fatmapper Documentation

Release 1.0.0

Patrick Neumann

Mar 26, 2017

Contents:

CHAPTER 1

INTRODUCTION

The task was to create a script to map a FAT16 (note the tool will probably work on other FAT's but is only required to work for FAT16 file systems) file system. The functionality should be similar to that of The SleuthKit's *fsstat* command. The tool should take a single command line argument (the **image file** to analyse) and an **optional offset** (see example below). If no offset is specified the **default value should be 0**.

When run as: python fatmapper.py -o 2048 image.dd the output from the tool should appear as:

```
File Information
-- Name:               image.dd
-- File Size:          134217728
-- MD5:                30E1D26A5C47EBD0E2FB770D470A23D0

FS Information
-- Volume Label:       NO NAME
-- OEM Name:           mkfs.fat
-- File System:        FAT 16
-- Volume ID:          0xAB0EA8BF

-- Sector Size:        512 bytes
-- Cluster Size:       2048 bytes (4 Sectors)
-- Number of Sectors: 262144
-- Number of FATs:     2

-- Total Range:        0 - 262143 (262144 Sectors)

FS Layout
-- Reserved:        0 - 0 (1 Sectors)
---- VBR:           0 - 0 (1 Sectors)
-- FAT 0:           1 - 256 (256 Sectors)
-- FAT 1:           257 - 512 (256 Sectors)
-- Data Area:       513 - 262143 (261631 Sectors)
---- Root Dir:      513 - 544 (32 Sectors)
```

The script should be well designed and well commented (see *USAGE* and *fatmapper MODULE (apidoc)*) and should handle error conditions gracefully (see *EXIT STATES*).

The autodoc feature is not really perfect but better than nothing.

For a report on the testing see *fatmapper TESTS*.

CHAPTER 2

USAGE

If you are new to fatmapper you may ask yourself how to use it. Here is the answer:

```
$ ./fatmapper.py --help
usage: fatmapper.py [-h] [-o OFFSET] image

positional arguments:
  image                 raw image file (esp.: image.dd)

optional arguments:
  -h, --help            show this help message and exit
  -o OFFSET, --offset OFFSET
                        offset in sectors (default=0)
```

CHAPTER 3

DESIGN

3.1 Sketch

While breaking down the features into smaller parts (= functions) I decided early to build a module.

I decided to accept and return single variables only if reasonable otherwise I prefere a dictionary.

That is the main reason for implementing some "private" functions (getargs and checks) to force other developers to pass the correct arguments to my functions.

While prototyping it has come out that calculating the MD5 hash for the hole image file will be the most time consuming feature (linear complexity) of fatmapper.

Although **precision** is more important than speed I decided to try to safe time whenever possible. Per example: I will use relative "f.seek()" (from the current pointer in the image) over absolute "f.seek()" (start every time from the beginning of the file). Or: Prefere internal hash of a dictionary over iterating over the hole dictionary to find a key in the index.

After some more time it felt incomplete to not support **FAT32**. Thats why I implemented it. The label is also stored in the root directory under all FAT varieties but only considered as part of the FAT32 support because it would break the given output for FAT16 (and FAT12) of Fergus.

Because I used simple debugging over and over I left it commented out in the source code for future use.

The development was mostly done under **macOS Sierra 10.12.3** and **Python 2.7.10**.

3.2 Documentation

More in the beginning than in the end I tried to find the right way of inline documentation.

Now I can say **The Hitchhiker's Guide** (http://docs.python-guide.org/en/latest/writing/style/) has inspired me the most. In addition there were some additional parts in the Google Python Style Guide (http://google.github.io/styleguide/pyguide.html) and PEP 8 – Style Guide for Python Code (https://www.python.org/dev/peps/pep-0008/).

What to place in the "Header" was mostly inspired by the Volatility and the Plaso (log2timeline) projects.

That is why I tested and decided to go with Spinx-doc (http://www.sphinx-doc.org/) for the "real" documentation. After some testing I have chosen the **NumPy Style Python Docstrings** (http://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_numpy.html) for the in line documentation.

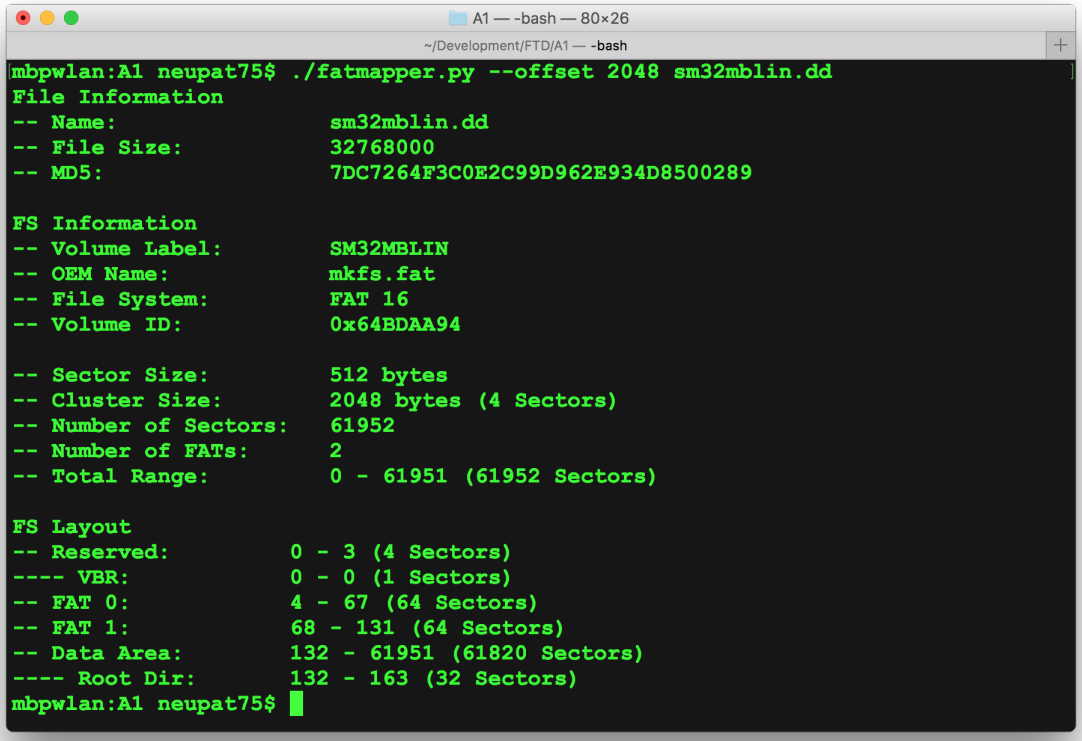
After finishing the first complete version of fatmapper I started to check the python syntax with Pylint (https://www.pylint.org/). While processing the errors and warnings from Pylint I fell to change "my" naming convention to fit Pylint (http://pylint-messages.wikidot.com/messages:c0103). As a side effect it produces his output in reStructuredText which I used as part of the "real documentation".

