TZWorks® FAT/exFAT Analysis (*fata*) Utility Users Guide

Abstract

fata is a standalone, command-line tool that parses the FAT32 and exFAT filesystems. The results are displayed in a delimited text type or CSV format where one file or folder is displayed per line. ***fata*** requires no installation on the target computer and can be run directly from a removable device. The algorithm specifically targets the raw disk sectors and/or volume clusters to parse the filesystem. ***fata*** has binary versions that run in Windows, Linux and macOS.

www.tzworks.com

Contact Info: info@tzworks.com

*Document applies to v0.13 of **fata***

Updated: Apr 25, 2025

Table of Contents

TZWorks® FAT Analysis (*fata*) Utility Users Guide

Webpage: http://www.tzworks.com/prototype_page.php?proto_id=55
Contact Information: info@tzworks.com

1 Introduction

This tool was created to be light weight and assist in the analysis of *FAT32* and *exFAT* filesystems while looking at only the raw disk sectors or volume clusters. The tool's algorithm is operating system agnostic when parsing the files or folders, and since it has no installation requirements, it is useful in various live collection and triaging situations. Furthermore, the tool's architecture was designed to be extensible so as to act as an architecture framework for future *FAT* filesystem work.

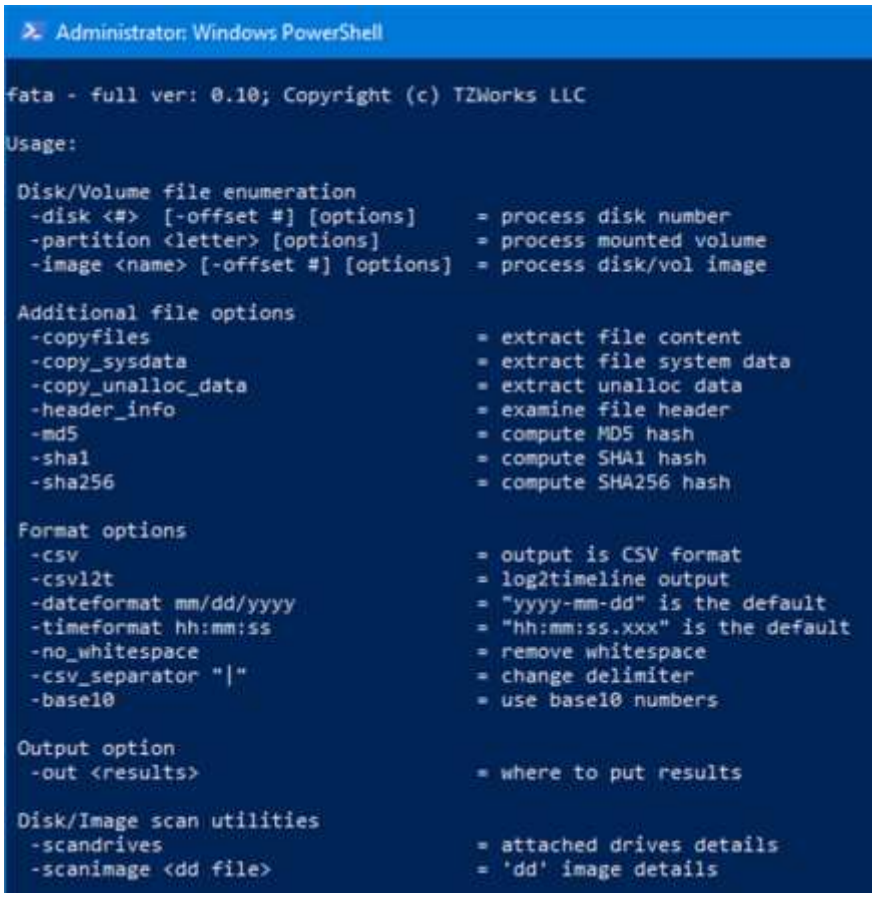
When considering the *FAT32* is typically the default filesystem for USB flash drives, coming up with a portable tool that can analyze the contents of the internal structures with or without mounting the device, as well as, not leaving a tool footprint on the system is useful in many forensic use-cases. Now that *exFAT* is commonly available and used for large storage devices, extending the ***fata*** architecture to handle that as well was a logic next step.

The ***fata*** tool parses all internal *FAT32* and *exFAT* filesystem data, and attempts to condense the reporting results in such a way as to make the output clear, while restricting the output to one line per record (file or folder). Header information is provided, if requested to assist in the identification of the file content without physically opening the file. Various hashing algorithms options are provided and can be annotated to the output, if requested. By default, both disk and volume offsets are provided where it makes sense, like for cluster runs, volume offset and directory entry locations. In this way, the information allows one to validate any of the results produced by this tool.

In addition to the filesystem internals, ***fata***, allows one to copy all the files that were enumerated; and/or all the system structures, such as the *Volume Boot Record*, *FAT* table(s), *Bitmap* table, *unallocated* clusters, etc. When found, deleted folders and files are shown and can be extracted, if requested.

2 How to Use *fata*

The screen shot below shows the available options for this tool.



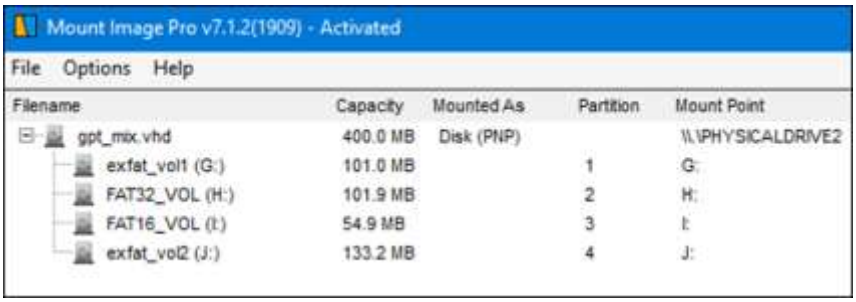
2.1 Disk/Volume file enumeration options

The basic options are the various file enumeration cases. One can enumerate the files via: (a) mounted partition letter, (b) disk number and volume offset relative to disk start, or (c) by a single file that contains a 'dd' image of another disk or volume. ***fata*** will return any file or folder it finds including deleted ones. Included with each file/folder is a complete set of metadata that was used by the filesystem internally to manage the file or folder. Each entry will be output on a separate line. The formats available are: CSV or Log2Timeline formats. Both are delimited data formats so they can be easily ported into an existing archival system.

2.1.1 Using Disk number that is attached to a system

If one has a disk that is to be analyzed that is attached to the computer system where the ***fata*** tool is run, then an option is available to analyze it as a raw disk. As a preparatory step, one first needs to find the physical disk number that is to be analyzed (see the section on *Scanning options*). Once that is done, one can invoke the ***-disk <number>*** command and the fata tool will enumerate the entire disk locating all the volumes, and if the filesystem is either FAT or exFAT, will report all the files on the respective volumes. If the disk has multiple volumes, one can target the specific volume by using the ***-offset <value>*** sub-option. This value inputted should be the volume offset relative to the start of the disk (eg. *physical sector 0*). For this last option, refer to the section of "*Scanning options*" to help locate the volume offsets.

As an example, we used *Mount Image Pro*, to mount a disk image that contained a variety of FAT partitions as shown below. The target image was mount point was physical disk 2.



Using the disk# parsing approach one can use ***fata*** to analyze all the *FAT32* and *exFAT* volumes on the mounted disk, using the following command.

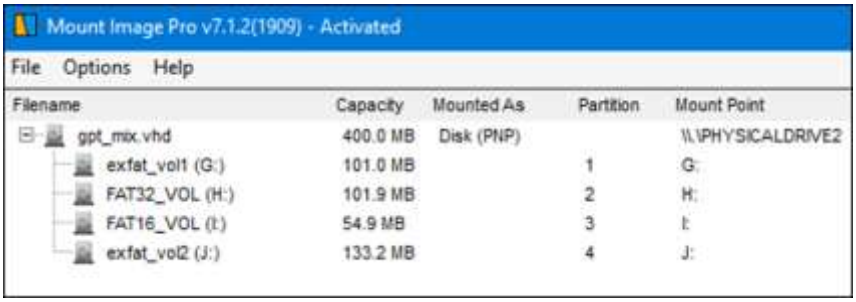
The default output is pipe-delimited; a portion of the data is shown below.

