory markers
- **0x01-0x7F** (InUse = 0). All other fields in this entry are not defined
- **0x81-0xFF** (InUse = 1). Regular record with all fields defined.

Table 13: Generic Primary Directory Entry Template

Offset	Size	Description	Comments
0 (0x00)	1	EntryType	
1 (0x01)	1	SecondaryCount	Number of secondary entries which immediately follow this primary entry and together comprise a directory entry set. Valid value is 0..255
2 (0x02)	2	SetChecksum	Checksum of all directory entries in the given set excluding this field. See EntrySetCheckSum().
4 (0x04)	2	GeneralPrimaryFlags (see below)	
6 (0x06)	14	CustomDefined	
20 (0x14)	4	FirstCluster	
24 (0x18)	8	DataLength	

Bits	Size	Description	Comments
0	1	AllocationPossible	0-not possible (FirstCluster and DataLength undefined), 1-possible
1	1	NoFatChain	0-FAT cluster chain is valid 1-FAT cluster chain is not used (contiguous data)
2	14	CustomDefined	

All critical primary directory entries are located in root directory (except file directory entries). Benign primary directory entries are optional. If one benign primary entry is not recognized, all directory entry set is ignored.

```
// data points to directory entry set in memory
UINT16 EntrySetChecksum(const unsigned char data[], int secondaryCount)
{
 UINT16 checksum = 0;
 int bytes = (secondaryCount + 1) * 32;

 for (int i = 0; i < bytes; i++)
 {
  if (i == 2 || i == 3)
   continue;
  checksum = (checksum << 15) | (checksum >> 1) + data[i];
 }
```

```
  return checksum;
 }
```

Related concepts

Extended File System (exFAT) on page 83
Understanding of underlying mechanisms of data storage, organization and data recovery.

Volume Layout on page 84
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Defined Directory Entries on page 91
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Cluster Heap on page 98
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Defined Directory Entries

Understanding of underlying mechanisms of data storage, organization and data recovery.

Main exFAT directory entries defined in table below:

Table 14: Defined Directory Entries list

Entry Type	Primary	Critical	Code	Directory Entry Name
0x81	state: available=yes	state: available=yes	1	Allocation Bitmap
0x82	state: available=yes	state: available=yes	2	Up-case Table
0x83	state: available=yes	state: available=yes	3	Volume Label
0x85	state: available=yes	state: available=yes	5	File
0xA0	state: available=yes	state: available=no	0	Volume GUID
0xA1	state: available=yes	state: available=no	1	TexFAT Padding
0xA2	state: available=yes	state: available=no	2	Windows CE Access Control Table
0xC0	state: available=no	state: available=yes	0	Stream Extension
0xC1	state: available=no	state: available=yes	1	File Name

Read about Directory entries below:

- Allocation Bitmap Directory Entry on page 91
- Up-Case Table Directory Entry on page 92
- Volume Label Directory Entry on page 92
- File Directory Entry on page 93
- Volume GUID Directory Entry on page 95
- exFAT Padding Directory Entry on page 96
- Windows CE Access Control Table Directory Entry on page 96
- Stream Extension Directory Entry on page 96
- File Name Directory Entry on page 97

Allocation Bitmap Directory Entry

Offset	Size	Description	Comments
0 (0x00)	1	Entry type	0x81

Offset	Size	Description	Comments
1 (0x01)	1	BitmapFlags (see below)	Indicates which Allocation Bitmap the given entry describes
2 (0x02)	18	Reserved	
20 (0x14)	4	First Cluster	
24 (0x18)	8	Data Length	

Table 15: Bitmap Flags

Bits	Size	Description	Comments
0	1	BitmapIdentifier	0 – 1st bitmap, 1 - 2nd bitmap
1	7	Reserved	

The number of bitmaps and therefore a number of Bitmap Allocation entries is equal to the number of FATs. In case of TexFAT two FATs are used and bit 0 of Flags indicates which bitmap and FAT are referred.

The First Allocation Bitmap shall be used in conjunction with the First FAT and the Second Allocation Bitmap shall be used with the Second FAT. ActiveFat field in Boot Sector defines which FAT and Allocation Bitmap are active.

Bitmap size in bytes must be a number of clusters in the volume divided by 8 and rounded up.

Up-Case Table Directory Entry

Offset	Size	Description	Comments
0 (0x00)	1	Entry type	0x82
1 (0x01)	3	Reserved1	
4 (0x04)	4	TableChecksum	Up-case Table checksum
8 (0x08)	12	Reserved2	
20 (0x14)	4	FirstCluster	
24 (0x18)	8	DataLength	

The checksum is calculated against DataLength bytes of Up-case Table according to the following code:

```
UINT32 UpCaseTableChecksum(const unsigned char data[], int bytes)
{
 UINT32 checksum = 0;

 for (int i = 0; i < bytes; i++)
  checksum = (checksum << 31) | (checksum >> 1) + data[i];

 return checksum;
}
```

Volume Label Directory Entry

Offset	Size	Description	Comments
0 (0x00)	1	Entry type	0x83

Offset	Size	Description	Comments
1 (0x01)	1	CharacterCount	Length in Unicode characters (max 11)
2 (0x02)	22	VolumeLabel	Unicode string
24 (0x18)	8	Reserved	

If volume is formatted without a label, the Volume Label Entry will be present but Entry Type will be set to 0x03 (not in use).

File Directory Entry

File directory entry describes files and directories. It is a primary critical directory entry and must be immediately followed by 1 Stream Extension directory entry and from 1 to 17 File Name directory entries. Those 3-19 directory entries comprise a directory entry set describing a single file or a directory.

Offset	Size	Description	Comments
0 (0x00)	1	Entry type	0x85
1 (0x01)	1	SecondaryCount	Must be from 2 to 18
2 (0x02)	2	SetChecksum	
4 (0x04)	2	FileAttributes (see below)	
6 (0x06)	2	Reserved1	
8 (0x08)	4	CreateTimestamp	
12 (0x0C)	4	LastModifiedTimestamp	
16 (0x10)	4	LastAccessedTimestamp	
20 (0x14)	1	Create10msIncrement	0..199
21 (0x15)	1	LastModified10msIncrement	0..199
22 (0x16)	1	CreateTimezoneOffset	Offset from UTC in 15 min increments
23 (0x17)	1	LastModifiedTimezoneOffset	Offset from UTC in 15 min increments
24 (0x18)	1	LastAccessedTimezoneOffset	Offset from UTC in 15 min increments
25 (0x19)	7	Reserved2	

Table 16: File Attributes

Bits	Size	Description	Comments
0	1	ReadOnly	
1	1	Hidden	
2	1	System	
3	1	Reserved1	
4	1	Directory	
5	1	Archive	

Bits	Size	Description	Comments
6	10	Reserved2	

Table 17: Time stamp Format

Bits	Size	Description	Comments
0-4	5	Seconds (as number of 2-second intervals)	0..29 29 represents 58 seconds
5-10	6	Minutes	0..59
11-15	5	Hour	0..23
16-20	5	Day	1..31
21-24	4	Month	1..12
25-31	7	Year (as offset from 1980)	0 represents 1980

Time stamp format records seconds as 2 seconds intervals, so 10ms increments are used to increase precision from 2 seconds to 10 milliseconds. The valid values are from 0 to 199 in 10ms intervals which are added to correspondent time stamp. Time stamp is recorded in local time.

Time zone offset is expressed in 15 minutes increments.

Table 18: Time Zone Offset Tablet

Timezone Offset field	TZ Offset	Time Zone	Comments
128 (0x80)	UTC	Greenwich Standard Time	
132 (0x84)	UTC+01:00	Central Europe Time	
136 (0x88)	UTC+02:00	Eastern Europe Standard Time	
140 (0x8C)	UTC+03:00	Moscow Standard Time	
144 (0x90)	UTC+04:00	Arabian Standard Time	
148 (0x94)	UTC+05:00	West Asia Standard Time	
152 (0x98)	UTC+06:00	Central Asia Standard Time	
156 (0x9C)	UTC+07:00	North Asia Standard Time	
160 (0xA0)	UTC+08:00	North Asia East Standard Time	
164 (0xA4)	UTC+09:00	Tokyo Standard Time	
168 (0xA8)	UTC+10:00	West Pacific Standard Time	
172 (0xAC)	UTC+11:00	Central Pacific Standard Time	
176 (0xB0)	UTC+12:00	New Zealand Standard Time	
180 (0xB4)	UTC+13:00	Tonga Standard Time	

Timezone Offset field	TZ Offset	Time Zone	Comments
208 (0xD0)	UTC-12:00	Dateline Standard Time	
212 (0xD4)	UTC-11:00	Samoa Standard Time	
216 (0xD8)	UTC-10:00	Hawaii Standard Time	
220 (0xDC)	UTC-09:00	Alaska Standard Time	
224 (0xE0)	UTC-08:00	Pacific Standard Time	
228 (0xE4)	UTC-07:00	Mountain Standard Time	
232 (0xE8)	UTC-06:00	Central Standard Time	
236 (0xEC)	UTC-05:00	Eastern Standard Time	
240 (0xF0)	UTC-04:00	Atlantic Standard time	
242 (0xF2)	UTC-03:30	Newfoundland Standard Time	
244 (0xF4)	UTC-03:00	Greenland Standard Time	
248 (0xF8)	UTC-02:00	Mid-Atlantic Standard Time	
252 (0xFC)	UTC-01:00	Azores Standard Time	

Volume GUID Directory Entry

In following table presented a benign primary directory entry and may not present in a file system.

Offset	Size	Description	Comments
0 (0x00)	1	EntryType	0xA0
1 (0x01)	1	SecondaryCount	Must be 0x00
2 (0x02)	2	SetChecksum	
4 (0x04)	2	GeneralPrimaryFlags (See below)	
6 (0x06)	16	VolumeGuid	All values are valid except null GUID {00000000-0000-0000-0000-00000000
22 (0x16)	10	Reserved	

Table 19: Primary Flags Definitions

Bits	Size	Description	Comments
0	1	AllocationPossible	Must be 0
1	1	NoFatChain	Must be 0
2	14	CustomDefined	

exFAT Padding Directory Entry

Offset	Size	Description	Comments
0 (0x00)	1	EntryType	0xA1
1 (0x01)	31	Reserved	

Remember:

exFAT 1.00 does not define TexFAT Padding directory entry. TexFAT Padding directory entries are only valid in the first cluster of directory and occupy every directory entry of the cluster. The implementations should not move TexFAT Padding directory entries.