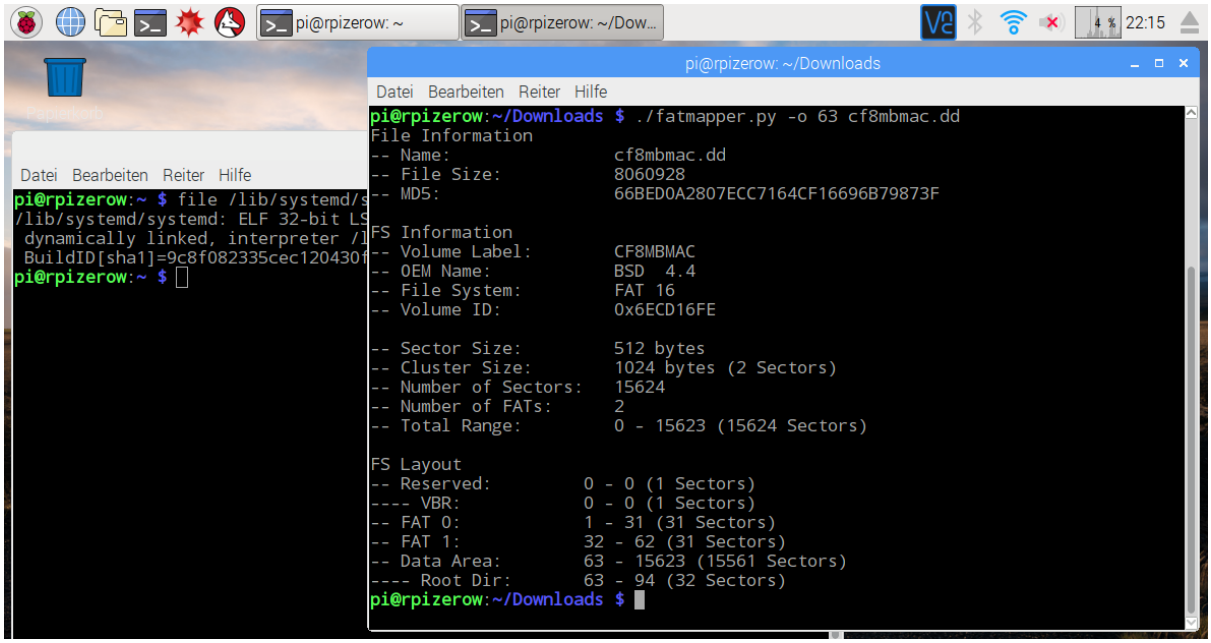
The "real" documentation was generated with **Spinx 1.5.3** and **Pylint 1.6.5** under macOS Sierra 10.12.3 and Python 2.7.10.

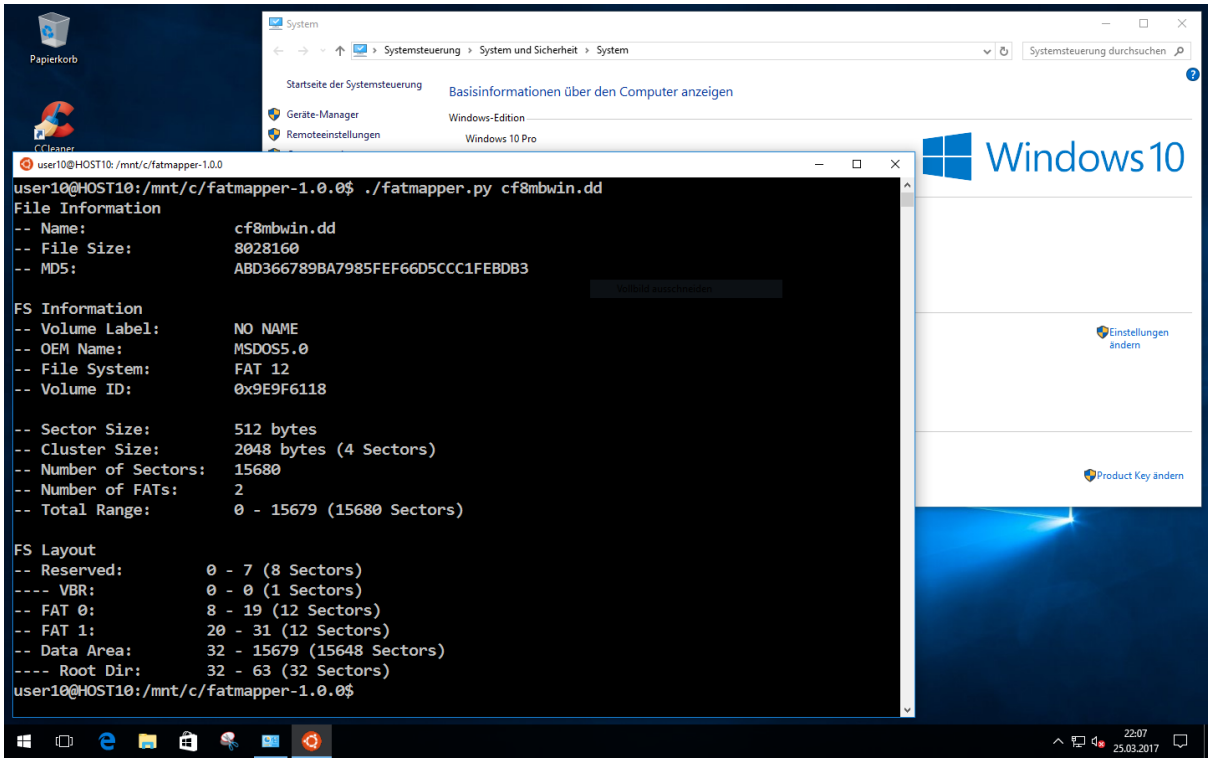
3.3 Tests

fatmapper was successfully tested on following systems:

- **macOS** (10.12.3) + python **2.7.10**
- Raspbian GNU/Linux 8.0 (**32-Bit**) + python **2.7.9**
- Bash on Ubuntu on **Windows 10** (64-Bit) + python **2.7.6**
- **ArchLinux** (64-Bit) + python **2.7.13**







fatmapper was successfully tested (incl. exit states) with following images:

- raw images
- filesystem(s) in partition(s)
- filesystem without a partition
- no partition/no filesystem (empty file)
- invalid filesystem
- FAT12, FAT16 and FAT32

CHAPTER 4

fatmapper MODULE (apidoc)

Display general details of a FAT file system.

fatmapper.**calculate**(*_file*, *loffset*, *_dict*)
Calcuate data.