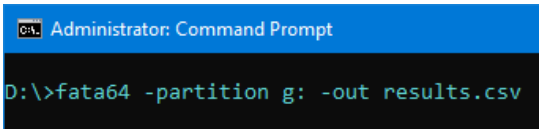
type	vol_type	modified_timestamp	access_timestamp	created_timestamp	utc_diff	name	path	size_valid	no
subdir	exfat	2022-12-01 19:05:26.220	2022-12-01 19:05:26.000	2022-12-01 19:05:26.220	utc-5.00	System Volume Information	[VBR_100000]	0x1000	["se
subdir	exfat	2022-12-01 19:07:51.730	2022-12-01 19:07:50.000	2022-12-01 19:07:51.730	utc-5.00	$RECYCLE.BIN	[VBR_100000]	0x1000	["se
subdir;deleted	exfat	2022-12-01 19:36:08.870	2022-12-01 19:36:08.000	2022-12-01 19:36:08.870	utc-5.00	tools	[VBR_100000]	0x1000	["se
subdir;deleted	exfat	2022-03-25 02:33:16.000	2022-12-01 19:36:26.000	2022-12-01 19:36:26.610	utc-5.00	logos	[VBR_100000]	0x1000	["se
subdir	exfat	2022-03-25 02:33:16.000	2022-12-01 19:40:18.000	2022-12-01 19:40:18.810	utc-5.00	logos	[VBR_100000]	0x1000	["se
subdir	exfat	2022-12-01 19:38:18.000	2022-12-01 19:40:18.000	2022-12-01 19:40:18.830	utc-5.00	tools	[VBR_100000]	0x1000	["se
file	exfat	2022-12-01 19:05:28.000	2022-12-01 19:05:28.000	2022-12-01 19:05:26.220	utc-5.00	WPSettings.dat	[VBR_100000]\System Volume Information	0x0c	["se
file	exfat	2022-12-01 19:05:28.000	2022-12-01 19:05:28.000	2022-12-01 19:05:26.570	utc-5.00	IndexerVolumeGuid	[VBR_100000]\System Volume Information	0x4c	["se
subdir;deleted	exfat	2022-12-01 19:07:39.580	2022-12-01 19:07:38.000	2022-12-01 19:07:39.580	utc-5.00	ClientRecoveryPasswordRotation	[VBR_100000]\System Volume Information	0x1000	["se
subdir;deleted	exfat	2022-12-01 19:07:39.580	2022-12-01 19:07:38.000	2022-12-01 19:07:39.580	utc-5.00	AadRecoveryPasswordDelete	[VBR_100000]\System Volume Information	0x1000	["se
subdir;deleted	exfat	2022-12-03 02:34:57.920	2022-12-03 02:34:56.000	2022-12-03 02:34:57.920	utc-5.00	ClientRecoveryPasswordRotation	[VBR_100000]\System Volume Information	0x1000	["se
subdir;deleted	exfat	2022-12-03 02:34:57.920	2022-12-03 02:34:56.000	2022-12-03 02:34:57.920	utc-5.00	AadRecoveryPasswordDelete	[VBR_100000]\System Volume Information	0x1000	["se
file	exfat	2022-12-01 19:07:52.000	2022-12-01 19:07:52.000	2022-12-01 19:07:51.730	utc-5.00	desktop.ini	[VBR_100000]\$RECYCLE.BIN	0x81	["se
file	exfat	2017-06-08 14:04:50.000	2022-12-01 19:40:18.000	2022-12-01 19:40:18.810	utc-5.00	image-0.jpg	[VBR_100000]\logos	0x836d	["se
file	exfat	2022-03-25 02:32:48.000	2022-12-01 19:40:18.000	2022-12-01 19:40:18.810	utc-5.00	image-1.jpg	[VBR_100000]\logos	0x11539	["se

2.1.2 Using a Mounted Partition letter

If one wanted to target a mounted volume, one can use the ***-partition <letter>***. This option is used for Windows.

Using the same example as above, the partition letters G, H, or J would be something ***fata*** could parse.





The output is the same as the previous disk # parse, however, the root folder is annotated to show the volume starting at a zero offset versus the disk offset of 0x100000.

type	vol_type	modified_timestamp	access_timestamp	created_timestamp	utc_diff	name	path		size_valid
subdir	exfat	2022-12-01 19:05:26.220	2022-12-01 19:05:26.000	2022-12-01 19:05:26.220	utc-5.00	System Volume Information	[VBR_000000]		0x1000
subdir	exfat	2022-12-01 19:07:51.730	2022-12-01 19:07:50.000	2022-12-01 19:07:51.730	utc-5.00	$RECYCLE.BIN	[VBR_000000]		0x1000
subdir;deleted	exfat	2022-12-01 19:36:08.870	2022-12-01 19:36:08.000	2022-12-01 19:36:08.870	utc-5.00	tools	[VBR_000000]		0x1000
subdir;deleted	exfat	2022-03-25 02:33:16.000	2022-12-01 19:36:26.000	2022-12-01 19:36:26.610	utc-5.00	logos	[VBR_000000]		0x1000
subdir	exfat	2022-03-25 02:33:16.000	2022-12-01 19:40:18.000	2022-12-01 19:40:18.810	utc-5.00	logos	[VBR_000000]		0x1000
subdir	exfat	2022-12-01 19:38:18.000	2022-12-01 19:40:18.000	2022-12-01 19:40:18.830	utc-5.00	tools	[VBR_000000]		0x1000
file	exfat	2022-12-01 19:05:28.000	2022-12-01 19:05:28.000	2022-12-01 19:05:26.220	utc-5.00	WPSettings.dat	[VBR_000000]	System Volume Information	0x0c
file	exfat	2022-12-01 19:05:28.000	2022-12-01 19:05:28.000	2022-12-01 19:05:26.570	utc-5.00	IndexerVolumeGuid	[VBR_000000]	System Volume Information	0x4c
subdir;deleted	exfat	2022-12-01 19:07:39.580	2022-12-01 19:07:38.000	2022-12-01 19:07:39.580	utc-5.00	ClientRecoveryPasswordRotation	[VBR_000000]	System Volume Information	0x1000
subdir;deleted	exfat	2022-12-01 19:07:39.580	2022-12-01 19:07:38.000	2022-12-01 19:07:39.580	utc-5.00	AadRecoveryPasswordDelete	[VBR_000000]	System Volume Information	0x1000
subdir;deleted	exfat	2022-12-03 02:34:57.920	2022-12-03 02:34:56.000	2022-12-03 02:34:57.920	utc-5.00	ClientRecoveryPasswordRotation	[VBR_000000]	System Volume Information	0x1000
subdir;deleted	exfat	2022-12-03 02:34:57.920	2022-12-03 02:34:56.000	2022-12-03 02:34:57.920	utc-5.00	AadRecoveryPasswordDelete	[VBR_000000]	System Volume Information	0x1000
file	exfat	2022-12-01 19:07:52.000	2022-12-01 19:07:52.000	2022-12-01 19:07:51.730	utc-5.00	desktop.ini	[VBR_000000]	$RECYCLE.BIN	0x81
file	exfat	2017-06-08 14:04:50.000	2022-12-01 19:40:18.000	2022-12-01 19:40:18.810	utc-5.00	image-0.jpg	[VBR_000000]	logos	0x836d
file	exfat	2022-03-25 02:32:48.000	2022-12-01 19:40:18.000	2022-12-01 19:40:18.810	utc-5.00	image-1.jpg	[VBR_000000]	logos	0x11539

2.1.3 Using an Offline Image of the disk or volume

The last option, can be used for Windows, Linux or macOS and will target a disk or volume in the form of a file (eg. *image*). This option assumes the image is not compressed or encrypted; the image file needs to be a 'dd' copy of a file. To process this image, one can use the ***-image <file>*** option. If the image file has multiple volumes, one can target the specific volume by using the ***-offset <value>*** sub-option. This value inputted should be the volume offset relative to the start of the image file. For this last option, refer to the section of "*Scanning options*" to help locate the volume offsets.

The results are the same as shown in the section "*Using Disk number that is attached to a system*".

2.2 Extraction of Data

The fata tool has a few options to extract more than just file path and its metadata. One can also copy the file contents, system filesystem structures that mange the filesystem and unallocated clusters. One can select these options independently or in any combination thereof.

2.2.1 Copying file contents

If the option ***-copyfiles*** is invoked, the tool will try to copy any file will an extracted cluster run. This includes both valid and deleted files. They are archived in the ***export/[VBR_<offset>]*** subfolder. See section on "Where files are copied to" for an example on using this option.

2.2.2 Copying system data

If the option ***-copy_sysdata*** is invoked, the tool will try to copy system structures and store the data in separate files. System structures include: Volume Boot Record (VBR), File Access Tables, Reserved sectors, Bitmap table (for exFAT), etc. All the files created are binary data in that they reflect the actual bytes from the data structures, with the exception of the *offset_map.txt* file. See section on "Where system data is copied to" for an example on using this option.