Windows CE Access Control Table Directory Entry

Offset	Size	Description	Comments
0 (0x00)	1	EntryType	0xA2
1 (0x01)	31	Reserved	

Remember:

exFAT 1.00 does not define Windows CE Access Control Table Directory Entry.

Stream Extension Directory Entry

Offset	Size	Description	Comments
0 (0x00)	1	EntryType	0xC0
1 (0x01)	1	GeneralSecondaryFlags (see below)	
2 (0x02)	1	Reserved1	
3 (0x03)	1	NameLength	Length of Unicode name contained in subsequent File Name directory entries
4 (0x04)	2	NameHash	Hash of up-cased file name
6 (0x06)	2	Reserved2	
8 (0x08)	8	ValidDataLength	Must be between 0 and DataLength
16 (0x10)	4	Reserved3	
20 (0x14)	4	FirstCluster	
24 (0x18)	8	DataLength	For directories maximum 256 MB

Table 20: Secondary Flags Definitions

Bits	Size	Description	Comments
0	1	AllocationPossible	Must be 1
1	1	NoFatChain	
2	14	CustomDefined	

Stream Extension directory entry must immediately follow the File directory entry in the set. It could be only one Stream Extension entry in the set. If NoFatChain flag is set, all allocated clusters are contiguous.

The NameHash field facilitates the purpose of fast file name comparison and is performed on up-cased file name. NameHash verify against a mismatch, however matching hashes cannot guarantee the equality of file names. If name hashes match, a subsequent full name comparison must be performed.

```
// fileName points to up-cased file name
UINT16 NameHash(WCHAR *fileName, int nameLength)
{
  UINT16 hash = 0;
  unsigned char *data = (unsigned char *)fileName;

  for (int i = 0; i < nameLength * 2; i++)
    hash = (hash << 15) | (hash >> 1) + data[i];

  return hash;
}
```

ValidDataLength determines how much actual data written to the file. Implementation shall update this field as data has been written. The data beyond the valid data length is undefined and implementation shall return zeros.

File Name Directory Entry

Offset	Size	Description	Comments
0 (0x00)	1	EntryType	0xC1
1 (0x01)	1	GeneralSecondaryFlags (see below)	
2 (0x02)	30	FileName	

Table 21: Secondary Flags Definitions

Bits	Size	Description	Comments
0	1	AllocationPossible	Must be 0
1	1	NoFatChain	Must be 0
2	14	CustomDefined	

File Name directory entries must immediately follow the Steam Extension directory entry in the number of NameLength/15 rounded up. The maximum number of File Name entries is 17, each can hold up to 15 Unicode characters and the maximum file name length is 255. Unused portion of FileName field must be set to 0x0000.

Table 22: Invalid File Name Characters

Character Code	Character	Description
0x0000 – 0x001F		Control codes
0x0022	"	Quotation mark
0x002A	*	Asterisk
0x002F	/	Forward slash
0x003A	:	Colon
0x003C	<	Less than
0x003E	>	Greater than
0x003F	?	Question mark
0x005C	\	Back slash
0x007C	|	Vertical bar

Related concepts

Extended File System (exFAT) on page 83
Understanding of underlying mechanisms of data storage, organization and data recovery.

Volume Layout on page 84
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Directory Structure on page 89
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Cluster Heap on page 98
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Cluster Heap

Understanding of underlying mechanisms of data storage, organization and data recovery.

The cluster heap is a set of clusters which hold data in exFAT. It contains:

- Root Directory
- Files
- Directories
- Allocation Bitmap on page 98
- Up-case Table on page 99

The allocation status of clusters in cluster heap is tracked by Bitmap Allocation Table which itself located inside the cluster heap.

Allocation Bitmap

Allocation Bitmap keeps track of the allocation status of clusters. FAT does not serve this purpose as in FAT16/FAT32 file system. Allocation Bitmap consists of a number of 8 bit bytes which can be treated as a sequence of bits. Each bit in bitmap corresponds to a data cluster. If it has a value of 1, the cluster is occupied, if 0 - the cluster is free. The least significant bit of bitmap table refers to the first cluster, i.e. cluster 2.

Offset	Size	Description	Comments
0x00	1	1st byte	Clusters 2-9
0x01	1	2nd byte	Clusters 10-17

Offset	Size	Description	Comments
0x02	1	3rd byte	Clusters 18-25
...			

Bitmap allocation table resides in cluster heap and referred by Bitmap Directory entry in root directory.

In exFAT could be 2 Bitmap Allocation tables, otherwise there will be only one bitmap. The NumberOfFats field in Boot Sectors determines the number of valid Allocation Bitmap directory entries in the root directory and the number of Allocation Bitmaps.

Up-case Table

Up-case table contains data used for conversion from lower-case to upper-case characters. File Name Directory Entry uses Unicode characters and preserves case when storing file name. exFAT itself is case insensitive, so it needs to compare file names converted to the upper-case during search operations.

Normally Up-case table is located right after Bitmap Allocation table but can be placed anywhere is the cluster heap. It has a corresponding primary critical directory entry in the root directory.

Up-case Table is an array of Unicode characters, an index of which represents the Unicode characters to be up-cased and the value is the target up-cased character. The Up-case Table shall contain at least 128 mandatory Unicode mappings. If implementation supports only mandatory 128 characters it may ignore the rest of Up-case Table. When up-casing file names such implementation shall up-case only characters from the mandatory 128 characters set and leave other characters intact. When comparing file names which are different only by characters in non-mandatory set, those file names shall be treated as equal.

Index	Value	Comments
0x0000	0x0000	
0x0001	0x0001	
0x0002	0x0002	
...	...	..
0x0041	0x0041	'A' is mapped into itself (identity mapping)
0x0042	0x0042	'B' is mapped into itself
..	..	..
0x061	0x041	'a' is mapped into 'A' (non-identity mapping)
0x062	0x0042	'b' is mapped into 'B'
..	..	..

Up-case Table can be written in compressed format where the series of identity mappings is represented with 0xFFFF followed by the number of identity mappings.

```
Mandatory First 128 Up-case Table Entries

Index |  Table Entries
_____________________________________________________________________________
0000  -  0000 0001 0002 0003 0004 0005 0006 0007 0008 0009 000A 000B 000C 000D 000E 000F
0010  -  0010 0011 0012 0013 0014 0015 0016 0017 0018 0019 001A 001B 001C 001D 001E 001F
0020  -  0020 0021 0022 0023 0024 0025 0026 0027 0028 0029 002A 002B 002C 002D 002E 002F
0030  -  0030 0031 0032 0033 0034 0035 0036 0037 0038 0039 003A 003B 003C 003D 003E 003F
0040  -  0040 0041 0042 0043 0044 0045 0046 0047 0048 0049 004A 004B 004C 004D 004E 004F
0050  -  0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 005A 005B 005C 005D 005E 005F
0060  -  0060 0041 0042 0043 0044 0045 0046 0047 0048 0049 004A 004B 004C 004D 004E 004F
0070  -  0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 005A 007B 007C 007D 007E 007F
```

Remember:

Non-identity mappings are highlighted in **bold**.

```
Mandatory First 128 Up-case Table Entries in compressed format

Index |  Table Entries
_____________________________________________________________________________
0000  -  FFFF 0061 0041 0042 0043 0044 0045 0046 0047 0048 0049 004A 004B 004C 004D 004E
0010  -  004F 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 005A FFFF 0005
```

The first highlighted group describes that first 0x0061 characters (0x0000-0x0060) have identity mappings. The next character after it (0x0061) maps to 0x0041 etc. until the next compressed group is encountered.

Remember:

The first highlighted **in bold** group describes that first 0x0061 characters (0x0000-0x0060) have identity mappings. The next character after it (0x0061) maps to 0x0041 etc. until the next compressed group is encountered.

Related concepts

Extended File System (exFAT) on page 83
Understanding of underlying mechanisms of data storage, organization and data recovery.

Volume Layout on page 84
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Directory Structure on page 89
Understanding of underlying mechanisms of data storage, organization and data recovery.

exFAT Defined Directory Entries on page 91
Understanding of underlying mechanisms of data storage, organization and data recovery.

Erase Disk Concept

Understanding of underlying mechanisms of data storage organization and data erasure.

Erasing Confidential Data

Modern methods of data encryption are deterring network attackers from extracting sensitive data from stored database files.

Attackers (who want to retrieve confidential data) become more resourceful and look for places where data might be stored temporarily. For example, the Windows **DELETE** command merely changes the files attributes and location so that the operating system will not look for the file located on FAT/exFAT volumes. The situation with NTFS file system is similar.

One avenue of attack is the recovery of data from residual data on a discarded hard drive. When deleting confidential data from hard drives, removable disks or USB devices, it is important to extract all traces of the data so that recovery is not possible.

Most official guidelines regarding the disposal of confidential magnetic data do not take into account the depth of today's recording densities nor the methods used by the OS when removing data.

Removal of confidential personal information or company trade secrets in the past might have been performed using the **FORMAT** command or the **FDISK** command. Using these procedures gives users a sense of confidence that the data has been completely removed.

When using the **FORMAT** command Windows displays a message like this: `Formatting a disk removes all information from the disk.`

Actually the **FORMAT** utility creates new empty directories at the root area, leaving all previous data on the disk untouched. Moreover, an image of the replaced FAT tables is stored so that the **UNFORMAT** command can be used to restore them.

FDISK merely cleans the Partition Table (located in the drive's first sector) and does not touch anything else.

Moreover, most of hard disks contain hidden zones (disk areas that cannot be accessed and addressed on a logical access level).

International Standards in Data Removal

KillDisk conforms to more than 20 international standards for clearing and sanitizing data (US DoD 5220.22-M, Gutmann and others). You can be sure that sensitive information is destroyed forever once you erase a disk with KillDisk.

KillDisk is a professional security application that destroys data permanently on any computer that can be started using a bootable CD/DVD/BD or USB Flash Disk. Access to the drive's data is made on the physical level via the BIOS (Basic Input-Output System) bypassing the operating system's logical drive structure organization. Regardless of the operating system, file systems, or type of machine, this utility can destroy all the data on all storage devices. It does not matter which operating systems or file systems are located on the machine.

Related information

Sanitization Types on page 109
Disk Hidden Zones on page 110

Secure Erase Concepts

Secure Erase for SSD is used to permanently delete data from the media and to restore the drive's speed if it starts to drop to noticeably lower performance than stated (at the same time, we don't consider SLC-caching and other "official" reasons for speed reduction since it's hardware drive features).