Do some calculations with the raw/modified data.

Parameters **_dict** (*dict*) – dictionary with mapping of transformed parts of a fat VBR to a human readable index.

Returns dictionary with mapping of transformed parts and added calculations of a fat VBR to a human readable index.

Return type dict

fatmapper.**checkvbr**(*_file*, *loffset*)
Check for FAT VBR.

Check for the existence of a valid FAT12/16/32 VBR.

Parameters

- **_file** (*str*) – path to the image file.
- **loffset** (*int*) – offset in bytes in the image file.

Returns file_system_type (FAT1X or FAT32)

Return type str

Raises Exits if the VBR can not be verified as a FAT12/16/32 VBR.

fatmapper.**getlabel2**(*_file*, *loffset*, *_dict*)
Get FAT32 root directory data.

Read raw data about the label from the FAT32 root directory into a python dictionary.

Parameters

- **loffset** (*int*) – offset in bytes in the image file.
- **_dict** (*dict*) – dictionary with mapping of parts of a fat VBR to a human readable index.

Returns volume label from root directory

Return type str

fatmapper.**getmd5**(*_file*)
Get MD5 of image file.

Get MD5 digest in hex for the content of the image file.

Parameters **_file** (*str*) – path to the image file.

Returns MD5 digest in uppercase hex.

Return type str

fatmapper.**getraw**(*_file*, *loffset*)
Get global data.

Read raw data from the VBR into a python dictionary.

Parameters

- **_file** (*str*) – path to the image file.
- **loffset** (*int*) – offset in bytes in the image file.

Returns

dictionary with mapping of parts of a fat VBR to a human readable index.

For example:

{"bytesPerSector": 512, "sectorsPerCluster": 64, "reservedSectors": 1, "fatCopies": 2, "rootDirectoryEntries": 1024, "sectorsInPartition": 0, "rawMediaDescriptor": 248, "sectorsPerFat": 64, "sectorsPerTrack": 32, "numberOfHeads": 64, "hiddenSectors": 0, "largerSectorsInPartition": 260096}

Return type dict

fatmapper.**getraw1x**(*_file*, *loffset*)
Get FAT12/16 spezial data.

Read raw data from the VBR into a python dictionary.

Parameters

- **_file** (*str*) – path to the image file.
- **loffset** (*int*) – offset in bytes in the image file.

Returns

dictionary with mapping of parts of a fat VBR to a human readable index.

For example:

{"logicalDriveNumber": 128, "RESERVED": 0, "extendedSignature": 41, "rawVolumeSerialNumber": 549101972, "oemNameVersion": "mkfs.fat", "volumeLabel": "NO NAME ", "rawFileSystemType": "FAT12 ", "vbrSignature": "U"}

Return type dict

fatmapper.**getraw32**(*_file*, *loffset*)
Get FAT32 spezial data.

Read raw data from the VBR into a python dictionary.

Parameters

- **_file** (*str*) – path to the image file.
- **loffset** (*int*) – offset in bytes in the image file.

Returns

dictionary with mapping of parts of a fat VBR to a human readable index.

For example:

{"sectorsPerFat": 947, "mirrorFlags": 0, "fileSystemVersion": 0, "firstClusterOfRootDirectory": 2, "fsinfoSector": 1, "backupBootSector": 6, "logicalDriveNumber": 128, "RESERVED": 0, "extendedSignature": 41, "rawVolumeSerialNumber": 774266948, "volumeLabel": "NO NAME ", "rawFileSystemType": "FAT32 ", "vbrSignature": "Uª"}

Return type dict

fatmapper.**getrootdirsize32**(*_file*, *loffset*, *_dict*)

Get FAT32 root directory informations.

Read raw data about the root directory from the FAT32 VBR into a python dictionary.

Parameters

- **loffset** (*int*) – offset in bytes in the image file.
- **_dict** (*dict*) – dictionary with mapping of parts of a fat VBR to a human readable index.

Returns dictionary with start, end and size of the root directory.

Return type dict

fatmapper.**getsize**(*_file*, *loffset*)

Get image file size.

Get size of the image file.

Parameters

- **_file** (*str*) – path to the image file.
- **loffset** (*int*) – offset in bytes in the image file.

Returns size of image file in bytes.

Return type int

Raises Exits if image file is empty or smaller than offset.

fatmapper.**main**(*_file*, *loffset*, *_dict*)

Print data.

Insert the data into the template and print it on stdout.

Parameters

- **_file** (*str*) – path to the image file.
- **loffset** (*int*) – offset in bytes in the image file.
- **_dict** (*dict*) – dictionary with mapping of transformed parts and added calculations of a fat VBR to a human readable index.

fatmapper.**transform**(*_dict*)

Transform data.

Do some modifications to the raw data.

Parameters **_dict** (*dict*) – dictionary with mapping of parts of a fat VBR to a human readable index.

Returns dictionary with mapping of transformed parts of a fat VBR to a human readable index.

Return type dict

CHAPTER 5

PYLINT REPORT

252 statements analysed.

5.1 Statistics by type

type	number	old number	difference	%documented	%badname
module	1	1	=	100.00	0.00
class	0	0	=	0	0
method	0	0	=	0	0
function	15	15	=	100.00	0.00

5.2 Raw metrics

type	number	%	previous	difference
code	300	34.92	300	=
docstring	348	40.51	348	=
comment	112	13.04	112	=
empty	99	11.53	99	=

5.3 Duplication

	now	previous	difference
nb duplicated lines	0	0	=
percent duplicated lines	0.000	0.000	=

5.4 Messages by category

type	number	previous	difference
convention	0	0	=
refactor	0	0	=
warning	0	0	=
error	0	0	=

5.5 Global evaluation

Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)

CHAPTER 6

EXIT STATES

6.1 Usage (2)

If too few:

```
$ ./fatmapper.py
usage: fatmapper.py [-h] [-o OFFSET] image
fatmapper.py: error: too few arguments
```

or too many arguments are supplied:

```
$ ./fatmapper.py --offset 0 --too many image.dd
usage: fatmapper.py [-h] [-o OFFSET] image
fatmapper.py: error: unrecognized arguments: --too image.dd
```

The exit state of 2 is set by the argparse module.