2.2.3 Copying unallocated clusters

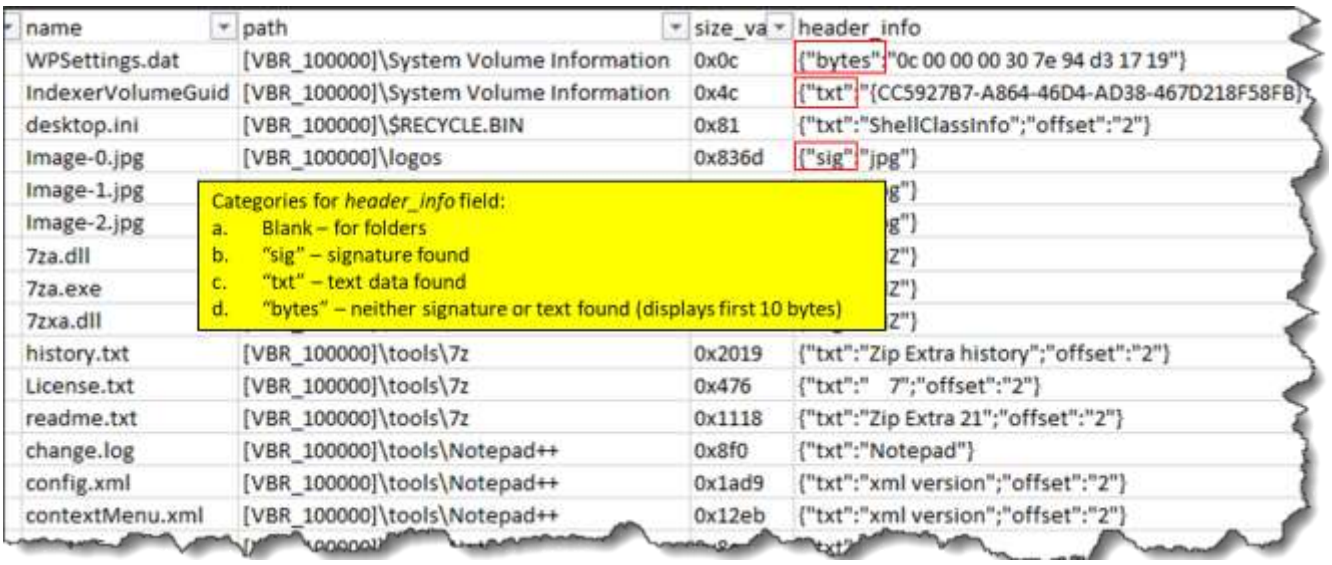
If the option ***-copy_unalloc_data*** is invoked, the tool will try to copy all the unallocated clusters and store the data into one file. The reason why this is not included in the ***-copy_sysdata*** option, is the resulting file that is generated can be very large depending on the size of the disk (or disk/volume image) and the number of unallocated clusters it has. With multi-terabyte drives as typical and exFAT able to make use of all the available space, one needs to plan accordingly when using ***-copy_unalloc_data*** option, since depending on how much space is unallocated, this option would create a very large file. For this reason, this functionality is split off from the ***-copy_sysdata*** option. See section on "Mapping of unallocated space system" for an example on using this option.

2.2.4 File content header information

During the parsing of the files, ***fata*** can take a look at the first sector of the raw file data to give the analyst a view of the data. If a signature is present, then the type should match the extension of the file. Conversely, if the type is generic (like a text document), then a portion of the string is displayed.

One uses the ***-header_info*** option to invoke this behavior. The output is rendered in a quasi-JSON format in the '*header_info*' column of the CSV output file. Below is an example of using this option and how the output is rendered.

The output uses prefixes with each of the outputs If magic file signatures are found, they are prefixed with "***sig***". If no signatures are found, ***fata*** then looks for any printable text; these are prefixed with "***txt***". Finally, if no signature or text is found, then the first 10 bytes of the file header are displayed; these are prefixed with "***bytes***". One should note, the tool does not make an attempt to categorize all magic file signatures, just some of the more common ones.



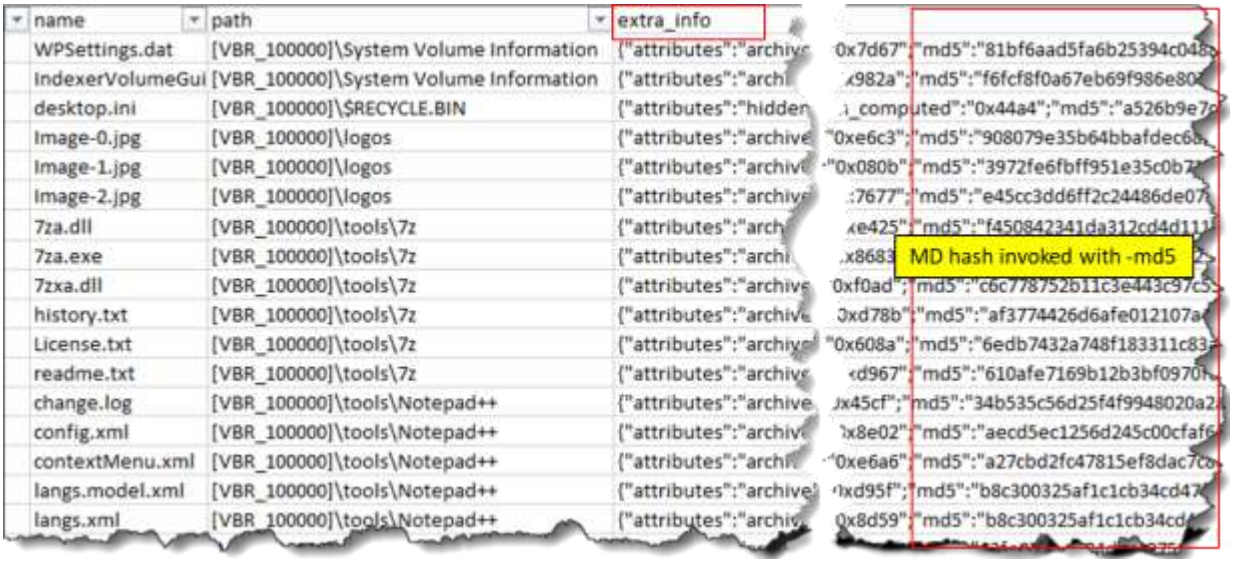
name	path	size_va	header_info
WPSettings.dat	[VBR_100000]\System Volume Information	0x0c	{"bytes":"0c 00 00 00 30 7e 94 d3 17 19"}
IndexerVolumeGuid	[VBR_100000]\System Volume Information	0x4c	{"txt":"{CC5927B7-A864-46D4-AD38-467D218F58F8}"
desktop.ini	[VBR_100000]\$RECYCLE.BIN	0x81	{"txt":"ShellClassInfo";"offset":"2"}
Image-0.jpg	[VBR_100000]\logos	0x836d	{"sig":"jpg"}
Image-1.jpg			g"}
Image-2.jpg			g"}
7za.dll			Z"}
7za.exe			Z"}
7zxa.dll			Z"}
history.txt	[VBR_100000]\tools\7z	0x2019	{"txt":"Zip Extra history";"offset":"2"}
License.txt	[VBR_100000]\tools\7z	0x476	{"txt":" 7";"offset":"2"}
readme.txt	[VBR_100000]\tools\7z	0x1118	{"txt":"Zip Extra 21";"offset":"2"}
change.log	[VBR_100000]\tools\Notepad++	0x8f0	{"txt":"Notepad"}
config.xml	[VBR_100000]\tools\Notepad++	0x1ad9	{"txt":"xml version";"offset":"2"}
contextMenu.xml	[VBR_100000]\tools\Notepad++	0x12eb	{"txt":"xml version";"offset":"2"}

Categories for *header_info* field:
a. Blank – for folders
b. "sig" – signature found
c. "txt" – text data found
d. "bytes" – neither signature or text found (displays first 10 bytes)

2.3 Hashing

There are three hashing options available to the user. One can select one or more hashing functions, from MD5 (-md5), SHA1 (-sha1) or SHA256 (-sha256). The hashing only considers valid file data and

ignores slack data. The results of the requested hashes are displayed in the '*extra_info*' column of the CSV output file.



2.4 Mapping the results

The results will be sent to whatever is specified in the ***-out <results>*** option. For example, if the option is specified as: ***-out 2022-11-30/results.csv***, the tool will create a relative folder [*2022-11-30*], if it doesn't exist, and the results of the file enumeration will be stored in *results.csv*.

Any other data that is requested either via (***-copyfiles***, ***-copy_systemdata***, or ***-copy_unalloc_data***) a secondary *export* subfolder will be created (eg. 2*022-11-30/export*) and depending on the options selected one or more tertiary folder(s) will be created.

2.4.1 FAT and exFAT internals and where it they are located in the output

For the default parsing, where only the files and folders are enumerated, a CSV type file is created. The CSV will have some fixed data fields and some variable data fields. The CSV fields and where they map to are shown below:

Field	Field name	Data type
Type	*type*	Type of entry, whether it be a: file, subdir, deleted, or volume label
Volume type	*vol_type*	fat32, or exFAT
Modified time	*modified_timestamp*	Default date/time format is: yyyy-mm-dd hh:mm:ss.xxx. Could be either local or UTC
Access time	*access_timestamp*	Default date/time format is: yyyy-mm-dd hh:mm:ss.xxx. Could be either local or UTC
Created time	*created_timestamp*	Default date/time format is: yyyy-mm-dd hh:mm:ss.xxx. Could be either local or UTC
UTC time difference	*utc_diff*	*Local* (fat32), *UTC+<offset value>* (exFAT)
File or Folder name	*name*	*Name* of the file or folder without the path
Path of file or folder	*path*	*Path* of the file or folder without the name
Size of file without slack	*size_valid*	*Size* of the file that is used
File signature based on starting bytes	*header_info*	Only populated if the ***-header_info*** option is selected. Looks at the bytes in the first sector of the file. If it can be recognized, the type will be displayed, if not, the first text found will be displayed. This option is experimental in that it can only recognize basic file formats.
Notes in a quasi-JSON format	*notes*	Data such as *sector size, cluster size, volume offset/size* and *volume serial number* will be displayed
Internals of the file/folder in a quasi-JSON format	*extra_info*	*Cluster run, attributes, DOS3.8 name* (if applicable), and *data allocated* will be displayed. If hashes are requested, via ***-md5***, ***-sha1***, or ***-sha256***, they will also be shown here.