The essence of the problem that Secure Erase can solve: drive began to work slowly (writing and reading data). There can be a lot of reasons, some of them are related to the hardware component and some to the software component. SSDs are very different in service from classic HDDs, therefore, simply deleting data or formatting the drive does not really mean resetting the cell - you need to clear it before recording, which slows down the process of recording new data. In theory, there shouldn't be such problems, because TRIM exists - a command to clear the data marked for deletion in cells. This command only works with 2.5" and M.2 SATA drives. For drives connected to the PCIe bus (M.2 or PCIe on the motherboard) there is an analogue - Deallocate. But it happens that these functions are disabled for some reason - an OS error, a user error in setting up a disk through third-party software, or the use of non-standard OS assemblies with unknown software components. So, the disk starts to work noticeably slower and it is quite noticeable without any benchmark performance measurements.

SSDs use a number of mapping layers that hide the physical layout of the flash-based memory, as well as help in managing how flash memory data integrity and lifetime are managed. Collectively, these layers are referred to as the Flash Translation Layer (FTL).

SSDs are also over-provisioned: they contain a bit more flash memory than what they're rated for. This extra memory is used internally by the FTL as empty data blocks, used when data needs to be rewritten, and as out-of-band sections for use in the logical to physical mapping.

The mapping layers, and how the flash controller manages memory allocation, pretty much ensure that either erasing or performing a conventional hard drive type of secure erase won't ensure all data is overwritten, or even erased at all.

One example of how data gets left behind intact is due to how data is managed in an SSD. When you edit a document and save the changes, the saved changes don't overwrite the original data (an in-place update). Instead, SSDs write the new content to an empty data block and then update the logical to physical map to point to the new location. This leaves the space the original data occupied on the SSD marked as free, but the actual data is left intact. In time, the data marked as free will be reclaimed by the SSD's garbage collection system, but until then, the data could be recovered.

A conventional Secure Erase, as used with hard drives, is unable to access all of the SSD's memory location, due to the FTL and how an SSD actually writes data, which could lead to intact data being left behind.

SSD manufacturers understand the need for an easy way to sanitize an SSD, and most have implemented the ATA command, Secure Erase Unit (used with SATA-based SSDs), or the NVMe command, Format NVM (used with PCIe-based SSDs) as a fast and effective method of securely erasing an SSD.

So, SSD drives have a non-trivial system of work, therefore, the scheme for the complete destruction of data should also not be the easiest. But in reality, this is not so at all. Any SSD has a controller that is the "brain" of the drive. He not only tells the system where to write data, but also encrypts the information passing through it and stores the key with himself. If you remove (or rather replace) a given key, then all the information will turn into a random set of 1 and 0 - it will be impossible to decrypt it in any way. Just one simple action by the user can solve the problem of safe data erasure. This method is the fastest and most effective.

Note:

To protect information that is critical, both for serious organizations that are concerned about the safety of data and for public sector enterprises working with information classified as state secrets, information systems should usually use certified sanitation algorithms (US DoD 5220.22-M, Canadian OPS-II, NSA 130-2 etc.).

If you combine these two methods (replacing the key and resetting the cells), you get the perfect algorithm for obtaining a completely sterile disk in the state of its maximum performance. This, firstly, solves the problem that we raised at the very beginning, and, secondly, it can help us answer the question about the degree of drive wear.

It is important to note that some drives with built-in encryption can receive only one algorithm upon receipt of a safe erase command - it depends on the controller settings by the manufacturer. If you "reset" your SSD and compare the actual performance with the declared one, you will get the answer to this question. This procedure does not affect disk wear (which is very important). Note that these actions are designed specifically for analyzing the state of the disk, but it will not be possible to achieve a long-term increase in the read/write speed due to the peculiarities of the operation of SSD disks - the situation may depend on both the drive model and the controller firmware. And it must be noted that not all drives support encryption. In this case, the controller simply resets the cells.

Erase Methods

One Pass Zeros or One Pass Random

When using One Pass Zeros or One Pass Random standard, the number of passes is fixed and cannot be changed. When the write head passes through a sector, it writes only zeros or a series of random characters.

US DoD 5220.22-M

The write head passes over each sector three times. The first time with zeros 0x00, second time with 0xFF and the third time with random characters. There is one final pass to verify random characters by reading.

Canadian CSEC ITSG-06

The write head passes over each sector, writing a random character. On the next pass, writes the compliment of previously written character. Final pass is random, proceeded by a verify.

Canadian OPS-II

The write head passes over each sector seven times (0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, random). There is one final pass to verify random characters by reading.

British HMG IS5 Baseline

Baseline method overwrites disk's surface with just zeros 0x00. There is one final pass to verify random characters by reading.

British HMG IS5 Enhanced

Enhanced method - the write head passes over each sector three times. The first time with zeros 0x00, second time with 0xFF and the third time with random characters. There is one final pass to verify random characters by reading.

Russian GOST p50739-95

The write head passes over each sector two times: 0x00, Random. There is one final pass to verify random characters by reading.

US Army AR380-19
The write head passes over each sector three times. The first time with 0xFF, second time with zeros 0x00 and the third time with random characters. There is one final pass to verify random characters by reading.
US Air Force 5020
The write head passes over each sector three times. The first time with random characters, second time with zeros 0x00 and the third time with 0xFF. There is one final pass to verify random characters by reading.
NAVSO P-5329-26 RL
RL method - the write head passes over each sector three times: 0x01, 0x27FFFFFF, Random. There is one final pass to verify random characters by reading.
NCSC-TG-025
The write head passes over each sector three times: 0x00, 0xFF, Random. There is one final pass to verify random characters by reading.
NSA 130-2
The write head passes over each sector two times: Random, Random. There is one final pass to verify random characters by reading.
NIST 800-88
Supported three NIST 800-88 media sanitation standards:

- 1. The write head passes over each sector one time (0x00).
- 2. The write head passes over each sector one time (Random).
- 3. The write head passes over each sector three times (0x00, 0xFF, Random).

For details about this,the most secure data clearing standard, you can read the original article at the link below: http://csrc.nist.gov/publications/nistpubs/800-88/NISTSP800-88_with-errata.pdf
German VSITR
The write head passes over each sector seven times.
Bruce Schneier
The write head passes over each sector seven times: 0xFF, 0x00, Random, Random, Random, Random, Random. There is one final pass to verify random characters by reading.
Peter Gutmann
The write head passes over each sector 35 times. For details about this, the most secure data clearing standard, you can read the original article: http://www.cs.auckland.ac.nz/%7Epgut001/pubs/se%0Acure_del.html
Australian ISM-6.2.93
The write head passes over each sector once with random characters. There is one final pass to verify random characters by reading.
Secure Erase (ANSI ATA, SE)
According to National Institute of Standards and Technology (NIST) Special Publication 800-88: Guidelines for Media Sanitation, *Secure Erase* is "*An overwrite technology using firmware based process to overwrite a hard drive. Is a drive command defined in the ANSI ATA and SCSI disk drive interface specifications, which runs inside drive hardware. It completes in about 1/8 the time of 5220 block erasure.*" The guidelines also state that "*degaussing and executing the firmware Secure Erase command (for ATA drives only) are acceptable methods for purging.*" ATA Secure Erase (SE) is designed for SSD controllers. The SSD controller resets all memory cells making them empty. In fact, this method restores the SSD to the factory state, not only deleting data but also returning the original performance. When implemented correctly, this standard processes all memory, including service areas and protected sectors.
User Defined
User indicates the number of times the write head passes over each sector. Each overwriting pass is performed with a buffer containing user-defined or random characters. User Defined method allows to define any kind of new erase algorithms based on user requirements.

Wipe Disk Concepts

Wiping Unoccupied Disk's Space

You may have confidential data on your hard drive in spaces where data may have been stored temporarily.

You may also have deleted files by using the Windows Recycle Bin and then emptying it. While you are still using your local hard drive, there may be confidential information available in these unoccupied spaces.

Wiping the logical drive's deleted data does not delete existing files and folders. It processes all unoccupied drive space so that recovery of previously deleted files becomes impossible.

Installed applications and existing data are not touched by this process. When you wipe unoccupied drive space, the process is run from the bootable CD/DVD operating system. As a result, the wipe or erase process uses an operating system that is outside the local hard drive and is not impeded by Windows system caching. This means that deleted Windows system records can be wiped clean.

KillDisk wipes unused data residue from file slack space, unused sectors, and unused space in MTF records or directory records.

Wiping drive space can take a long time, so do this when the system is not being otherwise utilized. For example, this can be done overnight.

Wipe Algorithms

The process of deleting files does not eliminate them from the hard drive. Unwanted information may still be left available for recovery on the computer. A majority of software that advertises itself as performing reliable deletions simply wipes out free clusters. Deleted information may be kept in additional areas of a drive. **KillDisk** therefore offers different wipe algorithms to ensure secure deletion: overwriting with zeros, overwriting with random values, overwriting with multiple passes using different patterns and much more. **KillDisk** supports more than 20 international data sanitizing standards, including US DoD 5220.22M and the most secure Gutmann's method overwriting with 35 passes.

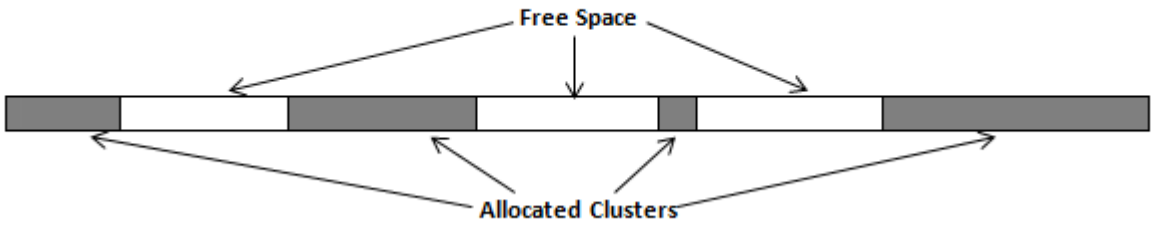


Figure 37: Disk Free Space and Allocated Clusters

Wiping File Slack Space

This relates to any regular files located on any file system. Free space to be wiped is found in the "tail" end of a file because disk space is usually allocated in 4 Kb clusters. Most files have sizes that are not 4 Kb increments and thus have slack space at their end.