6.2 Python version (1)

Too old:

```
$ ./fatmapper.py
Sorry, your Python version was not tested but may work?
```

Phyton version 3:

```
$ ./fatmapper.py
Error: Python 3.x is not supported!
```

6.3 Image file (2)

If another developer use this module every function that exepts a file runs a check for its existence:

```
$ ./fatmapper.py thisimagedoesnotexist.dd
Error: image file does not exist!
```

6.4 Offset (3)

If another developer use this module every function that exepts a offset runs a check if the offset is a) an integer:

```
$ ./fatmapper.py --offset string cf8mbwin.dd
usage: fatmapper.py [-h] [-o OFFSET] image
fatmapper.py: error: argument -o/--offset: invalid int value: 'string'
```

and b) positive:

```
$ ./fatmapper.py --offset -1 cf8mbwin.dd
Error: offset is not equal or greater zero!
```

6.5 Dictionary (4)

If another developer use this module every function that exepts a dictionary runs a check if the dictionary contains a) necessary data:

```
$ ./fatmapper_dict_missing_data.py
Error: necessary data is missing in dictionary!
```

and b) the data has the right type:

```
$ ./fatmapper_dict_wrong_type.py
Error: invalid type of data in dictionary!
```

6.6 Size vs. Offset (5/6)

There is no place for a filesystem in an empty file:

```
$ ./fatmapper.py empty.dd
Error: the image file is an empty file!
```

If the offset is too big there will not be enough place for a filesystem:

```
$ ./fatmapper.py --offset 15680 cf8mbwin.dd
Error: size of the image file is smaller than offset (+ 512 byte)!
```

6.7 Filesystems (7)

fatmapper does only support FAT12/FAT16 filesystems:

```
$ ./fatmapper.py invalid.dd
Error: no valid FAT12/FAT16/FAT32 VBR found!
```

CHAPTER 7

fatmapper TESTS

7.1 8MB CF-Card

For the 1st test I formated the storage media under M$ **Windows 10** by using the GUI (FAT):

```
$ ./fatmapper.py cf8mbwin.dd
File Information
-- Name:                cf8mbwin.dd
-- File Size:           8028160
-- MD5:                 ABD366789BA7985FEF66D5CCC1FEBDB3

FS Information
-- Volume Label:        NO NAME
-- OEM Name:            MSDOS5.0
-- File System:         FAT 12
-- Volume ID:           0x9E9F6118

-- Sector Size:         512 bytes
-- Cluster Size:        2048 bytes (4 Sectors)
-- Number of Sectors:   15680
-- Number of FATs:      2
-- Total Range:         0 - 15679 (15680 Sectors)

FS Layout
-- Reserved:         0 - 7 (8 Sectors)
---- VBR:            0 - 0 (1 Sectors)
-- FAT 0:            8 - 19 (12 Sectors)
-- FAT 1:            20 - 31 (12 Sectors)
-- Data Area:        32 - 15679 (15648 Sectors)
---- Root Dir:       32 - 63 (32 Sectors)
```

After the fatmapper run I have **successfully** verified the output against *openssl* and *fsstat* of The SleuthKit:

```
$ openssl dgst -md5 cf8mbwin.dd
MD5(cf8mbwin.dd)= abd366789ba7985fef66d5ccc1febdb3
$ fsstat cf8mbwin.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT12
```

```
OEM Name: MSDOS5.0
Volume ID: 0x9e9f6118
Volume Label (Boot Sector): NO NAME
Volume Label (Root Directory): CF8MBWIN
File System Type Label: FAT12

Sectors before file system: 0

File System Layout (in sectors)
Total Range: 0 - 15679
* Reserved: 0 - 7
** Boot Sector: 0
* FAT 0: 8 - 19
* FAT 1: 20 - 31
* Data Area: 32 - 15679
** Root Directory: 32 - 63
** Cluster Area: 64 - 15679

METADATA INFORMATION
------------------------------------------
Range: 2 - 250374
Root Directory: 2

CONTENT INFORMATION
------------------------------------------
Sector Size: 512
Cluster Size: 2048
Total Cluster Range: 2 - 3905

FAT CONTENTS (in sectors)
------------------------------------------
64-67 (4) -> EOF
68-71 (4) -> EOF
72-75 (4) -> EOF
```

For the 2nd test I formated the storage media under **macOS Sierra** by using the GUI too:

```
$ ./fatmapper.py --offset 63 cf8mbmac.dd
File Information
-- Name:                cf8mbmac.dd
-- File Size:           8060928
-- MD5:                 66BED0A2807ECC7164CF16696B79873F

FS Information
-- Volume Label:        CF8MBMAC
-- OEM Name:            BSD  4.4
-- File System:         FAT 16
-- Volume ID:           0x6ECD16FE

-- Sector Size:         512 bytes
-- Cluster Size:        1024 bytes (2 Sectors)
-- Number of Sectors:   15624
-- Number of FATs:      2
-- Total Range:         0 - 15623 (15624 Sectors)

FS Layout
-- Reserved:         0 - 0 (1 Sectors)
---- VBR:            0 - 0 (1 Sectors)
-- FAT 0:            1 - 31 (31 Sectors)
-- FAT 1:            32 - 62 (31 Sectors)
-- Data Area:        63 - 15623 (15561 Sectors)
---- Root Dir:       63 - 94 (32 Sectors)
```

The Verification was **successful** too:

```
$ openssl dgst -md5 cf8mbmac.dd
MD5(cf8mbmac.dd)= 66bed0a2807ecc7164cf16696b79873f
$ fsstat -o 63 cf8mbmac.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT16

OEM Name: BSD  4.4
Volume ID: 0x6ecd16fe
Volume Label (Boot Sector): CF8MBMAC
Volume Label (Root Directory):
File System Type Label: FAT16

Sectors before file system: 63

File System Layout (in sectors)
Total Range: 0 - 15623
* Reserved: 0 - 0
** Boot Sector: 0
* FAT 0: 1 - 31
* FAT 1: 32 - 62
* Data Area: 63 - 15623
** Root Directory: 63 - 94
** Cluster Area: 95 - 15622
** Non-clustered: 15623 - 15623

METADATA INFORMATION
--------------------------------------------
Range: 2 - 248982
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 1024
Total Cluster Range: 2 - 7765

FAT CONTENTS (in sectors)
--------------------------------------------
95-96 (2) -> EOF
97-98 (2) -> EOF
99-100 (2) -> EOF
101-102 (2) -> EOF
103-104 (2) -> EOF
105-106 (2) -> EOF
```