2.4.1.1 Notes Field

This field is a quasi-JSON paring of the *{"name1":"value1"; "name2":"value2"; etc}*. The data is defined as follows.

Name	Meaning	Other info
sector_size	Size of the sector in bytes	
cluster_size	Size of the cluster in bytes	
vol_offset	Volume offset relative to the start of the disk in terms of bytes	If using a partition type parse, this value will be 0.

vol_size	Volume size in terms of bytes	
vol_serial_number	Volume serial number.	For those volumes with a serial number of 8 bytes, only the least significant 4 bytes are shown

2.4.1.2 Extra_info Field

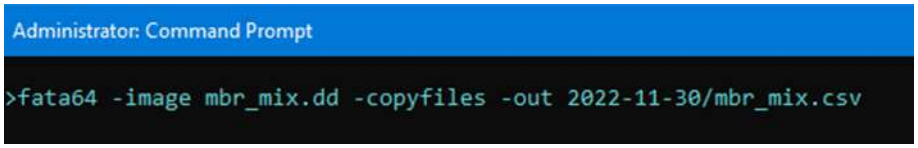
This field is a quasi-JSON paring of the *{"name1":"value1"; "name2":"value2"; etc}*. The data is defined as follows.

Name	Meaning	Other info
attributes	Attributes flag in the Directory Entry	Read Only, Hidden, System, Volume label, Folder, archive, etc
checksum_embeded	Checksum contained in the exFAT File Directory Entry	Only applies to exFAT
checksum_computed	Checksum recomputed based on the Directory Entry data	Only applies to exFAT and is used to verify the directory entry collection is valid
cluster_run	All clusters used to store the file/folder content	For folders, this is the cluster run for all the children directory entries. For files, this is the cluster run for data content. Cluster run notation is: *<disk offset of starting cluster>-<LCN of starting cluster>:<number of clusters>*. This is done for each fragment.
data_size_alloc	Size of the allocated clusters, translated to number of bytes	
dir_entry_vol_offset	Location of the start of the Directory Entry collection for this record.	Location is relative to the start of the volume.
md5	MD5 hash of the data content	Only applies to files and not folders and only include valid data (no slack data).
sha1	SHA1 hash of the data content	Only applies to files and not folders and only include valid data (no slack data).
sha256	SHA256 hash of the data content	Only applies to files and not folders and only include valid data (no slack data).
header_info	Information about the initial data in the file	Only applies to ***-csvl2t*** format, since this already is a separate csv field is in *-csv*. Only applies

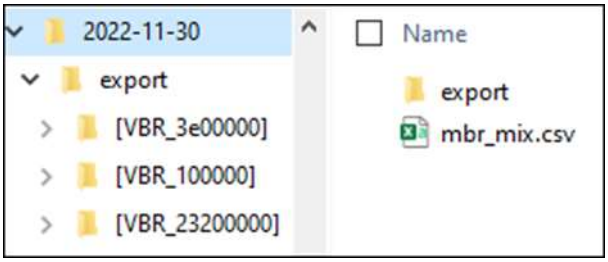
		to files with data. Requires the ***-header_info*** command line option.

2.4.2 Where files are copied to

If the option ***-copyfiles*** in invoked, the tool will try to copy any file with a valid cluster run; this includes both valid and deleted files. Extracted files are archived in the ***export/[VBR_<offset>]*** subfolder. Below is an example of running ***fata*** targeting all the volumes in a disk image using just the ***-copyfiles*** option and which folders are generated.

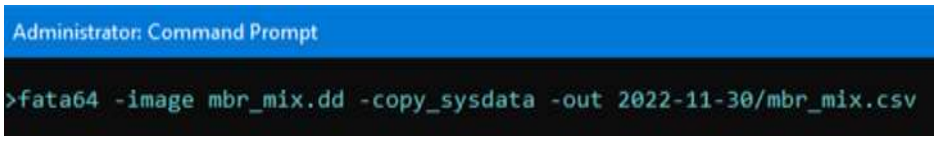


In this case, we will use the relative subdirectory [*2022-11-30*] to store the results. By including the subdirectory in the ***-out*** parameter, the tool will create the subdirectory and the appropriate subdirectories that are needed. From the example, the tool creates the export sub folder as well as sub folders for each volume found that is either FAT32 or exFAT. These sub folders are annotated with the image offset of the respective volumes. Inside these sub folders, the files/folders are copied.

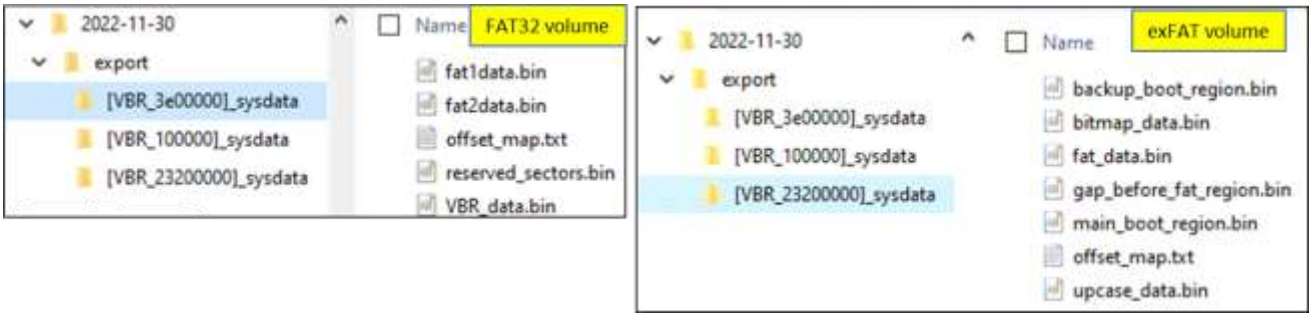


2.4.3 Where system data is copied to

If the option ***-copy_sysdata*** in invoked, the tool will try to copy system structures and store the data in separate files. System structures include: Volume Boot Record (VBR), File Access Tables, Reserved sectors, Bitmap table (for exFAT), etc. All the files created are binary data in that they reflect the actual bytes from the data structures, with the exception of the *offset_map.txt* file. The *offset_map.txt* file identifies the actual disk offsets and where the binary data comes from. It also shows how it is mapped to the file offset of the archived data. These collections of files are created for each volume parsed.



For this example by adding the ***-copy_sysdata*** from the previous example, the tool generates the extra *[VBR_<offset>]_sysdata* folder(s) for each volume parsed. The screenshot below shows the types of system files generated depending on whether the filesystem is FAT32 or exFAT.



The contents of the ***offset_map.txt*** file are shown for the FAT32 filesystem starting at offset 0x3e00000.

offset_map.txt ×

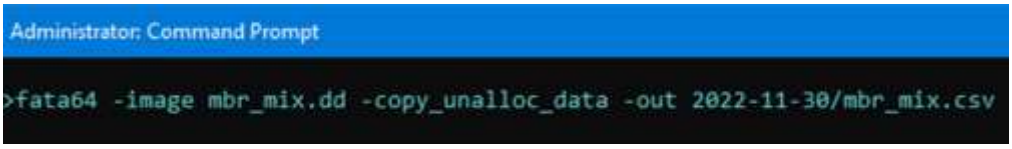
```
volume stats
volume (byte offset)              : 0x3e00000
volume length (bytes)             : 0x1f400000
sector size (bytes)               : 0x000200
cluster size (bytes)              : 0x001000
fat1 (byte offset)                : 0x307c00
fat2 (byte offset)                : 0x383e00
fat size (bytes)                  : 0x07c200
cluster heap (byte offset)        : 0x3fe000
cluster count                     : 0x01f002
root dir (byte offset)            : 0x000006
volume serial#                    : 0x82348628

files extracted (system offsets)


disk offset |volume offset |file offset |size bytes |          comment |                name
  0x3e00000 |         0x00 |       0x00 |     0x200 |                  | 2022-11-30-1\export\[VBR_3e00000]_sysdata\VBR_data.bin
  0x4107c00 |     0x307c00 |       0x00 |    0x7c200 |                  | 2022-11-30-1\export\[VBR_3e00000]_sysdata\fat1data.bin
  0x4183e00 |     0x383e00 |       0x00 |    0x7c200 |                  | 2022-11-30-1\export\[VBR_3e00000]_sysdata\fat2data.bin
  0x7c00000 |    0x3e00000 |       0x00 |   0x307c00 |                  | 2022-11-30-1\export\[VBR_3e00000]_sysdata\reserved_sectors.bin
```

2.4.4 Mapping of unallocated space

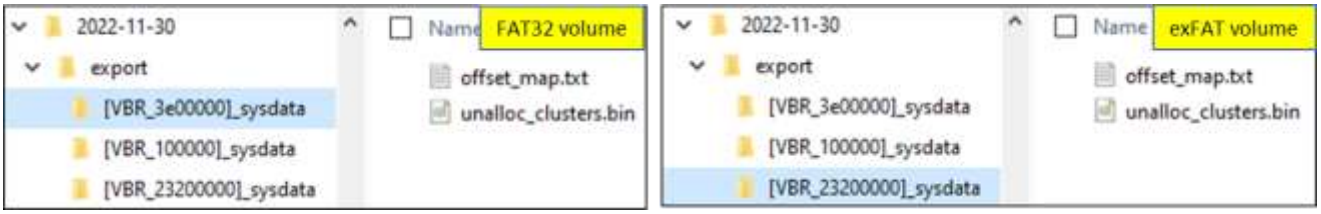
If the option ***-copy_unalloc_data*** is invoked, the tool will try to copy all the unallocated clusters and store the data into one file. The reason why this is not included in the ***-copy_sysdata*** option, is the resulting file that is generated can be very large depending on the size of the disk (or disk/volume image) and the number of unallocated clusters it has. With multi-terabyte drives as typical and exFAT able to make use of all the available space, one needs to plan accordingly when using ***-copy_unalloc_data*** option, since depending on how much space is unallocated, this option would create a very large file. For this reason, this functionality is split off from the ***-copy_sysdata*** option.