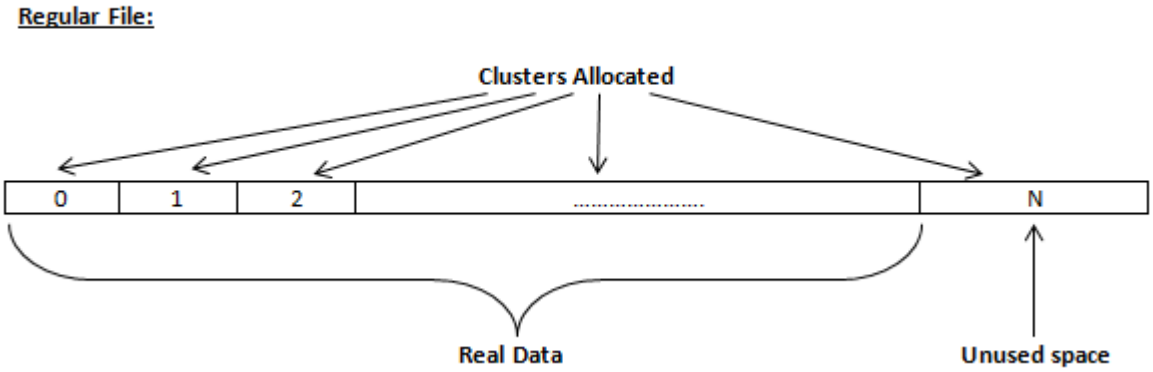


Figure 38: File Slack Space and Allocated Clusters

Specifics of Wiping Microsoft NTFS File System

NTFS Compressed Files

Wiping free space inside a file: The algorithm NTFS uses to "compress" a file operates by separating the file into compressed blocks (usually 64 Kb long). After it is processed, each of these blocks has been allocated a certain amount of space on the volume. If the compressed information takes up less space than the source file, then the rest of the space is labeled as sparse space and no space on the volume is allocated to it. Because the compressed data often doesn't have a size exactly that of the cluster, the end of each of these blocks stays as unusable space of significant size. Our algorithm goes through each of these blocks in a compressed file and wipes the unusable space, erasing previously deleted information that was kept in those areas.

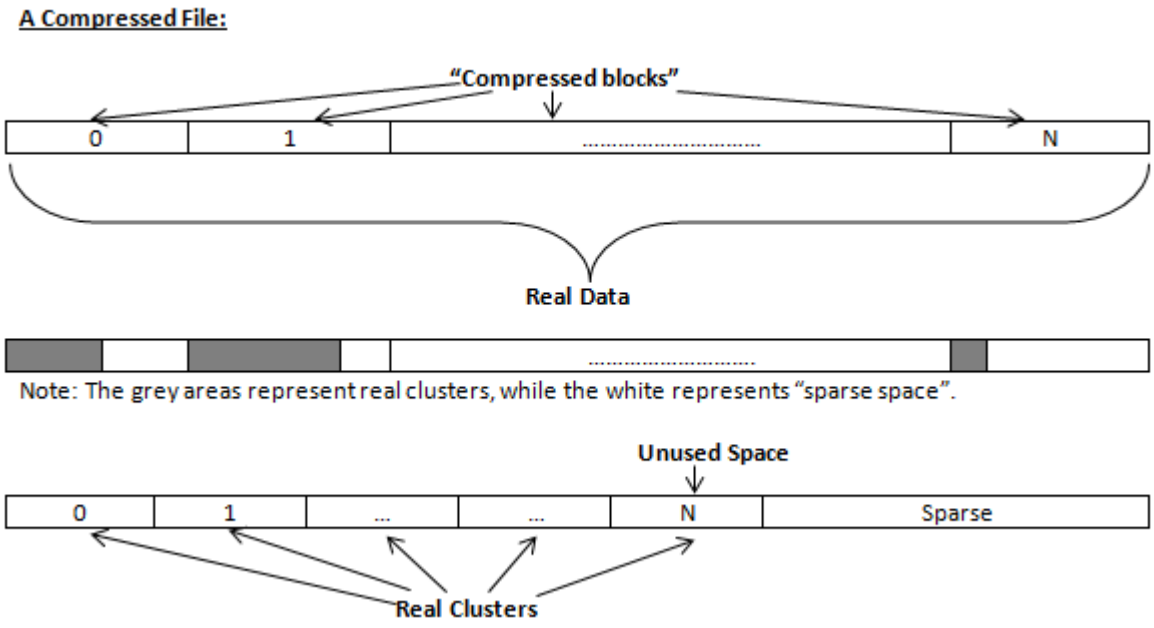


Figure 39: Compressed File Structure

The MFT (Master File Table) Area

Wiping the system information:

The MFT file contains records, describing every file on the volume. During the deletion of these files, the records of their deletion are left untouched - they are simply recorded as "deleted". Therefore file recovery software can use this information to recover anything from the name of the file and the structure of the deleted directories down to files smaller than 1Kb that are able to be saved in the MFT directly. The algorithm used by **KillDisk** wipes all of the unused information out of the MFT records and wipes the unusable space, making a recovery process impossible.

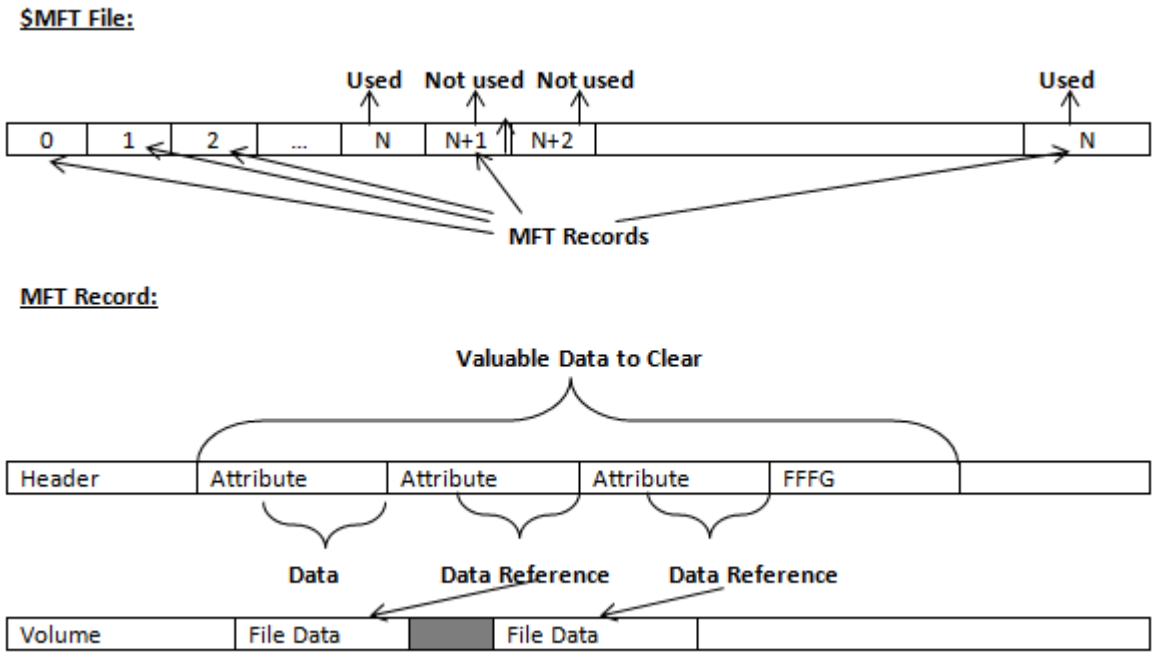


Figure 40: MFT Structure

Specifics of Wiping Microsoft FAT File System

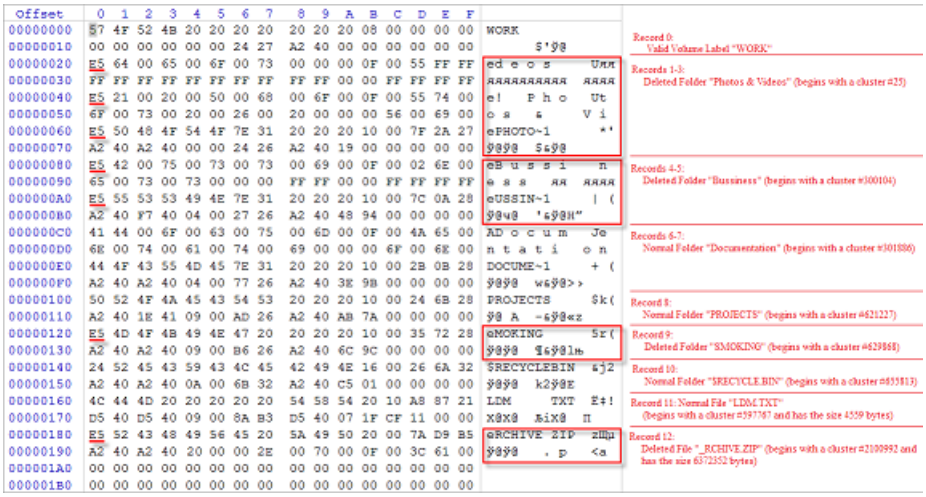
Wiping Directory Areas

Each directory on a FAT/FAT32 or an exFAT volume can be considered as a specific file, describing the contents of the directory. Inside this descriptor there are many 32-byte records, describing every file and other inner folders.

When you delete files this data is not being fully erased. It is just marked as deleted (hex symbol 0xE5). That's why data recovery software can detect and use these records to restore file names and full directory structures.

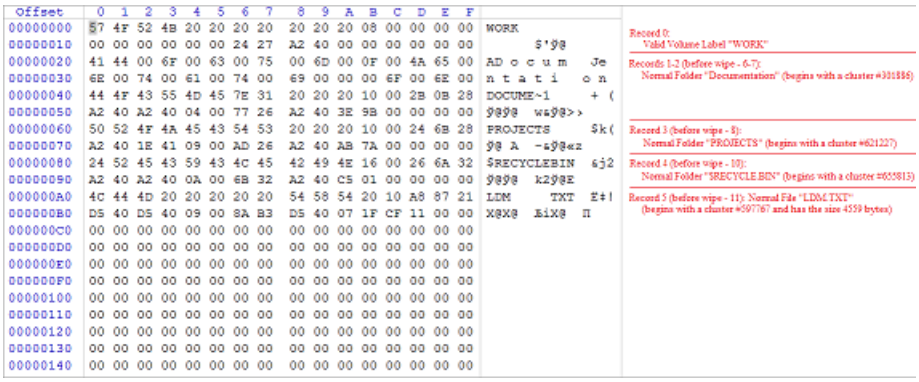
In some cases dependent on whether a space where item located has been overwritten yet or not, files and folders can be fully or partially recovered..

KillDisk makes data recovery impossible by using an algorithm that wipes out all unused information from directory descriptors. **KillDisk** not only removes unused information, but also defragments Directory Areas, thus speeding up directory access.



In this example red rectangles display deleted records.

Figure 41: FAT Directory before Wipe



In this example all deleted records removed and root folder defragmented.