For the 3th test I formated the storage media unter **Arch Linux** by using “fdisk” and “mkfs.vfat” in a terminal:

```
$ ./fatmapper.py --offset 2048 cf8mblin.dd
File Information
-- Name:                cf8mblin.dd
-- File Size:           8028160
-- MD5:                 5099DEBA212FD5615B190D3E5A1212AE

FS Information
-- Volume Label:        CF8MBLIN
-- OEM Name:            mkfs.fat
-- File System:         FAT 12
-- Volume ID:           0x4D1CB86A
```

```
-- Sector Size:          512 bytes
-- Cluster Size:         2048 bytes (4 Sectors)
-- Number of Sectors:    13632
-- Number of FATs:       2
-- Total Range:          0 - 13631 (13632 Sectors)

FS Layout
-- Reserved:          0 - 3 (4 Sectors)
---- VBR:             0 - 0 (1 Sectors)
-- FAT 0:             4 - 15 (12 Sectors)
-- FAT 1:             16 - 27 (12 Sectors)
-- Data Area:         28 - 13631 (13604 Sectors)
---- Root Dir:        28 - 59 (32 Sectors)
```

The Verification was **successful** too:

```
$ openssl dgst -md5 cf8mblin.dd
MD5(cf8mblin.dd)= 5099deba212fd5615b190d3e5a1212ae
$ fsstat -o 2048 cf8mblin.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT12

OEM Name: mkfs.fat
Volume ID: 0x4d1cb86a
Volume Label (Boot Sector): CF8MBLIN
Volume Label (Root Directory): CF8MBLIN
File System Type Label: FAT12

Sectors before file system: 2048

File System Layout (in sectors)
Total Range: 0 - 13631
* Reserved: 0 - 3
** Boot Sector: 0
* FAT 0: 4 - 15
* FAT 1: 16 - 27
* Data Area: 28 - 13631
** Root Directory: 28 - 59
** Cluster Area: 60 - 13631

METADATA INFORMATION
--------------------------------------------
Range: 2 - 217670
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 2048
Total Cluster Range: 2 - 3394

FAT CONTENTS (in sectors)
--------------------------------------------
```

7.2 32MB SM-Card

For the 4th test I formated the storage media under M$ Windows 10 by using the GUI (FAT):

```
$ ./fatmapper.py sm32mbwin.dd
File Information
-- Name:                sm32mbwin.dd
-- File Size:           32768000
-- MD5:                 A350F05A57E1CD945BDA0ED106080F1B

FS Information
-- Volume Label:        NO NAME
-- OEM Name:            MSDOS5.0
-- File System:         FAT 16
-- Volume ID:           0xAC984317

-- Sector Size:         512 bytes
-- Cluster Size:        512 bytes (1 Sectors)
-- Number of Sectors:   64000
-- Number of FATs:      2
-- Total Range:         0 - 63999 (64000 Sectors)

FS Layout
-- Reserved:         0 - 7 (8 Sectors)
---- VBR:            0 - 0 (1 Sectors)
-- FAT 0:            8 - 255 (248 Sectors)
-- FAT 1:            256 - 503 (248 Sectors)
-- Data Area:        504 - 63999 (63496 Sectors)
---- Root Dir:       504 - 535 (32 Sectors)
```

The Verification was **successful** too:

```
$ openssl dgst -md5 sm32mbwin.dd
MD5(sm32mbwin.dd)= a350f05a57e1cd945bda0ed106080f1b
$ fsstat sm32mbwin.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT16

OEM Name: MSDOS5.0
Volume ID: 0xac984317
Volume Label (Boot Sector): NO NAME
Volume Label (Root Directory): SM32MB
File System Type Label: FAT16

Sectors before file system: 0

File System Layout (in sectors)
Total Range: 0 - 63999
* Reserved: 0 - 7
** Boot Sector: 0
* FAT 0: 8 - 255
* FAT 1: 256 - 503
* Data Area: 504 - 63999
** Root Directory: 504 - 535
** Cluster Area: 536 - 63999

METADATA INFORMATION
--------------------------------------------
Range: 2 - 1015942
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 512
Total Cluster Range: 2 - 63465
```

```
FAT CONTENTS (in sectors)
-----------------------------------------
536-536 (1) -> EOF
537-537 (1) -> EOF
538-538 (1) -> EOF
539-539 (1) -> EOF
540-540 (1) -> EOF
```

For the 5th test I formated the storage media under macOS Sierra by using the GUI too:

```
$ ./fatmapper.py --offset 63 sm32mbmac.dd
File Information
-- Name:                sm32mbmac.dd
-- File Size:           32768000
-- MD5:                 86B7CAFAAB98751338F6EFB817802324

FS Information
-- Volume Label:        SM32MBMAC
-- OEM Name:            BSD  4.4
-- File System:         FAT 16
-- Volume ID:           0x702E160F

-- Sector Size:         512 bytes
-- Cluster Size:        2048 bytes (4 Sectors)
-- Number of Sectors:   63882
-- Number of FATs:      2
-- Total Range:         0 - 63881 (63882 Sectors)

FS Layout
-- Reserved:         0 - 0 (1 Sectors)
---- VBR:            0 - 0 (1 Sectors)
-- FAT 0:            1 - 63 (63 Sectors)
-- FAT 1:            64 - 126 (63 Sectors)
-- Data Area:        127 - 63881 (63755 Sectors)
---- Root Dir:       127 - 158 (32 Sectors)
```

The Verification was **successful** too:

```
$ openssl dgst -md5 sm32mbmac.dd
MD5(sm32mbmac.dd)= 86b7cafaab98751338f6efb817802324
$ fsstat -o 63 sm32mbmac.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT16

OEM Name: BSD  4.4
Volume ID: 0x702e160f
Volume Label (Boot Sector): SM32MBMAC
Volume Label (Root Directory):
File System Type Label: FAT16

Sectors before file system: 63

File System Layout (in sectors)
Total Range: 0 - 63881
* Reserved: 0 - 0
** Boot Sector: 0
* FAT 0: 1 - 63
* FAT 1: 64 - 126
* Data Area: 127 - 63881
** Root Directory: 127 - 158
** Cluster Area: 159 - 63878
```

```
** Non-clustered: 63879 - 63881

METADATA INFORMATION
--------------------------------------------
Range: 2 - 1020086
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 2048
Total Cluster Range: 2 - 15931

FAT CONTENTS (in sectors)
--------------------------------------------
159-162 (4) -> EOF
163-166 (4) -> EOF
167-170 (4) -> EOF
171-174 (4) -> EOF
175-178 (4) -> EOF
179-182 (4) -> EOF
183-186 (4) -> EOF
```

For the 6th test I formated the storage media unter Arch Linux by using “fdisk” and “mkfs.vfat” in a terminal:

```
$ ./fatmapper.py --offset 2048 sm32mblin.dd
File Information
-- Name:                sm32mblin.dd
-- File Size:           32768000
-- MD5:                 7DC7264F3C0E2C99D962E934D8500289

FS Information
-- Volume Label:        SM32MBLIN
-- OEM Name:            mkfs.fat
-- File System:         FAT 16
-- Volume ID:           0x64BDAA94

-- Sector Size:         512 bytes
-- Cluster Size:        2048 bytes (4 Sectors)
-- Number of Sectors:   61952
-- Number of FATs:      2
-- Total Range:         0 - 61951 (61952 Sectors)

FS Layout
-- Reserved:         0 - 3 (4 Sectors)
---- VBR:           0 - 0 (1 Sectors)
-- FAT 0:           4 - 67 (64 Sectors)
-- FAT 1:           68 - 131 (64 Sectors)
-- Data Area:       132 - 61951 (61820 Sectors)
---- Root Dir:      132 - 163 (32 Sectors)
```