When this option is invoked, it will create a *cluster run* of all the unallocated clusters, which will then be placed in the ***offset_map.txt*** file. In this way, one can later examine any unallocated cluster from the extracted data and map it back into the actual disk (or image) physical location.

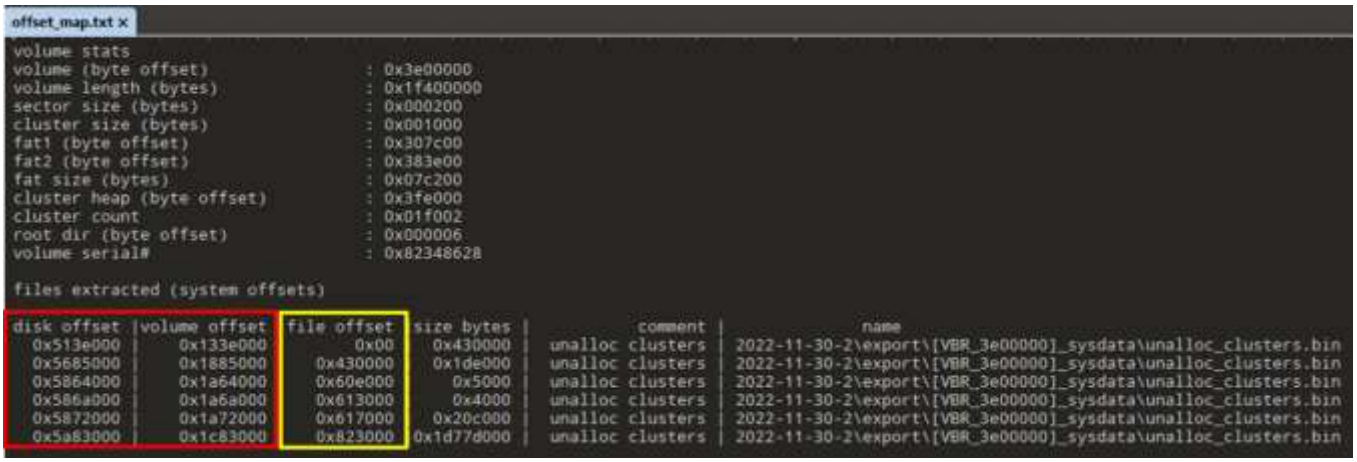
To see how this is represented, below is an example running ***fata*** with the same image that was done in the previous section. The option used, however, will be to only extract unallocated clusters.



Based on the above command the following folders and files were created



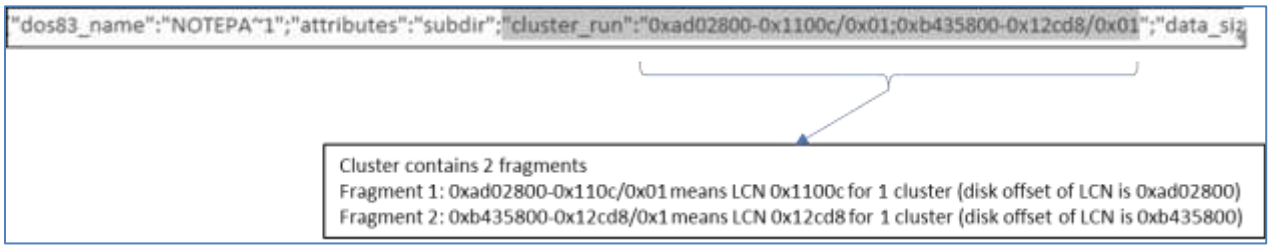
The *offset_map.txt* output is shown below and only the unallocated clusters are shown. The highlighted *red* section shows the absolute disk (or image file) offset along with the relative volume offset for this entry. The *yellow* section shows where this entry maps to relative to the binary file (unalloc_clusters.bin) that was created. In this way, one can examine the binary file and go back to the original image and located any cluster fragments, or just verify the results.



2.5 Cluster Runs and how to read them

fata will output the cluster runs of the data that it parses. For folder type data, *fata* will identify the *cluster run* of the *Directory Entries* for its first level children that includes both files and folders. For file type data, the *cluster run* reported represents the actual content of the file data.

For this example, the application *Notepad++* subfolder contains 23 sets of directory entry collections for its children. It happens that this requires 2 clusters to store all the directory entry collections and they are not contiguous, which means each cluster represents a fragment. This is how ***fata*** displays the cluster run.



Each fragment is broken up into 3 fields. The disk offset of the starting cluster (or the logical cluster number – LCN), the LCN and the extent (the number of clusters that are contiguous). The disk offset is provided so one can go to the starting cluster number quickly to examine the raw data.

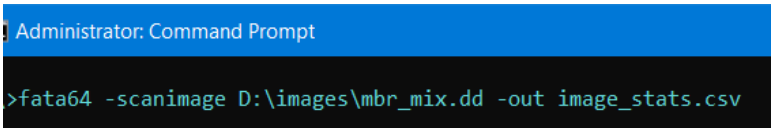
3 Scanning options

If targeting a disk that is mounted or if processing a 'dd' image, one can find where the volumes are located as well as the filesystem of each volume by using one of the two commands: ***-scandrives*** or ***-scanimage***. These scanning options are designed to assist the user locate filesystems and their respective offsets, so as to target a specific volume instead of processing the entire disk/image.

The first command, ***-scandrives*** is only for mounted disks on the same system that the ***fata*** tool will be running on. The second command, ***-scanimage*** is only for an unmount disk or volume image. The fata tool only works with images that are not compressed or encrypted. Examples of both options are shown below along with their respective outputs.

3.1 Scan 'dd' image file

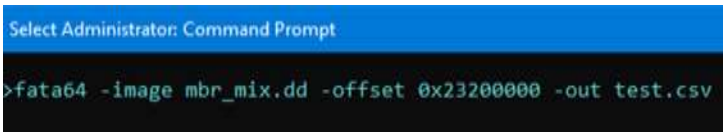
To scan the volumes in an 'dd' type image, one uses the ***-scanimage***. This is only works with images that not compressed or encrypted. As an example, the sample image is called 'mbr_mix.dd'. The image file is a copy of a disk that has a MBR (master boot record), two FAT32 partitions, an exFAT partition, and an NTFS BitLocker partition.



After enumeration of the volumes in the image, the following is outputted. Highlighted are the offset of the volumes relative the start of the file and their respective filesystems.

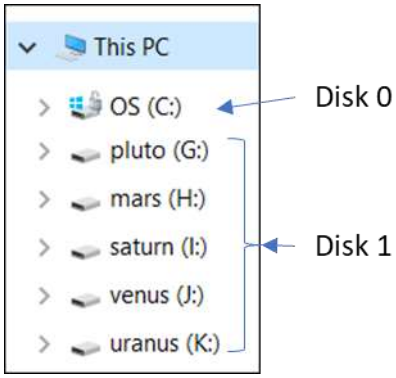
drive_typ	disk_sig	type	start_offset	end_offset	num_bytes	vol sig	format	description
image	e0ae-5a54	MBR	0x00	0x01ff	0x0200		0x00	mbr; sector_size: 0x0200; cluster_size: 0xf800
image	e0ae-5a54	MBR	0x0200	0x0fffff	0x0ffe00		0x00	unalloc
image	e0ae-5a54	MBR	0x100000	0x03dfffff	0x03d00000	6e7f-824c	0x0c	fat32; sector_size: 0x0200; cluster_size: 0x0200
image	e0ae-5a54	MBR	0x03e00000	0x231fffff	0x1f400000	8234-862E	0x0c	fat32; sector_size: 0x0200; cluster_size: 0x1000
image	e0ae-5a54	MBR	0x23200000	0xa65fffff	0x83400000	289a-675:	0x07	exfat; sector_size: 0x0200; cluster_size: 0x8000
image	e0ae-5a54	MBR	0xa6600000	0xffefffff	0x59900000		0x0f	exten
image	e0ae-5a54	MBR	0xa6700000	0xc5afffff	0x1f400000	414e-204f	0x07	bitlocker container; sector_size: 0x0200; cluster_size: 0x1000
image	e0ae-5a54	MBR	0xc5b00000	0xffffffff	0x3a500000		0x00	unalloc

One can then use this data to process the desired volume, via the ***-image <name>*** and ***-offset <value>*** options. Below is an example of processing the exFAT volume at disk offset 0x23200000.