Figure 42: FAT Directory after Wipe

Specifics of Wiping Apple HFS+ File System

HFS+ B-tree

A B-tree file is divided up into fixed-size nodes, each of which contains records consisting of a key and some data.

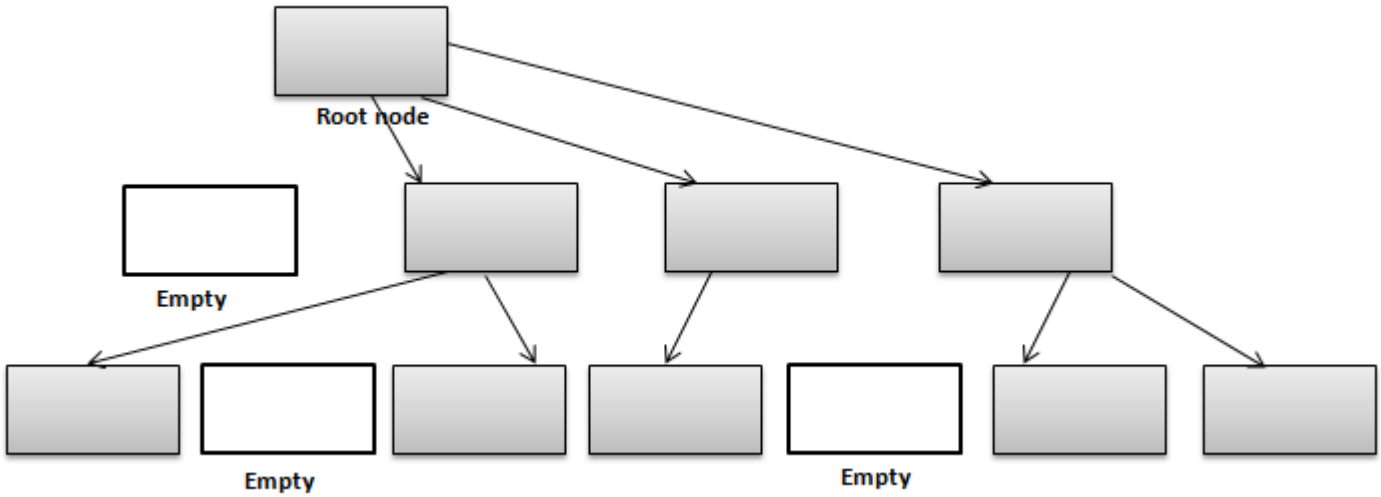


Figure 43: B-tree Structure

In the event of the deletion of a file or folder, there is a possibility of recovering the metadata of the file, (such as its name and attributes), as well as the actual data that the file consists of. **KillDisk**'s Wipe method clears out all of this free space in the system files.

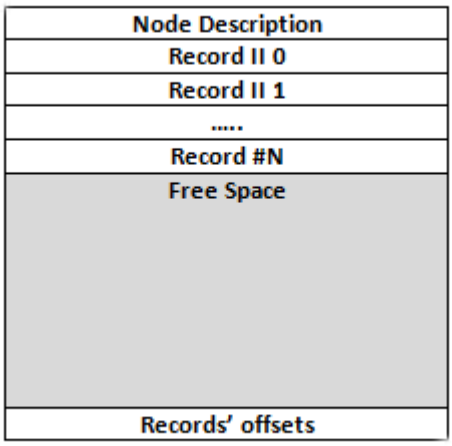


Figure 44: HFS+ System Table

Specifics of Wiping Linux Ext2/Ext3/Ext4 File Systems

A Linux Ext file system (Ext2/Ext3/Ext4) volume has a global descriptors table. Descriptors table records are called group descriptors and describe each blocks group. Each blocks group has an equal number of data blocks.

A data block is the smallest allocation unit: size vary from 1024 bytes to 4096 bytes. Each group descriptor has a blocks allocation bitmap. Each bit of the bitmap shows whether the block is allocated (1) or available (0). **KillDisk** software enumerates all groups, and for each and every block within the group on the volume checks the related bitmap to define its availability. If the Block is available, **KillDisk** wipes it using the method supplied by the user.

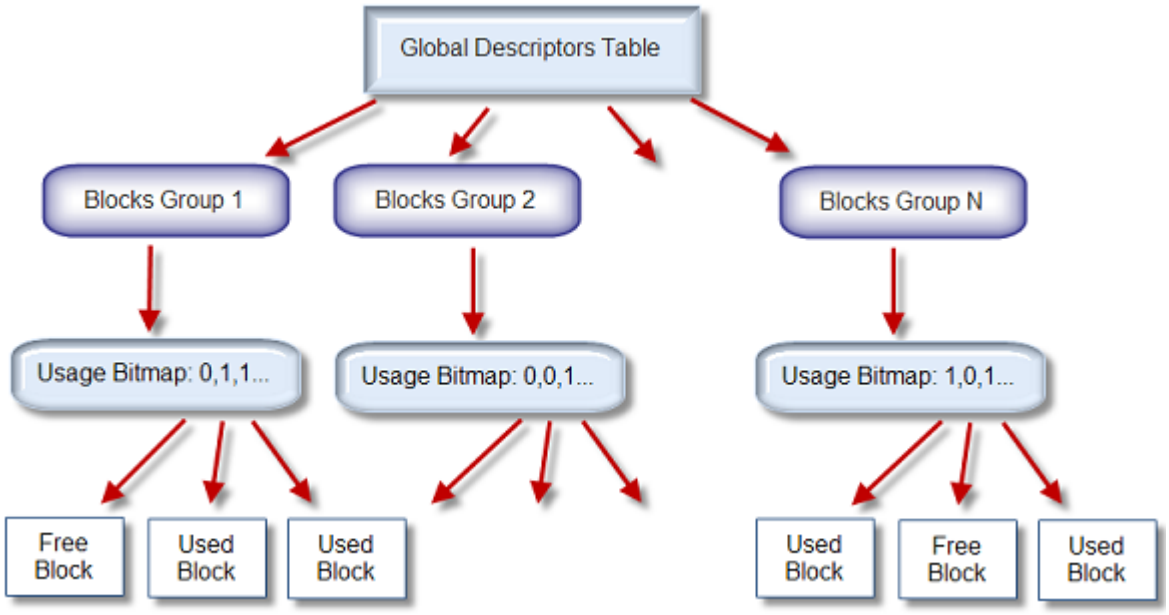


Figure 45: Ext2/Ext3/Ext4 Descriptors Table

Sanitization Types

Sanitization Types

NIST 800-88 international security standard (Guidelines for Media Sanitization) defines different types of sanitization.

Regarding sanitization, the principal concern is ensuring that data is not unintentionally released. Data is stored on media, which is connected to a system. Simply data sanitization applied to a representation of the data as stored on a specific media type.

When media is re-purposed or reaches end of life, the organization executes the system life cycle sanitization decision for the information on the media. For example, a mass-produced commercial software program contained on a DVD in an unopened package is unlikely to contain confidential data. Therefore, the decision may be made to simply dispose of the media without applying any sanitization technique. Alternatively, an organization is substantially more likely to decide that a hard drive from a system that processed Personally Identifiable Information (PII) needs sanitization prior to Disposal.

Disposal without sanitization should be considered only if information disclosure would have no impact on organizational mission, would not result in damage to organizational assets, and would not result in financial loss or harm to any individuals. The security categorization of the information, along with internal environmental factors, should drive the decisions on how to deal with the media. The key is to first think in terms of information confidentiality, then apply considerations based on media type. In organizations, information exists that is not associated with any categorized system. Sanitization is a process to render access to target data (the data subject to the sanitization technique) on the media infeasible for a given level of recovery effort. The level of effort applied when attempting to retrieve data may range widely. NIST SP 800-88 Rev. 1 Guidelines for Media Sanitization Clear, Purge, and Destroy are actions that can be taken to sanitize media. The categories of sanitization are defined as follows:

Clear

Clear applies logical techniques to sanitize data in all user-addressable storage locations for protection against simple non-invasive data recovery techniques; typically applied through the standard Read and Write commands to the storage device, such as by rewriting with a new value or using a menu option to reset the device to the factory state (where rewriting is not supported).
For HDD/SSD/SCSI/USB media this means overwrite media by using organizationally approved and validated overwriting technologies/methods/tools. The Clear pattern should be at least a single write

pass with a fixed data value, such as all zeros. Multiple write passes or more complex values may optionally be used.

KillDisk supports Clear sanitization type through the **Disk Erase** command for all R/W magnetic types of media, more than 20 international sanitation methods including custom patterns implemented and can be used.

Purge

Purge applies physical or logical techniques that render Target Data recovery infeasible using state of the art laboratory techniques.
For HDD/SSD/SCSI/USB media this means ATA SECURE ERASE UNIT, ATA CRYPTO SCRAMBLE EXT, ATA EXT OVERWRITE, ATA/SCSI SANITIZE and other low-level direct controller commands.

KillDisk supports Purge sanitization type through the **Secure Erase** command only for media types supporting ATA extensions.

Destroy

Destroy renders Target Data recovery infeasible using state of the art laboratory techniques and results in the subsequent inability to use the media for storage of data due to physical damages.
For HDD/SSD/SCSI media this means Shred, Disintegrate, Pulverize, or Incinerate by burning the device in a licensed incinerator.

It is suggested that the user categorize the information, assess the nature of the medium on which it is recorded, assess the risk to confidentiality, and determine the future plans for the media. Then, the organization can choose the appropriate type(s) of sanitization. The selected type(s) should be assessed as to cost, environmental impact, etc., and a decision should be made that best mitigates the risk to confidentiality and best satisfies other constraints imposed on the process.

Disk Erase performance

How fast erasing occurs? An actual erase speed depends on many factors:

- HDD/SSD/NVMe disk speed: RPM and SATA/SCSI/SAS/NVMe type - the most important factors
- Disk Controller speed: SAS (6 Gbps/12 Gbps), SATA III (6Gbps), (SATA II 3 Gbps), SATA I (1.5 Gbps)
- Computer overall performance (CPU, RAM) and workload (how many parallel erases occur)

For most modern computers and disks manufactured within last years SATA III standard is supported, so erase speed is limited by HDD throughput (disk write speed) only.

Our tests give the results: **10 GB per minute (in average) per pass** with decent computer configuration and disks with age of up to 5 years old.

For example, 2 TB Toshiba disk has been erased on Windows platform with one pass within 3 hours and 32 minutes, 14 TB Western Digital disk - within 18 hours 53 minutes.