The Verification was **successful** too:

```
$ openssl dgst -md5 sm32mblin.dd
MD5(sm32mblin.dd)= 7dc7264f3c0e2c99d962e934d8500289
$ fsstat -o 2048 sm32mblin.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT16

OEM Name: mkfs.fat
Volume ID: 0x64bdaa94
Volume Label (Boot Sector): SM32MBLIN
```

```
Volume Label (Root Directory): SM32MBLIN
File System Type Label: FAT16

Sectors before file system: 2048

File System Layout (in sectors)
Total Range: 0 - 61951
* Reserved: 0 - 3
** Boot Sector: 0
* FAT 0: 4 - 67
* FAT 1: 68 - 131
* Data Area: 132 - 61951
** Root Directory: 132 - 163
** Cluster Area: 164 - 61951

METADATA INFORMATION
------------------------------------------
Range: 2 - 989126
Root Directory: 2

CONTENT INFORMATION
------------------------------------------
Sector Size: 512
Cluster Size: 2048
Total Cluster Range: 2 - 15448

FAT CONTENTS (in sectors)
------------------------------------------
```

7.3 128MB CF-Card

For the 7th test I formated the storage media under M$ Windows 10 by using the GUI (FAT32):

```
$ ./fatmapper.py cf128mbwin.dd
File Information
-- Name:                cf128mbwin.dd
-- File Size:           128188416
-- MD5:                 6BDD6A12AEDC6F2AC8D1B3AA957978C9

FS Information
-- Volume Label:        NO NAME
-- Volume Label #2:     CF128MB
-- OEM Name:            MSDOS5.0
-- File System:         FAT 32
-- Volume ID:           0x2E266044

-- Sector Size:         512 bytes
-- Cluster Size:        1024 bytes (2 Sectors)
-- Number of Sectors:   250368
-- Number of FATs:      2
-- Total Range:         0 - 250367 (250368 Sectors)

FS Layout
-- Reserved:         0 - 6297 (6298 Sectors)
---- VBR:            0 - 0 (1 Sectors)
-- FAT 0:            6298 - 7244 (947 Sectors)
-- FAT 1:            7245 - 8191 (947 Sectors)
-- Data Area:        8192 - 250367 (242176 Sectors)
---- Root Dir:       8192 - 8193 (2 Sectors)
```

The Verification was **successful** too:

```
$ openssl dgst -md5 cf128mbwin.dd
MD5(cf128mbwin.dd)= 6bdd6a12aedc6f2ac8d1b3aa957978c9
$ fsstat cf128mbwin.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT32

OEM Name: MSDOS5.0
Volume ID: 0x2e266044
Volume Label (Boot Sector): NO NAME
Volume Label (Root Directory): CF128MB
File System Type Label: FAT32
Next Free Sector (FS Info): 8204
Free Sector Count (FS Info): 242164

Sectors before file system: 0

File System Layout (in sectors)
Total Range: 0 - 250367
* Reserved: 0 - 6297
** Boot Sector: 0
** FS Info Sector: 1
** Backup Boot Sector: 6
* FAT 0: 6298 - 7244
* FAT 1: 7245 - 8191
* Data Area: 8192 - 250367
** Cluster Area: 8192 - 250367
*** Root Directory: 8192 - 8193

METADATA INFORMATION
--------------------------------------------
Range: 2 - 3874822
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 1024
Total Cluster Range: 2 - 121089

FAT CONTENTS (in sectors)
--------------------------------------------
8192-8193 (2) -> EOF
8194-8195 (2) -> EOF
8196-8197 (2) -> EOF
8198-8199 (2) -> EOF
8200-8201 (2) -> EOF
8202-8203 (2) -> EOF
```

For the 8th test I formated the storage media under macOS Sierra by using the GUI too:

```
$ ./fatmapper.py --offset 63 cf128mbmac.dd
File Information
-- Name:                cf128mbmac.dd
-- File Size:           128188416
-- MD5:                 C337E6C43F53D5E82F81F04FEC9782C1

FS Information
-- Volume Label:        CF128MBMAC
-- OEM Name:            BSD  4.4
-- File System:         FAT 16
```

```
-- Volume ID:           0x2EAD1615

-- Sector Size:         512 bytes
-- Cluster Size:        2048 bytes (4 Sectors)
-- Number of Sectors:   250299
-- Number of FATs:      2
-- Total Range:         0 - 250298 (250299 Sectors)

FS Layout
-- Reserved:         0 - 0 (1 Sectors)
---- VBR:            0 - 0 (1 Sectors)
-- FAT 0:            1 - 244 (244 Sectors)
-- FAT 1:            245 - 488 (244 Sectors)
-- Data Area:        489 - 250298 (249810 Sectors)
---- Root Dir:       489 - 520 (32 Sectors)
```