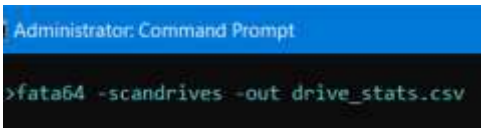


3.2 Scan attached drives

To scan all the attached drives on a system, one can use the ***-scandrives*** option. As an example, the two attached disks have the following explorer profile. Disk 0 has the system volume which a NTFS *Bitlocker* volume. Disk 1 has 5 volumes that have an exFAT filesystem with various cluster sizes.

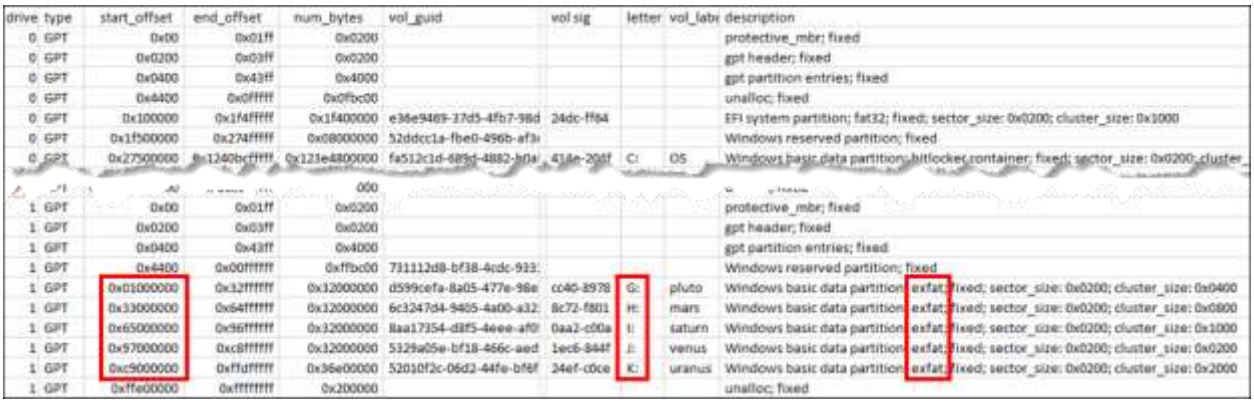


With an administrator command shell, one can enumerate the disks and which volumes they include, via:



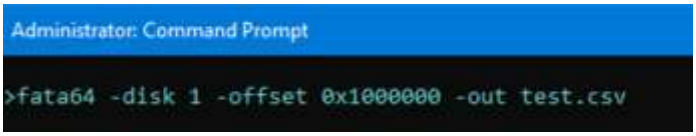
If this was done in Windows, the disk identification will be an integer (eg. 0, 1, etc). If this was done on macOS or Linux, one will get a device name in the form of a path (for macOS /dev/disk0, /dev/disk1, etc, for Linux /dev/sda, /dev/sdb, etc).

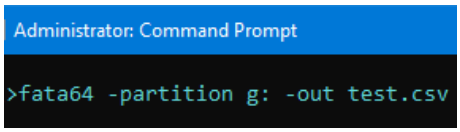
The results file is a pipe delimited CSV file. Highlighted are the exFAT volumes starting offset relative to the physical disk and the volume letter that was used for mounting purposes. Given this data, one can either analyze a specific volume either by the ***-disk <#>*** (if Windows) and ***-offset <value>*** option or via the ***-partition <letter>*** option. The ***-partition*** option is only for Windows.



drive	type	start_offset	end_offset	num_bytes	vol_guid	vol sig	letter	vol_labe	description
0	GPT	0x00	0x01ff	0x0200					protective_mbr; fixed
0	GPT	0x0200	0x03ff	0x0200					gpt header; fixed
0	GPT	0x0400	0x43ff	0x4000					gpt partition entries; fixed
0	GPT	0x4400	0x0fffff	0x0fbc00					unalloc; fixed
0	GPT	0x100000	0x1f4fffff	0x1f400000	e36e9469-17d5-4fb7-98d	24dc-ff84			EFI system partition; fat32; fixed; sector_size: 0x0200; cluster_size: 0x1000
0	GPT	0x1f500000	0x274fffff	0x08000000	52ddcc1a-fbe0-496b-af3				Windows reserved partition; fixed
0	GPT	0x27500000	0x1340bcffff	0x123e4800000	fa512c1d-689d-4882-b0a	418e-206f	C:	OS	Windows basic data partition; bitlocker container; fixed; sector_size: 0x0200; cluster
				000					
1	GPT	0x00	0x01ff	0x0200					protective_mbr; fixed
1	GPT	0x0200	0x03ff	0x0200					gpt header; fixed
1	GPT	0x0400	0x43ff	0x4000					gpt partition entries; fixed
1	GPT	0x4400	0x00ffffff	0xffbc00	731112d8-bf38-4cdc-933				Windows reserved partition; fixed
1	GPT	0x01000000	0x12ffffff	0x12000000	d599cefa-8a05-477e-98e	cc40-8978	G:	pluto	Windows basic data partition; exfat; fixed; sector_size: 0x0200; cluster_size: 0x0400
1	GPT	0x13000000	0x64ffffff	0x12000000	6c3247d4-9405-4a00-a32	8c72-f801	H:	mars	Windows basic data partition; exfat; fixed; sector_size: 0x0200; cluster_size: 0x0800
1	GPT	0x65000000	0x96ffffff	0x12000000	8aa17354-d8f5-4eee-af0	0aa2-c00a	I:	saturn	Windows basic data partition; exfat; fixed; sector_size: 0x0200; cluster_size: 0x1000
1	GPT	0x97000000	0xc8ffffff	0x12000000	5329a05e-bf18-466c-aed	1ec6-844f	J:	venus	Windows basic data partition; exfat; fixed; sector_size: 0x0200; cluster_size: 0x0200
1	GPT	0xc9000000	0xffdfffff	0x36e00000	52010f2c-06d2-44fe-bf8f	34ef-c0ce	K:	uranus	Windows basic data partition; exfat; fixed; sector_size: 0x0200; cluster_size: 0x2000
1	GPT	0xffe00000	0xffffffff	0x200000					unalloc; fixed

For example, in Windows, to target the '*pluto*' volume, one can use either the ***-disk*** or ***-partition*** options. Both are shown below. Note, the ***-disk*** option requires both the disk number and the offset of the volume, whereas the ***-partition*** option only requires the mounted volume letter.





4 Available Options

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>-image</td><td>Process the volumes present in the image file. The syntax is: -image <filename> [-offset <volume offset value>]. If no offset is provided, then all the volumes in the image are processed. If the offset is provided, only that volume is processed. This option can be used for Windows, Linux or macOS</td></tr>
<tr><td>-disk</td><td>Process the volumes present in the physical disk number. The syntax is:

<table><tr><td>-disk <number> [-offset <volume offset value>]</td><td>Windows</td></tr><tr><td>-disk /dev/disk<#> [-offset <volume offset value>]</td><td>macOS</td></tr><tr><td>-disk /dev/sda (or sdb…) [-offset <volume offset value>]</td><td>Linux</td></tr></table>If no offset is provided, then all the volumes in the disk are processed. If the offset of the volume is provided, only that volume is processed.</td></tr>
<tr><td>-partition</td><td>Process the volume that equates to the partition letter passed in. The syntax is: -partition <letter>. Note. This is only a Windows option.</td></tr>
<tr><td>-csv</td><td>Outputs the data fields delimited by commas. Since filenames can have commas, to ensure the fields are uniquely separated, any commas in the filenames get converted to spaces.</td></tr>
<tr><td>-csvl2t</td><td>Outputs the data fields in accordance with the log2timeline format.</td></tr>
<tr><td>-no_whitespace</td><td>Used in conjunction with -csv option to remove any whitespace between the field value and the CSV separator.</td></tr>
<tr><td>-csv_separator</td><td>Used in conjunction with the -csv option to change the CSV separator from the default comma to something else. Syntax is -csv_separator "|" to change the CSV separator to the pipe character. To use the tab as a separator, one can use the -csv_separator "tab" OR -csv_separator "\t" options.</td></tr>
<tr><td>-dateformat</td><td>Output the date using the specified format. Default behavior is -dateformat "yyyy-mm-dd". Using this option allows one to adjust the format to mm/dd/yy, dd/mm/yy, etc. The restriction with this option is the forward slash (/) or dash (-) symbol needs to separate month, day and year and the month is in digit (1-12) form versus abbreviated name form.</td></tr>
<tr><td>-timeformat</td><td>Output the time using the specified format. Default behavior is -timeformat "hh:mm:ss.xxx" One can adjust the format to microseconds, via "hh:mm:ss.xxxxxx" or nanoseconds, via "hh:mm:ss.xxxxxxxxx", or no fractional seconds, via "hh:mm:ss". The restrictions with this option is a colon (:) symbol needs to separate hours, minutes and seconds, a period (.)</td></tr>
</table>