The following snapshots are real-test certificates for erasing of:

1) **2 TB** Toshiba (manufactured in 2015) SATA III (6 GBps) 7200 rpm disk with One Pass Zeros and US DoD 5220.22-M (3 passes + verification) showing the average speed of **9 GB/min per pass**

2) **14 TB** (Western Digital manufactured in 2019) SATA III (6 Gbps) 7200 rpm disk with One Pass Zeros and US DoD 5220.22-M (3 passes + 10% verification) showing the average speed of **12 GB/min per pass**

Disk Hidden Zones

KillDisk is able to detect and reset Disk's Hidden Zones: HPA and DCO.

Host Protected Area

The Host Protected Area (*HPA*) is an area of a hard drive or solid-state drive that is not normally visible to an operating system. It was first introduced in the ATA-4 standard CXV (T13) in 2001.

How it works:

The IDE controller has registers that contain data that can be queried using ATA commands. The data returned gives information about the drive attached to the controller. There are three ATA commands involved in creating and using a host protected area. The commands are:

- IDENTIFY DEVICE
- SET MAX ADDRESS
- READ NATIVE MAX ADDRESS

Operating systems use the IDENTIFY DEVICE command to find out the addressable space of a hard drive. The IDENTIFY DEVICE command queries a particular register on the IDE controller to establish the size of a drive.

This register however can be changed using the SET MAX ADDRESS ATA command. If the value in the register is set to less than the actual hard drive size then effectively a host protected area is created. It is protected because the OS will work with only the value in the register that is returned by the IDENTIFY DEVICE command and thus will normally be unable to address the parts of the drive that lie within the HPA.

The HPA is useful only if other software or firmware (e. g. BIOS) is able to use it. Software and firmware that are able to use the HPA are referred to as 'HPA aware'. The ATA command that these entities use is called READ NATIVE MAX ADDRESS. This command accesses a register that contains the true size of the hard drive. To use the area, the controlling HPA-aware program changes the value of the register read by IDENTIFY DEVICE to that found in the register read by READ NATIVE MAX ADDRESS. When its operations are complete, the register read by IDENTIFY DEVICE is returned to its original fake value.

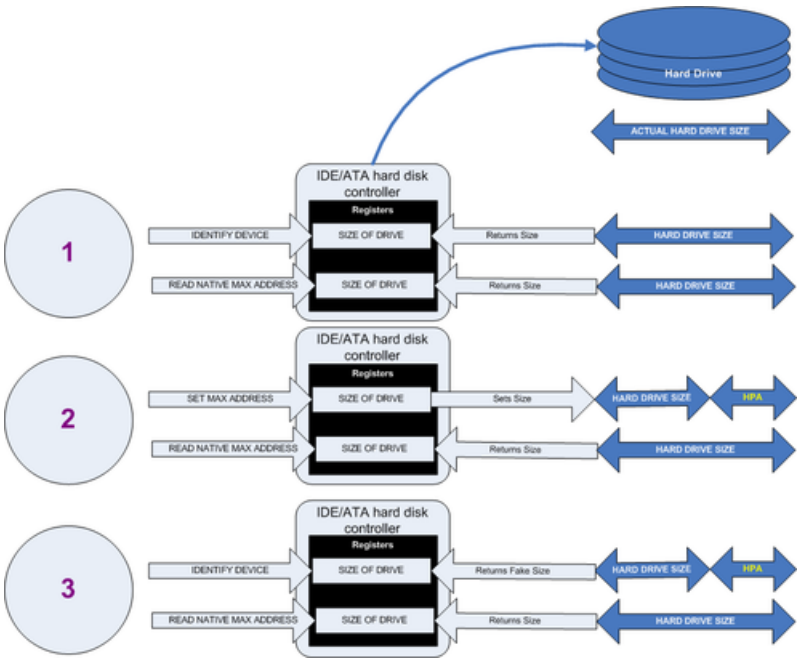


Figure 46: Creation of an HPA

The diagram shows how a host protected area (HPA) is created:

1. IDENTIFY DEVICE returns the true size of the hard drive. READ NATIVE MAX ADDRESS returns the true size of the hard drive.
2. SET MAX ADDRESS reduces the reported size of the hard drive. READ NATIVE MAX ADDRESS returns the true size of the hard drive. An HPA has been created.
3. IDENTIFY DEVICE returns the now fake size of the hard drive. READ NATIVE MAX ADDRESS returns the true size of the hard drive, the HPA is in existence.

Usage:

- At the time HPA was first implemented on hard-disk firmware, some BIOS had difficulty booting with large hard disks. An initial HPA could then be set (by some jumpers on the hard disk) to limit

the number of cylinder to 4095 or 4096 so that older BIOS would start. It was then the job of the boot loader to reset the HPA so that the operating system would see the full hard-disk storage space.

- HPA can be used by various booting and diagnostic utilities, normally in conjunction with the BIOS. An example of this implementation is the Phoenix First BIOS, which uses Boot Engineering Extension Record (BEER) and Protected Area Run Time Interface Extension Services (PARTIES). Another example is the Gujin installer which can install the bootloader in BEER, naming that pseudo-partition /dev/hda0 or /dev/sdb0; then only cold boots (from power-down) will succeed because warm boots (from Ctrl-Alt-Delete) will not be able to read the HPA.
- Computer manufacturers may use the area to contain a preloaded OS for install and recovery purposes (instead of providing DVD or CD media).
- Dell notebooks hide Dell MediaDirect utility in HPA. IBM ThinkPad and LG notebooks hide system restore software in HPA.
- HPA is also used by various theft recovery and monitoring service vendors. For example, the laptop security firm Computrace use the HPA to load software that reports to their servers whenever the machine is booted on a network. HPA is useful to them because even when a stolen laptop has its hard drive formatted the HPA remains untouched.
- HPA can also be used to store data that is deemed illegal and is thus of interest to government and police.
- Some vendor-specific external drive enclosures (Maxtor) are known to use HPA to limit the capacity of unknown replacement hard drives installed into the enclosure. When this occurs, the drive may appear to be limited in size (e.g. 128 GB), which can look like a BIOS or dynamic drive overlay (DDO) problem. In this case, one must use software utilities (see below) that use READ NATIVE MAX ADDRESS and SET MAX ADDRESS to change the drive's reported size back to its native size, and avoid using the external enclosure again with the affected drive.
- Some rootkits hide in the HPA to avoid being detected by anti-rootkit and antivirus software.
- Some NSA exploits use the HPA for application persistence.

Device Configuration Overlay

Device Configuration Overlay (*DCO*) is a hidden area on many of today's hard disk drives (HDDs). Usually when information is stored in either the DCO or host protected area (HPA), it is not accessible by the BIOS, OS, or the user. However, certain tools can be used to modify the HPA or DCO. The system uses the IDENTIFY_DEVICE command to determine the supported features of a given hard drive, but the DCO can report to this command that supported features are nonexistent or that the drive is smaller than it actually is. To determine the actual size and features of a disk, the DEVICE_CONFIGURATION_IDENTIFY command is used, and the output of this command can be compared to the output of IDENTIFY_DEVICE to see if a DCO is present on a given hard drive. Most major tools will remove the DCO in order to fully image a hard drive, using the DEVICE_CONFIGURATION_RESET command. This permanently alters the disk, unlike with the (HPA), which can be temporarily removed for a power cycle.

Usage:

The Device Configuration Overlay (DCO), which was first introduced in the ATA-6 standard, "allows system vendors to purchase HDDs from different manufacturers with potentially different sizes, and then configure all HDDs to have the same number of sectors. An example of this would be using DCO to make an 80-gigabyte HDD appear as a 60-gigabyte HDD to both the (OS) and the BIOS.... Given the potential to place data in these hidden areas, this is an area of concern for computer forensics investigators. An additional issue for forensic investigators is imaging the HDD that has the HPA and/or DCO on it. While certain vendors claim that their tools are able to both properly detect and image the HPA, they are either silent on the handling of the DCO or indicate that this is beyond the capabilities of their tool.

Virtual Disks

KillDisk provides full support for Virtual Disks - dynamic disks created and managed by:

- **Logical Disk Manager** (LDM on Windows)

- **Logical Volume Manager** (LVM on Linux)
- **Windows Storage Spaces** (WSS on Windows)

Virtual Disks are virtual devices which look like regular physical disks to all applications. These virtual devices are stored on one or more physical disks and emulate different types of volumes and RAID disk arrays not on a hardware level (inside disk controller), but on Operating System level (software emulation). Virtual devices are fully supported by the **KillDisk**. These disks will appear in **Local Devices** view like any other regular disks. When you launch an erase for the virtual disk, the progress is displayed in the same color on all components of the composite virtual drive.