The Verification was **successful** too:

```
$ openssl dgst -md5 cf128mbmac.dd
MD5(cf128mbmac.dd)= c337e6c43f53d5e82f81f04fec9782c1
$ fsstat -o 63 cf128mbmac.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT16

OEM Name: BSD  4.4
Volume ID: 0x2ead1615
Volume Label (Boot Sector): CF128MBMAC
Volume Label (Root Directory):
File System Type Label: FAT16

Sectors before file system: 63

File System Layout (in sectors)
Total Range: 0 - 250298
* Reserved: 0 - 0
** Boot Sector: 0
* FAT 0: 1 - 244
* FAT 1: 245 - 488
* Data Area: 489 - 250298
** Root Directory: 489 - 520
** Cluster Area: 521 - 250296
** Non-clustered: 250297 - 250298

METADATA INFORMATION
--------------------------------------------
Range: 2 - 3996966
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 2048
Total Cluster Range: 2 - 62445

FAT CONTENTS (in sectors)
--------------------------------------------
521-524 (4) -> EOF
525-528 (4) -> EOF
529-532 (4) -> EOF
533-536 (4) -> EOF
537-540 (4) -> EOF
541-544 (4) -> EOF
```

```
545-552 (8) -> 1993
553-556 (4) -> EOF
557-560 (4) -> EOF
561-576 (16) -> EOF
577-580 (4) -> EOF
581-584 (4) -> EOF
645-652 (8) -> EOF
981-1044 (64) -> EOF
1045-1052 (8) -> EOF
1053-1060 (8) -> EOF
1061-1076 (16) -> EOF
1077-1084 (8) -> EOF
1085-1100 (16) -> EOF
1101-1120 (20) -> EOF
1121-1124 (4) -> EOF
1125-1252 (128) -> EOF
1253-1380 (128) -> EOF
1381-1580 (200) -> 1917
1581-1780 (200) -> 2049
1781-1908 (128) -> EOF
1917-1948 (32) -> EOF
1949-1952 (4) -> EOF
1953-1956 (4) -> EOF
1957-1960 (4) -> EOF
1961-1964 (4) -> EOF
1965-1968 (4) -> EOF
1969-1972 (4) -> EOF
1973-1976 (4) -> EOF
1977-1980 (4) -> EOF
1981-1984 (4) -> EOF
1985-1988 (4) -> EOF
1989-1992 (4) -> EOF
1993-1996 (4) -> 2397
1997-2000 (4) -> EOF
2001-2008 (8) -> EOF
2009-2012 (4) -> EOF
2013-2016 (4) -> EOF
2017-2020 (4) -> EOF
2021-2024 (4) -> EOF
2025-2032 (8) -> EOF
2033-2040 (8) -> EOF
2041-2044 (4) -> EOF
2045-2048 (4) -> EOF
2049-2080 (32) -> EOF
2081-2088 (8) -> EOF
2089-2092 (4) -> EOF
2093-2096 (4) -> EOF
2097-2104 (8) -> EOF
2105-2108 (4) -> EOF
2109-2116 (8) -> EOF
2125-2128 (4) -> EOF
2129-2136 (8) -> EOF
2137-2144 (8) -> EOF
2149-2152 (4) -> EOF
2173-2180 (8) -> EOF
2193-2196 (4) -> EOF
2241-2244 (4) -> EOF
2253-2380 (128) -> EOF
2393-2396 (4) -> EOF
2397-2400 (4) -> EOF
2401-2404 (4) -> EOF
2417-2420 (4) -> EOF
2421-2548 (128) -> 2677
```

```
2549-2676 (128) -> EOF
2677-2680 (4) -> EOF
```

For the last test I formated the storage media unter Arch Linux by using “fdisk” and “mkfs.vfat” (FAT32) in a terminal:

```
$ ./fatmapper.py --offset 2048 cf128mblin.dd
File Information
-- Name:                cf128mblin.dd
-- File Size:           128188416
-- MD5:                 1B927CEC90D244AA8B0AA76B1EDEB3EA

FS Information
-- Volume Label:        CF128MBLIN
-- Volume Label #2:     CF128MBLIN
-- OEM Name:            mkfs.fat
-- File System:         FAT 32
-- Volume ID:           0x7A410776

-- Sector Size:         512 bytes
-- Cluster Size:        512 bytes (1 Sectors)
-- Number of Sectors:   248320
-- Number of FATs:      2
-- Total Range:         0 - 248319 (248320 Sectors)

FS Layout
-- Reserved:         0 - 31 (32 Sectors)
---- VBR:            0 - 0 (1 Sectors)
-- FAT 0:            32 - 1941 (1910 Sectors)
-- FAT 1:            1942 - 3851 (1910 Sectors)
-- Data Area:        3852 - 248319 (244468 Sectors)
---- Root Dir:       3852 - 3852 (1 Sectors)
```

The Verification was **successful** too:

```
$ openssl dgst -md5 cf128mblin.dd
MD5(cf128mblin.dd)= 1b927cec90d244aa8b0aa76b1edeb3ea
$ fsstat -o 2048 cf128mblin.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT32

OEM Name: mkfs.fat
Volume ID: 0x7a410776
Volume Label (Boot Sector): CF128MBLIN
Volume Label (Root Directory): CF128MBLIN
File System Type Label: FAT32
Next Free Sector (FS Info): 3852
Free Sector Count (FS Info): 244467

Sectors before file system: 2048

File System Layout (in sectors)
Total Range: 0 - 248319
* Reserved: 0 - 31
** Boot Sector: 0
** FS Info Sector: 1
** Backup Boot Sector: 6
* FAT 0: 32 - 1941
* FAT 1: 1942 - 3851
* Data Area: 3852 - 248319
** Cluster Area: 3852 - 248319
```

```
*** Root Directory: 3852 - 3852

METADATA INFORMATION
--------------------------------------------
Range: 2 - 3911494
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 512
Total Cluster Range: 2 - 244469

FAT CONTENTS (in sectors)
--------------------------------------------
3852-3852 (1) -> EOF
```

7.4 A real world example

For a read world example I downloaded the Raspbian Jessie Lite image (https://downloads.raspberrypi.org/raspbian_lite_latest) and unzipped it:

```
$ ./fatmapper.py --offset 8192 2017-03-02-raspbian-jessie-lite.img
File Information
-- Name:                2017-03-02-raspbian-jessie-lite.img
-- File Size:           1393557504
-- MD5:                 CE6660A13974E00C27A63B33A36EF895

FS Information
-- Volume Label:        boot
-- Volume Label #2:     boot
-- OEM Name:            mkfs.fat
-- File System:         FAT 32
-- Volume ID:           0x70CDBC89

-- Sector Size:         512 bytes
-- Cluster Size:        512 bytes (1 Sectors)
-- Number of Sectors:   129024
-- Number of FATs:      2
-- Total Range:         0 - 129023 (129024 Sectors)

FS Layout
-- Reserved:         0 - 31 (32 Sectors)
---- VBR:            0 - 0 (1 Sectors)
-- FAT 0:            32 - 1024 (993 Sectors)
-- FAT 1:            1025 - 2017 (993 Sectors)
-- Data Area:        2018 - 129023 (127006 Sectors)
---- Root Dir:       2018 - 10502 (4 Sectors)
```