	symbol needs to separate the seconds and fractional seconds, and the repeating symbol 'x' is used to represent number of fractional seconds.
-out	Specifies the file to send the output to. Syntax is: *-out <output file>*.
-quiet	Show no progress during the parsing operation.
-base10	Output values in base10. Default is base16.
-utf8_bom	All output is in Unicode UTF-8 format. If desired, one can prefix an UTF-8 *byte order mark* to the output using this option.
-copyfiles	Option that tells ***fata*** to extract file contents, if possible. The data for these folders/files are put in the *export* subfolder that has the parent of the base output folder.
-copy_sysdata	Option that tells ***fata*** to extract certain filesystem data structures. This includes the: VBR (volume boot record), FAT (file allocation table), Bitmap (if available for that filesystem), etc. The output for these extra files are put in a [*export/[VBR_xxxx]_sysdata*] subfolder where the root is the base output folder. Finally, for all the sections extracted, a summary file (*offset_map.txt*) is created in the same subdirectory.
-copy_unalloc_data	Option that tells ***fata*** to extract all the *unallocated* data into one file. Since this resulting file will have fragmented clusters of unallocated data consequently ordered, one can reconstruct the which clusters are associated by volume offset, but referring to the *offset_map.txt* file where each cluster is mapped. The output for the unallocated clusters is a file [***unalloc_clusters.bin***], in the subfolder [*export/[VBR_xxxx]_sysdata*] where the root is the base output folder. In addition to the sections extracted, the summary of all the extracted files is presented in the *offset_map.txt* file.
-header_info	Option to examine the first sector of the file and if a signature is found, it is displayed in the output.
-md5	Computes the MD hash of the valid file contents (does not include the slack in the file).
-sha1	Computes the SHA1 hash of the valid file contents (does not include the slack in the file).
-sha256	Computes the SHA256 hash of the valid file contents (does not include the slack in the file).
-scandrives	Details about the volumes of the attached drives on the system where fata is run.

-scanimage	Details about the volumes in an image file. The syntax is: *-scanimage <dd file>*. The 'dd' file needs to be uncompressed image of the disk or volume and cannot be encrypted.

5 Internals of the FAT32 Filesystem

The FAT file system has four basic regions

#	Region	Size in Sectors	Contents
1	Reserved Sectors	(num of reserved sectors in VBR)	Volume Boot Sector; file system info sector (FAT32 only); and other optional reserved sectors
2	FAT Region	(num of FATs) * (sectors per FAT)	File Allocation Table #1 File Allocation Table #2 (optional)
3	Root Directory Region	(num of root entries * 0x20) / (bytes per sector)	Root Directory (FAT12 and FAT16 only). Eliminated as a separate region in FAT32 and became part of the data region
4	Data Region	(num of clusters) * (sectors per cluster)	Data Region (for files and directories).. (to the end of partition on disk)

The Volume Boot Record (VBR) is always located in logical sector 0 (LS 0) of the logical volume. The VBR is created during the high-level formatting process of the volume and contains information about the volume. The VBR of a primary partition will contain boot code needed to continue the boot process if that partition is set as the active primary partition. The VBR is different than the Master Boot Record (MBR). The MBR is located in physical sector 0 (PS 0) of the physical disk and contains the Master Boot Code and Master Partition Table.

Note: that removable media does not always have an MBR; smaller media commonly have only a VBR. In these cases, PS 0 of the physical disk is the same as the LS 0 of the logical volume. (PS 0 relates to the disk and LS 0 relates to the volume).

Microsoft refers to the VBR for volumes formatted with a FAT file system as the Boot Sector with a BPB (BIOS Parameter Block).

5.1 Volume Parameter Block

Offset	Size	Name	Description

0x00	3	JMP Instruction	The Jump instruction continued from the MBR
0x03	8	OEMID	String of characters that can indicate the OS used during format MSWIN4.0 = Windows 95 per-OSR2 MSWIN4.1 = Windows 95 OSR2 through Windows 98 MS-DOS5.0 = indicates Windows 2k and newer
0x0B	2	Bytes per sector	Start of the BPB; can contain values of 512, 1024, 2048, 4096
0x0D	1	Sectors per cluster	Represents the number of sectors assigned to a single allocation unit. If the value is positive, then the value is taken as is. If the value is negative (high bit set), then the twos-complement is taken and that resulting value is used as a power of 2 (eg. 1 << (-1 * sectors_per_cluster))
0x0E	2	Number of Reserved Sectors	FAT12, FAT16 should be a value of 1. FAT32 can be a value of 32 and higher.
0x10	1	Number of FATs	Should always be a value of 2
0x11	2	Num of Root Directory Entries	Number of 0x20 byte directory entries in the Root Directory region. For **FAT32 is value is 0**, since the root directory entries in FAT32 are in the data area and are only restricted by the size of the data area.
0x13	2	Total Sectors	Count of sectors occupied by one FAT for FAT12 and FAT16 volumes. On a **FAT32 volume is value is 0**, and is represented at offset 0x24
0x15	1	Media Descriptor	Legal values include: 0xF0, 0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE and 0xFF. The two **most common values are 0xF8 for fixed media and 0xF0 for removable media**
0x16	2	Sectors per FAT	Count of sectors occupied by one FAT for FAT12 and FAT16 volumes. **On a FAT32 volume this value is 0**, and the value is represented at offset 0x24.
0x18	2	Sectors per track	This field is only relevant for media that have disk geometry with CHS (Cylinder, Head, Sector).
0x1A	2	Number of heads	This field is only relevant for media that have disk geometry with CHS (Cylinder, Head, Sector).
0x1C	4	Hidden Sectors	Count of hidden sectors preceding the partition containing the FAT volume. Part of the BPB.

0x20	4	Total Sectors	Count of total sectors for the volume, including system areas. For volumes where the total sector count exceeds the value that can be stored in 16 bits.

This shows the remaining byte structure for a FAT12/16 VBR

Offset	Size	Name	Description
0x24	1	BIOS drive number	Supports MS-DOS bootstrap and is set to the interrupt 13 drive number of the media. 0x00 for floppy disks; 0x80 for hard disks. This field is OS specific
0x25	1	Reserved	Should be set to 0
0x26	1	Extended boot signature	If set to (0x29), then the next 3 fields are present
0x27	4	Volume Serial Number	32-bit value usually generated from the date/time. Used for tracking removable media. This value can often be found in the LNK files (in Windows).
0x2B	11	Volume label	11-byte volume label recorded in the Root directory. If no volume label is provided, then "NO NAME" is the default value.
0x36	8	File System Type	Although this field generally represents the file system formatted on the volume, it is an informational field only and is not used by FAT drivers to determine the FAT system type.

This shows the remaining byte structure for a FAT32 VBR

Offset	Size	Name	Description
0x24	4	Sectors per FAT	Number of sectors occupied by one File Allocation Table (for FAT32 only)
0x28	2	Extended Flags	Bits 0-3 zero-based number of active FATs. Only valid if mirroring is disabled. Bits 4-6 are reserved. Bit 7 - value of 0 means the FAT is mirrored; value of 1 means only 1 FAT is active and is referenced. In the first 3 bytes, bits 8-15 are reserved
0x2A	2	FAT Version	The high byte is the major revision number and the low bit is the minor revision number.
0x2C	4	Root Directory Cluster	Points to the starting cluster for the Root Directory. This is usually cluster 2, but is not required to be cluster 2.
0x30	2	File system info sector	The sector number of the FSINFO structure in the reserved area of the FAT32 volume. Usually set to 1.

0x32	2	Backup boot sector	If not 0, the value represents the sector number of the copy of the boot sector in the reserved area. Generally, set to 6
0x34	12	Reserved	Should be zero
0x40	1	BIOS drive number	Supports MS-DOS bootstrap and is set to the interrupt 13 drive number of the media. 0x00 = floppy disks; 0x80 = 0 hard disks.
0x41	1	Reserved / Error	Reserved for the Volume Error flag
0x42	1	Extended boot signature	If value = 0x29, then the Extended boot signature indicates the following 3 fields are present
0x43	4	Volume Serial number	32-bit value usually generated from the date/time. Used for tracking removable media. This value can often be found in the LNK files (in Windows).
0x47	11	Volume label	11-byte volume label recorded in the Root directory. If no volume label is provided, then "NO NAME" is the default value.
0x52	8	File System Type	Although this field generally represents the file system formatted on the volume, it is an informational field only and is not used by FAT drivers to determine the FAT system type.

5.2 File Allocation Table (FAT) basics

The size of each entry within the FAT is determined by the FAT version. The number after the FAT is actually the number of bits used by the File Allocation Table for each entry. FAT16 = 16 bits for each entry; FAT32 = 32 bits for each entry. More bits per entry equates to more addressable clusters.

Media Descriptors
The first entry in the FAT table is the "Media Descriptor". It gives an indication as to the type of media on which the FAT File System is located as well as the type of FAT being used.

- 0xF0 = 3.5" single-sized floppy disk
- 0xF9 = 3.5" double-sized floppy disk
- 0xF8 = Hard disk drive

The next remaining bits of the "***Media Descriptor***" is the "***FAT Type Descriptor***" which gives an indication as to the type of the FAT file system itself (FAT12/16/32).