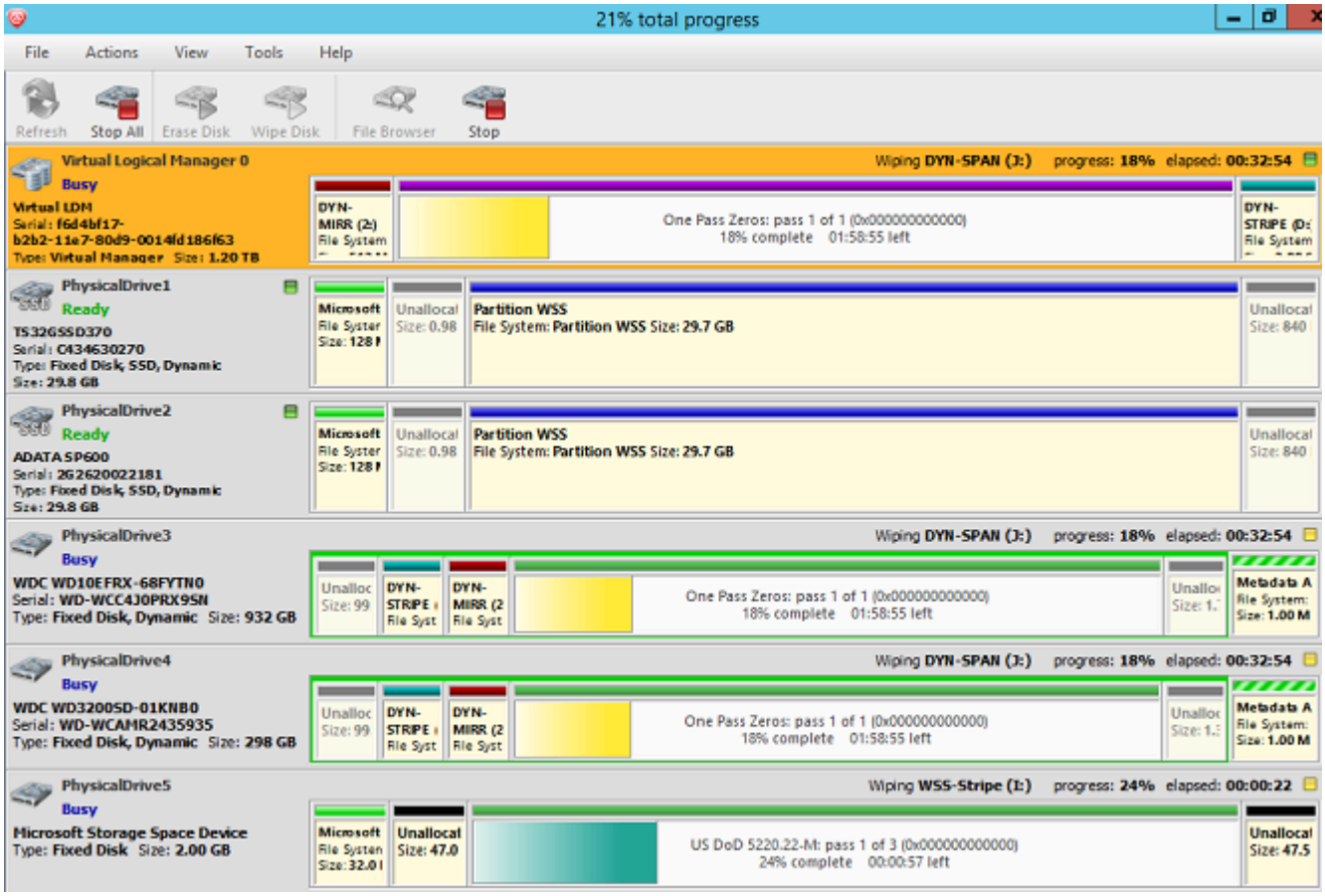


Figure 47: Erasing a Virtual Drive (Striped Disk Array)

Note: By default Virtual Disks are not being displayed in the list of devices. To display Virtual Disks go to **Preferences** > **General Settings** and turn on **Initialize virtual disks** option.

Data Recovery Concept

Understanding of underlying mechanisms of data storage organization and data recovery.

Software recovery algorithms in nutshell:

Understanding File Recovery Process
Describes basic approaches and techniques of File and Folder recovery process.

Understanding Partition Recovery Process
Describes most common partition failures and techniques of their recovery.

File Recovery

Understanding of underlying mechanisms of data storage, organization and data recovery.

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 scan for deleted entries on page 115
- Define clusters chain for the deleted entry on page 117
- Clusters chain recovery for the deleted entry on page 119

However, not every deleted file can be recovered, there are some assumptions, for sure:

- First, we assume that the file entry still exists (not overwritten with other data). The less the files have been created on the drive where the deleted file was resided, the more 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) 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 less the write operations have been performed on the drive where deleted file was resided, the more chances that the space occupied by data clusters of the deleted file has not been used for other data storage.

Important:

As general advices 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 could 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's better to save data onto another logical, removable, network or floppy drive.

Related concepts

Partition Recovery on page 120
Understanding of underlying mechanisms of data storage, organization and data recovery.

Disk scan for deleted entries on page 115
Understanding of underlying mechanisms of data storage, organization and data recovery.

Define clusters chain for the deleted entry on page 117
Understanding of underlying mechanisms of data storage, organization and data recovery.

Clusters chain recovery for the deleted entry on page 119
Understanding of underlying mechanisms of data storage, organization and data recovery.

Disk scan for deleted entries

Understanding of underlying mechanisms of data storage, organization and data recovery.

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.

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 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 folder on FAT16

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

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

4. 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   ................
```

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...^...
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.^...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).

Related concepts

File Recovery on page 113
Understanding of underlying mechanisms of data storage, organization and data recovery.

Define clusters chain for the deleted entry on page 117
Understanding of underlying mechanisms of data storage, organization and data recovery.

Clusters chain recovery for the deleted entry on page 119
Understanding of underlying mechanisms of data storage, organization and data recovery.

Define clusters chain for the deleted entry

Understanding of underlying mechanisms of data storage, organization and data recovery.

To define clusters chain we need to scan drive, going through one by one all file (NTFS) clusters or free (FAT) clusters belonging (presumably) to the file until we reach the file size equals to the total size of the selected clusters. If the file is fragmented, clusters chain will be composed of several extents in case of NTFS or we take clusters bypassing occupied ones in case of FAT.

Location of these clusters can vary depending on file system. For example, file deleted on FAT volume has its first cluster in its Root entry, the other clusters can be found in File Allocation Table. On NTFS each file has _DATA_ attribute that describes "data runs". Disassembling data runs to "extents" for each extent we have start cluster offset and number of clusters in extent, so enumerating extents, we can compose file's cluster chain.

You can try to define clusters chain manually, using low-level disk editors, however it's much simpler to use data recovery tools, like **Active@ UNDELETE**.

Example of defining clusters chain on FAT16

Lets continue examine an example for deleted file **MyFile.txt** from the previous topic.

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

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
-----------------------------------------------------------------------
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 other file's data. Now we have chain of clusters `3, 4, 5, 6` and 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 consequently. We need 4 clusters, we found 4 free consecutive clusters, so this assumption sounds reasonable, although in real life it may be not true.

Note:

There are a lot of cases where the file's data can not be successfully recovered, because clusters chain can not be defined. Most of them occur when you write another data (files, folders) on the same drive where deleted file located. You'll see these warnings while recovering data using, for example **Active@ UNDELETE**.

Example of defining clusters chain on NTFS

When recovering on NTFS part of DATA attribute called Data Runs give us location about file clusters. In most cases DATA attribute is stored inside MFT record, so if we found MFT record for the deleted file, most likely we'll be able to determine cluster's chain.

In example below DATA attribute is marked with a green color. Data Runs inside, 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.........
```

Data Runs need to be decrypted. First byte (0x31) shows how many bytes are allocated for the length of the run (0x1 in our case) and for the first cluster offset (0x3 in our case). Next, we take one byte (0x6E) that points to the length of the run. Next, we pick up 3 bytes pointing to the start cluster offset (0xEBC404).

Changing bytes order we get first cluster of the file 312555 (equals 0x04C4EB). Starting from this cluster we need to pick up 110 clusters (equals 0x6E). Next byte (0x00) tells us that no more data runs exist. Our file is not fragmented, so we have the only one data run.

Lets check, isn't there enough information about the file data? Cluster size is 512 bytes. We have 110 clusters, 110*512 = 56320 bytes. Our file size was defined as 56320 bytes, so we have enough information now to recover the file clusters.

Important:

1. DO NOT WRITE ANYTHING ONTO THE DRIVE CONTAINING YOUR IMPORTANT DATA THAT YOU HAVE JUST DELETED ACCIDENTALLY!Even data recovery software installation could 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 whole hard drive out of the computer and plug into another computer where data recovery software has been already installed.

2. DO NOT TRY TO SAVE ONTO THE SAME DRIVE DATA THAT YOU FOUND AND TRYING TO RECOVER!While saving recovered data onto the same drive where sensitive data was located, you can intrude in process of recovering by overwriting FAT records for this and other deleted entries. It's better to save data onto another logical, removable, network or floppy drive.

Related concepts

File Recovery on page 113
Understanding of underlying mechanisms of data storage, organization and data recovery.

Disk scan for deleted entries on page 115
Understanding of underlying mechanisms of data storage, organization and data recovery.

Clusters chain recovery for the deleted entry on page 119
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition Recovery on page 120
Understanding of underlying mechanisms of data storage, organization and data recovery.

Clusters chain recovery for the deleted entry

Understanding of underlying mechanisms of data storage, organization and data recovery.

After clusters chain is defined, automatically or manually, the only task left is to read and save contents of the defined clusters to another place verifying their contents.

We have a chain of clusters; we can calculate each cluster offset from the beginning of the drive, using standard formulas. After that we copy amount of data equals to the cluster size, starting from the calculated offset into the newly created file. For the last one we copy not all cluster, but reminder from the file size minus number of copied clusters multiplied by cluster size.

Formulas for calculating cluster offset could vary depending on file system.

To calculate, for example, offset of the cluster for FAT 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

On the NTFS, we have linear space so we can calculate cluster offset simply as cluster number multiplied by cluster size.

Example of recovery clusters chain on FAT16

Lets continue examine an example for deleted file **MyFile.txt** from the previous topics.

By now we have chain of clusters `3, 4, 5, 6` ready for recovering. Our cluster consists of 64 sectors, sector size is 512 bytes, so cluster size is: 64*512 = 32,768 bytes = 32 Kb First data sector is 535 (we have 1 boot sector, plus 2 copies of FAT by 251 sectors each, plus root folder 32 sectors, total 534 occupied by system data sectors). Clusters 0 and 1 do not exist, so first data cluster is 2. Cluster number 3 is next to cluster 2, i.e. is located 64 sectors behind the first data sector (535). i.e. 535 + 64 = 599 sector, 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 3 cluster, or 599 sector:

```
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
```

All we need to do is just copy 112,435 bytes starting from this place because clusters chain is consecutive. If it was not - we would need to re-calculate offset of each found cluster, and copy 3 times by 64*512 = 32768 bytes starting from each cluster offset, and then from the last cluster copy reminder: 14,131 bytes that is calculated as 112,435 bytes - (3 * 32768 bytes).

Example of recovery clusters chain on 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
```

Here is our data. What's left to do is just reading from this point 110 clusters (56320 bytes) and then copy them to another location. Data recovery is complete now.

Important:

DO NOT SAVE ONTO THE SAME DRIVE DATA THAT YOU FOUND AND TRYING TO RECOVER! process of recovering by overwriting FAT records for this and other deleted entries. It's better to save data onto another logical, removable, network or floppy drive.

Related concepts

File Recovery on page 113
Understanding of underlying mechanisms of data storage, organization and data recovery.

Disk scan for deleted entries on page 115
Understanding of underlying mechanisms of data storage, organization and data recovery.

Define clusters chain for the deleted entry on page 117
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition Recovery on page 120
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition Recovery

Understanding of underlying mechanisms of data storage, organization and data recovery.

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 OS to boot - if you have ever seen the famous "NTLDR is missing ..." error, you understand the situation.

When using **Active@ UNDELETE**, 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@ UNDELETE, Active@ UNERASER software both implement this approach.

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.

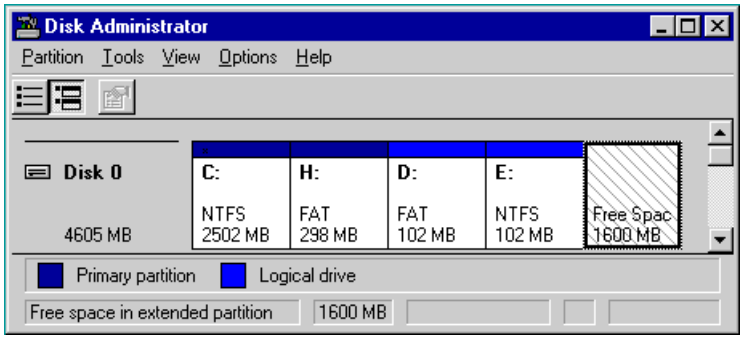
Other Partition Recovery Topics

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

- Damaged MBR on page 122

- Partition is deleted or Partition Table is damaged on page 124
- Partition Boot Sector is damaged on page 125
- Missing or Corrupted System Files on page 127

For these topics the following disk layout will be used:



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))

Damaged MBR

Understanding of underlying mechanisms of data storage, organization and data recovery.

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.

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?.?#d#2N.¦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?
```

What will happen if the first sector has been damaged (by virus, for example)?