The Verification was **successful** too:

```
$ openssl dgst -md5 2017-03-02-raspbian-jessie-lite.img
MD5(2017-03-02-raspbian-jessie-lite.img)= ce6660a13974e00c27a63b33a36ef895
$ fsstat -o 8192 2017-03-02-raspbian-jessie-lite.img
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT32

OEM Name: mkfs.fat
Volume ID: 0x70cdbc89
```

```
Volume Label (Boot Sector): boot
Volume Label (Root Directory): boot
File System Type Label: FAT32
Next Free Sector (FS Info): 43526
Free Sector Count (FS Info): 85497

Sectors before file system: 0

File System Layout (in sectors)
Total Range: 0 - 129023
* Reserved: 0 - 31
** Boot Sector: 0
** FS Info Sector: 1
** Backup Boot Sector: 6
* FAT 0: 32 - 1024
* FAT 1: 1025 - 2017
* Data Area: 2018 - 129023
** Cluster Area: 2018 - 129023
*** Root Directory: 2018 - 10502

METADATA INFORMATION
------------------------------------------
Range: 2 - 2032102
Root Directory: 2

CONTENT INFORMATION
------------------------------------------
Sector Size: 512
Cluster Size: 512
Total Cluster Range: 2 - 127007

FAT CONTENTS (in sectors)
------------------------------------------
2018-2018 (1) -> 2117
2019-2019 (1) -> 43148
2020-2056 (37) -> EOF
2057-2059 (3) -> EOF
2060-2088 (29) -> EOF
2089-2116 (28) -> EOF
2117-2117 (1) -> 2266
2118-2145 (28) -> EOF
2146-2173 (28) -> EOF
2174-2203 (30) -> EOF
2204-2235 (32) -> EOF
2236-2265 (30) -> EOF
2266-2266 (1) -> 10502
2267-2365 (99) -> EOF
2366-2366 (1) -> EOF
2367-2370 (4) -> EOF
2371-2383 (13) -> EOF
2384-2388 (5) -> EOF
2389-2408 (20) -> EOF
2409-2428 (20) -> EOF
2429-10501 (8073) -> EOF
10502-10502 (1) -> EOF
10503-18774 (8272) -> EOF
18775-24335 (5561) -> EOF
24336-25615 (1280) -> EOF
25616-35348 (9733) -> EOF
35349-43024 (7676) -> EOF
43025-43134 (110) -> EOF
43135-43137 (3) -> EOF
43138-43142 (5) -> EOF
```

```
43143-43147 (5) -> EOF
43148-43148 (1) -> 43162
43149-43151 (3) -> EOF
43152-43153 (2) -> EOF
43154-43157 (4) -> EOF
43158-43159 (2) -> EOF
43160-43161 (2) -> EOF
43162-43162 (1) -> 43181
43163-43163 (1) -> EOF
43164-43165 (2) -> EOF
43166-43167 (2) -> EOF
43168-43169 (2) -> EOF
43170-43171 (2) -> EOF
43172-43172 (1) -> EOF
43173-43174 (2) -> EOF
43175-43177 (3) -> EOF
43178-43180 (3) -> EOF
43181-43181 (1) -> 43191
43182-43184 (3) -> EOF
43185-43186 (2) -> EOF
43187-43188 (2) -> EOF
43189-43190 (2) -> EOF
43191-43191 (1) -> 43219
43192-43194 (3) -> EOF
43195-43196 (2) -> EOF
43197-43198 (2) -> EOF
43199-43203 (5) -> EOF
43204-43209 (6) -> EOF
43210-43211 (2) -> EOF
43212-43216 (5) -> EOF
43217-43218 (2) -> EOF
43219-43219 (1) -> 43236
43220-43225 (6) -> EOF
43226-43228 (3) -> EOF
43229-43231 (3) -> EOF
43232-43233 (2) -> EOF
43234-43234 (1) -> EOF
43235-43235 (1) -> EOF
43236-43236 (1) -> 43249
43237-43238 (2) -> EOF
43239-43241 (3) -> EOF
43242-43244 (3) -> EOF
43245-43246 (2) -> EOF
43247-43248 (2) -> EOF
43249-43249 (1) -> 43306
43250-43252 (3) -> EOF
43253-43255 (3) -> EOF
43256-43285 (30) -> EOF
43286-43289 (4) -> EOF
43290-43293 (4) -> EOF
43294-43302 (9) -> EOF
43303-43305 (3) -> EOF
43306-43306 (1) -> 43327
43307-43312 (6) -> EOF
43313-43314 (2) -> EOF
43315-43316 (2) -> EOF
43317-43317 (1) -> EOF
43318-43320 (3) -> EOF
43321-43326 (6) -> EOF
43327-43327 (1) -> 43360
43328-43333 (6) -> EOF
43334-43338 (5) -> EOF
43339-43342 (4) -> EOF
```

```
43343-43347 (5) -> EOF
43348-43353 (6) -> EOF
43354-43359 (6) -> EOF
43360-43360 (1) -> 43376
43361-43363 (3) -> EOF
43364-43366 (3) -> EOF
43367-43369 (3) -> EOF
43370-43372 (3) -> EOF
43373-43374 (2) -> EOF
43375-43375 (1) -> EOF
43376-43376 (1) -> 43393
43377-43378 (2) -> EOF
43379-43383 (5) -> EOF
43384-43384 (1) -> EOF
43385-43386 (2) -> EOF
43387-43388 (2) -> EOF
43389-43392 (4) -> EOF
43393-43393 (1) -> 43413
43394-43396 (3) -> EOF
43397-43400 (4) -> EOF
43401-43402 (2) -> EOF
43403-43406 (4) -> EOF
43407-43410 (4) -> EOF
43411-43412 (2) -> EOF
43413-43413 (1) -> 43427
43414-43414 (1) -> EOF
43415-43417 (3) -> EOF
43418-43419 (2) -> EOF
43420-43421 (2) -> EOF
43422-43423 (2) -> EOF
43424-43425 (2) -> EOF
43426-43426 (1) -> EOF
43427-43427 (1) -> 43464
43428-43431 (4) -> EOF
43432-43435 (4) -> EOF
43436-43440 (5) -> EOF
43441-43444 (4) -> EOF
43445-43448 (4) -> EOF
43449-43453 (5) -> EOF
43454-43463 (10) -> EOF
43464-43464 (1) -> 43487
43465-43466 (2) -> EOF
43467-43469 (3) -> EOF
43470-43475 (6) -> EOF
43476-43477 (2) -> EOF
43478-43480 (3) -> EOF
43481-43483 (3) -> EOF
43484-43486 (3) -> EOF
43487-43487 (1) -> EOF
43488-43488 (1) -> EOF
43489-43526 (38) -> EOF
```

7.5 Summary

The result of the test was that fatmapper should work as expected!

CHAPTER 8

CHANGES in fatmapper

8.1 Release 1.0 (in development)

8.1.1 (Incompatible) changes

- none

8.1.2 Features added

- none

8.1.3 Bugs fixed

- none

CHAPTER 9

fatmapper AUTHORS

fatmapper is written and maintained by Patrick Neumann <patrick@neumannsland.de>.

CHAPTER 10

LICENSE

10.1 Comments and Documentation

You are free to:

Share — copy and redistribute the material in any medium or format Adapt — remix, transform, and build upon the material

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial — You may not use the material for commercial purposes.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices:

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation. No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

Source: https://creativecommons.org/licenses/by-nc/4.0/legalcode

10.2 Software

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

See the LICENSE file for more details.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index