- FAT12 - 4bits after the Media Descriptor are 1 (eg. 0x0F)
- FAT16 - 8 bits after the Media Descriptor are 1 (eg. 0xFF)
- FAT32 - 20 bits after the Media Descriptor are 1 (eg. 0x0F 0xFF)

The bits after the "***Media Descriptor***" and "***FAT Type Descriptor***" are reserved. Cluster 1 is padded with either 0xFF or the "End of Cluster Chain" marker, depending on the OS used to format the volume. Cluster mapping values start after these 2 FAT entries with the Cluster 2 entry.

The first entry for a Hard disk with a FAT32 filesystem would be: 0x0FFFFFF8
The second entry for FAT32 would be: 0xFFFFFFFF

FAT table have 4 different types of entries:

1. Unallocated cluster: value of 0x00
2. Allocated cluster: value is the hex value of the next cluster in the cluster run
3. Allocated cluster: End of File (EOF), normally represented by 0xFB, 0xFF, 0x0F, depending on the OS writing to the FAT. For FAT32 this is normally 0x0FFFFFFF
4. Bad cluster: normally 0xF7

Symbolic value	FAT12 (hex)	FAT16 (hex)	FAT32 (hex)	Description
Unallocated	0 00	00 00	00 00 00 00	Unused cluster that is available for storage
Next cluster	0 02 - F EF	00 02 - FF FE	00 00 00 02 - FF FF FF EF	Cluster is in use and indicates the next cluster in the run
End of File	F FB	FF FB - FF FF	FF FF FF FB - FF FF FF 0F	Indicates the last cluster in the run (or that the run only contains one cluster of data
Bad Cluster	F F7	FF F7	FF FF FF F7	Cluster is bad and will not be used by the OS. Each sector is verified to ensure it is able to hold 0x200 bytes of info. If the sector is unable to hold 0x200 bytes of data, the sector is marked as BAD. A cluster marked as bad survives a quick format. (data of evidentiary value can be hidden in clusters marked as BAD in the FAT; also a user can mark a cluster as bad to hide data from average users; thus all clusters should be examined in detail when possible).

5.3 FAT32 Volume layout

Offset	Description
Start of Partition	Volume boot sector
Start of Partition + # of Reserved Sectors	FAT tables
Start of Partition + # of Reserved Sectors + (# of sectors/FAT * 2)	Data area

5.4 FAT Formatting options:

Command line	Type of forma	What actually happens
FORMAT A: /Q	"quick" format	1. The VBR will be verified and update with at least a new OEM ID, volume serial number, and volume label. 2. The FAT entries that contain a cluster number or EOF marker will all be changed to a 0x00. Clusters marked as BAD will not be changed. 3. The Root Directory entries will all be overwritten with 0x00. For FAT32, only the first cluster of the Root Directory is overwritten with 0x00
FORMAT A:	"normal" format	1. Each sector that is not marked as BAD in the FAT is checked for read errors. 2. Any newly discovered BAD sectors are updated in the FAT as BAD.
FORMAT A: /U	"unconditional" format (only available via command line)	1. Every sector on the media is verified as GOOD or BAD. 2. The FAT is updated to reflect the current status of that cluster. 3. Sectors previously marked as BAD in the FAT are rechecked and updated. 4. On removeable media only, every byte in the Data Area is overwritten with a value such as 0xE6, 0xF6, or 0x00, since the command will perform both a read and write test for each sector.

5.5 Long File Name (LFN) Directory Entry Structure

Offset	Length	Byte Usage
0x00	1	Bits 0-5 = LFN sequence number, bit 6 (0x40) is set if this is the last entry for the file.
0x01	10	1st 5 letters of the LFN entry
0x0B	1	0x0f (first nibble of attributes byte is set)
0x0C	1	Reserved; set to 0
0x0D	1	Checksum generate from SFN (Short Filename)
0x0E	12	Next 6 letters of the LFN entry
0x1A	2	Always 0

0x1C	4	Last 2 letters of the LFN entry

5.6 LFN Sequence Numbers

Directory Entry	Hex values of the Sequence Byte
1	0x41
2	0x01 0x42
3	0x01 0x02 0x43
4	0x01 0x02 0x03 0x44
5	0x01 0x02 0x03 0x04 0x45
6	0x01 0x02 0x03 0x04 0x05 0x46
7	0x01 0x02 0x03 0x04 0x05 0x06 0x47
8	0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x48
9	0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x49
10	0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x4A
11	0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x4B
12	0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x4C
13	0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x4D
14	0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x4E
15	0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x4F
16	0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F 0x50

6 Internals of the exFAT Filesystem

he Main Boot and the Backup Boot Region has the same sub-regions and data structures.

The Boot regions are created when the volume is formatted and can be further broken down into 5 separate data structures called sub-regions. These are:

1. Boot Sector
2. Extended Boot Sectors
3. OEM Parameters
4. Reserved
5. Boot Checksum

6.1 Main Boot Sector

Offset	Length	Name	Description
0x00	0x03	Jump Boot	Jump Instruction to boot code field
0x03	0x08	File system name	ASCII - exFAT
0x0B	0x35	Must be zero	Replaces the FAT BIOS parameter block
0x40	0x08	Partition offset	Sectors from the start of the media
0x48	0x08	Volume length	Total sectors in the volume
0x50	0x04	FAT offset	Logical start sector of FAT
0x54	0x04	FAT length	Length of the FAT table in sectors
0x58	0x04	Cluster heap offset	Logical start sector of the cluster heap
0x5C	0x04	Cluster count	Number of clusters in cluster heap
0x60	0x04	Volume serial number	
0x68	0x02	File system version	Major/minor
0x6A	0x02	Volume flags	(see below)
0x6C	0x01	Bytes per sector shift	2^N, where N = value for bytes per sector shift
0x6D	0x01	Sectors per cluster shift	2^N, where N = value for sectors per cluster shift
0x6E	0x01	Number of FATs	0x01 = 1 FAT and 1 Bitmap (current exFAT version) 0x02 = 2 FATs and 2 Bitmaps (TexFAT only)
0x70	0x01	Percent in use	Percentage of allocated clusters in the cluster heap. 0x00 - 0x64, 0xFF are not available
0x71	0x07	Reserved	

0x78	0x186	Boot code	Boot strapping instructions
0x1FE	0x02	Boot Signature	0x55AA [this signature will always be at this offset, regardless of the sector size. For example, if the sector size was 0x400, the signature would not be relocated to the end of the sector 0]. Thus, it is important to read the bytes per sector field located at offset 0x6C of the Boot Sector in an exFAT volume.
0x200		Excess space	If sector size > 0x200 bytes.

6.2 Boot Sector Volume Flags

Bit	Name	Description
0	Active FAT	Which FAT and Bitmap are in use. 0 = first FAT and first Bitmap 1 = second FAT and second Bitmap
1	Volume dirty	0 = Volume consistent 1 = Volume potential inconsistent
2	Media failure	0 = Any known failures marked as "bad" clusters 1 = Media reported failures
3	Clear to zero	No significant meaning (revision 1.00)
4	Reserved	Bits4-15 = Reserved

6.3 FAT Region

Sector 24 of an exFAT volume marks the beginning of the FAT region. The exFAT file system does not operate the FAT in the same way as the FAT32 file system. There are 2 major changes.

a. exFAT does not utilize the FAT for cluster allocation status; this is now done by a Bitmap file
b. exFAT uses the FAT for fragmented files only; if a file is in contiguous clusters (not fragmented) the FAT is unused for that file. This is annotated in the flags in the directory entry for that file; which will indicate if the file is contiguous or fragmented. For ***system*** files only, exFAT makes entries in both the FAT and the Bitmap sections.

Although sector 24 is the start of the FAT region, the 1st FAT will not necessarily be located at the beginning of the FAT region. While the current version of exFAT only contains 1 FAT sub-region, the spec has a definition for the 2nd FAT to be used for the Transaction-Safe exFAT (TexFAT) version.

6.4 Data Region

There are 10 different types of Directory Entries. There is no support for DOS compliant file names. No dot and double-dot directory entries. The additional of 2 system files (Bitmap and UpCase).

Below are the Directory Entry Types. The first three are used for "system files" and the Volume Label. The Volume GUID, TexFAT, Vendor Allocation, and Vendor Extension types are not currently in use.

Identifier (In Use)	Directory Entry Type	Identifier (Not in Use)
0x81	Allocation Bitmap	0x01
0x82	UpCase Table	0x02
0x83	Volume Label	0x03
0x85	File	0x05
0xA0	Volume GUID	0x20
0xA1	TexFAT Padding	0x21
0xC0	Stream Extension	0x40
0xC1	File name	0x41
	Vendor Extension	
	Vendor Allocation	

7 Authentication and the License File

This tool has authentication built into the binary. The primary authentication mechanism is the digital X509 code signing certificate embedded into the binary (Windows and macOS).

The other mechanism is the runtime authentication, which applies to all the versions of the tools (Windows, Linux and macOS). The runtime authentication ensures that the tool has a valid license. The license needs to be in the same directory of the tool for it to authenticate. Furthermore, any modification to the license, either to its name or contents, will invalidate the license.

8 References

1. Microsoft FAT32 Specification. Various sources including https://en.wikipedia.org/wiki/Design_of_the_FAT_file_system
2. Microsoft exFAT Specification [https://learn.microsoft.com/en-us/windows/win32/fileio/exfat-specification].
3. International Association of Computer Investigative Services (IACIS) Basic Computer Forensic Examiner (BCFE) class notes