Lets overwrite the first 16 bytes with zeros.

```
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??.±.
```

When we try to boot after hardware testing procedures, we see just blank screen without any messages. It means the piece of code at the beginning of the MBR could not be executed properly. That's why even error messages could not be displayed. However, if we boot from the floppy, we can see FAT partition, files on it and we are able to perform standard operations like file copy, program execution... It happens because in our example only part of the MBR has been damaged which does not allow the system to boot properly. However, the partition table is safe and we can access our drives when we boot from the operating system installed on the other drive.

What will happen if sector signature (last word 0x55AA) has been removed or damaged?

Lets write zeros to the location of sector signature.

```
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 an error message like "Operating System not found".

Thus the first thing if computer does not boot is to run Disk Viewer and check the first physical sector on HDD, whether it looks like valid MBR or not:

- check, may be it's 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 disk signature (0x55AA) is present or not

The simplest way to repair or re-create MBR is to run Microsoft's standard utility called FDISK with a parameter **/MBR**, like

```
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 start-up 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 fix MBR.

Also you can use third party MBR recovery software or if you've created MBR backup, restore it from there (Active@ Partition Recovery has such capabilities).

What will happen if the first sector is bad/unreadable?

Most likely we'll get the same black screen, which we got when trying to boot. When you try to read it using Disk Viewer/Editor you should get an error message saying that sector is unreadable. In this case recovery software is unable to help you to bring HDD 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), and in case if something is found - display them and give the user an opportunity to save important data to another location. Software, like **Active@ UNDELETE**, **Active@ UNERASER** will help you here.

Related concepts

Partition Recovery on page 120
Understanding of underlying mechanisms of data storage, organization and data recovery.

Missing or Corrupted System Files on page 127
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition is deleted or Partition Table is damaged on page 124
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition Boot Sector is damaged on page 125

Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition is deleted or Partition Table is damaged

Understanding of underlying mechanisms of data storage, organization and data recovery.

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?.?#d#2N.¦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 and one empty entries:

- 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.

Lets consider the situations which cause computer to hang up while booting or data loss.

What will happen if no partition has been set to the Active state (Boot Indicator=0x80)?

Lets remove Boot Indicator from the first partition:

```
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". It means that the loader cannot determine which partition is system and active to pass control to.

What will happen if partition has been set to the Active state (Boot Indicator=0x80) but there are no system files on that partition?

(it could happen if we had used for example FDISK and selected not the proper active partition).

Loader will try to boot from there, fails, try to boot again from other devices like floppy, and if fails to boot again, we'll see an error message like "Non-System Disk or Disk Error".

What will happen if partition entry has been deleted?

If it has been deleted, next two partitions will move one line up in the partition table.

```
  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?.?#d#2N.¦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 previous second (FAT) partition becomes the first and the loader will try to boot from it. And if it's not a system partition, we'll get the same error messages.

What will happen if partition entry has been damaged?

Let's 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?.?#d#2N.¦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 and we'll get an error message like "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.

How can recovery software help you in the above-mentioned scenarios?

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 partition boot sector is so important?

Because if recovery software finds it, all necessary parameters to reconstruct partition entry in the Partition Table are there. (see Partition Boot Sector is damaged on page 125 topic for details).

What would happen if partition entry had been deleted then recreated with other parameters 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 (for example, Active@ Partition Recovery and Active@ UNERASER can do it) before, 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 disk surface and try to reconstruct the previously deleted partition information from the pieces of left information (i.e. perform virtual partition recovery). However it is not guaranteed that you can recover something.

Related concepts

Partition Recovery on page 120
Understanding of underlying mechanisms of data storage, organization and data recovery.

Missing or Corrupted System Files on page 127
Understanding of underlying mechanisms of data storage, organization and data recovery.

Damaged MBR on page 122
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition Boot Sector is damaged on page 125
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition Boot Sector is damaged

Understanding of underlying mechanisms of data storage, organization and data recovery.

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
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
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 00  C7 06 5B 00 10 00 B8 00   .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 systemdis
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?
```

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, 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 as half as less successful than on NTFS.

What will happen if Partition Boot Sector is damaged or bad/unreadable?

Lets fill up with zeros several lines of Partition Boot Sector:

```
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 not bootable.

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 backup of MBR, Partition Table and Volume Boot Sectors so that if for some reason it fails to boot, you can always restore your partition information and have an access to files/folders on that partition.

What to do 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..

How can we fix NTFS boot sector using standard Windows NT/2000/XP tools?

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

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

How can recovery software help you 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.

Related concepts

Partition Recovery on page 120
Understanding of underlying mechanisms of data storage, organization and data recovery.

Missing or Corrupted System Files on page 127
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition is deleted or Partition Table is damaged on page 124
Understanding of underlying mechanisms of data storage, organization and data recovery.

Damaged MBR on page 122
Understanding of underlying mechanisms of data storage, organization and data recovery.

Missing or Corrupted System Files

Understanding of underlying mechanisms of data storage, organization and data recovery.

For Operating System to boot properly, system files 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, damaged by virus, Windows will be unable to boot. You'll see error messages like "NTLDR is missing ...".

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

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

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

Emergency Repair Process

To proceed with Emergency Repair Process, you need Emergency Repair Disk (ERD). This disk is recommended to create after you install and customize Windows. To create it, use the "Backup" utility from System Tools. You can use the ERD to repair 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 Windows bootable disks or CD, and choose 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 Repair option, when system suggests you to proceed with installation or repairing and then press **C** to run Recovery Console. You will be asked to which system you want to log on to and then for **Administrator'**s password, and after you logged on - you can display drive's contents, check the existence and safety of critical files and, for example, copy them back 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.

Related concepts

Partition Recovery on page 120
Understanding of underlying mechanisms of data storage, organization and data recovery.

Damaged MBR on page 122
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition is deleted or Partition Table is damaged on page 124
Understanding of underlying mechanisms of data storage, organization and data recovery.

Partition Boot Sector is damaged on page 125
Understanding of underlying mechanisms of data storage, organization and data recovery.

Glossary

ANSI

It's a repertoire of character encodings that include (most of) the original 96 ASCII character set, plus up to 128 additional characters.

ASCII

An acronym for American Standard Code for Information Interchange, is a character encoding standard for electronic communication. ASCII codes represent text in computers, telecommunications equipment, and other devices. The ASCII standard is a seven-bit code with no parity guidelines, containing 128 code positions.

Dynamic Disk

A dynamic storage made out of whole or part of physical disk to increase performance and reliability.

Extended Partition

A hard disk may contain only one extended partition; the extended partition can be subdivided into multiple logical partitions. In DOS/Windows systems, each logical partition may then be assigned an additional drive letter.

File Signature

Set of unique file properties, that allows

Virtual partition

A virtual copy of a volume (logical drive) using a defined geometry that emulates a real logical drive or partition

Virtual disk

A virtual copy of a physical disk using a defined disk geometry that uses real physical disk as a source but access it

Virtual RAID array

Software layer that sits above assembled physical disks that were part of a hardware RAID system.

Boot record

The information about primary partitions and an extended partition contained in the Partition Table. See Master Boot Record (MBR) on page 51.

Boot partition

Name commonly used for the partition that contains the start-up files.

Boot sector

Part of a hard disc, floppy disc, or similar data storage device that contains code for bootstrapping programs (usually, but not necessarily, operating systems) stored in other parts of the disc.

Data storage device

See physical device.

Disk geometry

Set of disk attributes that specify format, partitioning etc. of a disk

Drive letter

Abstraction at the user level to distinguish one disk or partition from another. For example, the path C:\WINDOWS\represents a directory WINDOWS on the partition represented by C:.

FAT (File Allocation Table)

File that contains the records of every other file and directory in a FAT-formatted hard disk drive. The operating system needs this information to access the files. There are FAT32, FAT16 and FAT versions.

File system

Method in which files are named and where they are placed logically for storage and retrieval in a computer. Under scope of this document, one of the Microsoft Windows file systems, such as FAT12, FAT16, FAT32 and NTFS.

Logical drive

Partitioned space on a physical device.

Partition (disk)

Hard disk's storage space divided into independent parts.

Physical device

Device for storing data, that can be connected internally (Hard Drive) or externally (USB Flash card, USB Hard Drive).

Physical device geometry

see Disk Geometry

Master Boot Record (MBR)

All disks start with a boot sector. When you start the computer, the code in the MBR executes before the operating system is started. The location of the MBR is always track (cylinder) 0, side (head) 0, and sector 1. The MBR contains a file system identifier.

MBR

MBR (Master Boot Record)

All disks start with a boot sector. When you start the computer, the code in the MBR executes before the operating system is started. The location of the MBR is always track (cylinder) 0, side (head) 0, and sector 1. The MBR contains a file system identifier.

MFT or MFT records (Master File Table)

File that contains the records of every other file and directory in an NTFS-formatted hard disk drive. The operating system needs this information to access the files.

System partition

Name commonly used for the partition that contains the operating system files.

Virtual RAID Virtual Disk Array

Software layer that sits above assembled physical disks that were part of a hardware RAID system.

Volume boot record

First sector of a data storage device that has not been partitioned, or the first sector of an individual partition on a data storage device that has been partitioned. It contains code to load and invoke the operating system (or other standalone program) installed on that device or within that partition.

Uninstall Active@ Disk Editor

How to uninstall Active@ Disk Editor.

Active@ Disk Editor software comes with a standard installer\uninstaller accessible from the Control Panel.

To uninstall the software:

1. Open Control Panel;
2. Navigate **Programs & Features** > **Uninstall** or change a program;
3. Select **Active@ Disk Editor** section and click **Uninstall** or just double click it;
4. Click **Yes** to confirm uninstall process;