Hachoir Documentation

Release 3.0a3

Victor Stinner

Jul 16, 2017

Contents

Hachoir is a Python library to view and edit a binary stream field by field. In other words, Hachoir allows you to "browse" any binary stream just like you browse directories and files. A file is splitted in a tree of fields, where the smallest field is just one bit. Examples of fields types: integers, strings, bits, padding types, floats, etc. Hachoir is the French word for a meat grinder (meat mincer), which is used by butchers to divide meat into long tubes; Hachoir is used by computer butchers to divide binary files into fields.

- Hachoir3 website (this site)
- Hachoir3 at GitHub (source code, bugs)

Command line tools using Hachoir parsers:

- *hachoir-metadata*: get metadata from binary files
- *hachoir-urwid*: display the content of a binary file in text mode
- *hachoir-grep*: find a text pattern in a binary file
- *hachoir-strip*: modify a file to remove metadata

See also Hachoir at Bitbucket: original Hachoir for Python 2.

CHAPTER 1

User Guide

Install Hachoir3

To install Hachoir3, type:

```
python3 -m pip install -U hachoir3
```

For use hachoir-urwid, you will also need to install urwid library:

```
python3 -m pip install -U urwid
```

Hachoir3 requires Python 3.3 or newer.

hachoir-metadata program

The `hachoir-metadata` program is a tool to extract metadata from multimedia files: videos, pictures, sounds, archives, etc. It supports most common file formats:

- Archives: bzip2, gzip, zip, tar
- Audio: MPEG audio ("MP3"), WAV, Sun/NeXT audio, Ogg/Vorbis (OGG), MIDI, AIFF, AIFC, Real audio (RA)
- Image: BMP, CUR, EMF, ICO, GIF, JPEG, PCX, PNG, TGA, TIFF, WMF, XCF
- Misc: Torrent
- Program: EXE
- Video: ASF format (WMV video), AVI, Matroska (MKV), Quicktime (MOV), Ogg/Theora, Real media (RM)

Features:

- Gtk interface
- Plugins for Nautilus (Gnome) and Konqueror (KDE)

- Support invalid / truncated files
- Unicode compliant (charset ISO-8859-XX, UTF-8, UTF-16), convert string to your terminal charset
- Remove duplicate values and if a string is a substring of another, just keep the longest one
- Set priority to value, so it's possible to filter metadata (option –level)
- Only depends on [[hachoir-parser|hachoir-parser]] (and not on libmatroska, libmpeg2, libvorbis, etc.)

It tries to give as much information as possible. For some file formats, it gives more information than libextractor for example, such as the RIFF parser, which can extract creation date, software used to generate the file, etc. But hachoir-metadata cannot guess informations. The most complex operation is just to compute duration of a music using frame size and file size.

hachoir-metadata has three modes:

- classic mode: extract metadata, you can use –level=LEVEL to limit quantity of information to display (and not to extract)
- `--type`: show on one line the file format and most important informations
- `--mime`: just display file MIME type

The command `hachoir-metadata --mime` works like `file --mime`, and `hachoir-metadata --type` like `file`. But today file command supports more file formats then hachoir-metadata.

Example

Example on AVI video (RIFF file format):

```
$ hachoir-metadata pacte_des_gnous.avi
Common:
- Duration: 4 min 25 sec
- Comment: Has audio/video index (248.9 KB)
- MIME type: video/x-msvideo
- Endian: Little endian
Video stream:
- Image width: 600
- Image height: 480
- Bits/pixel: 24
- Compression: DivX v4 (fourcc:"divx")
- Frame rate: 30.0
Audio stream:
- Channel: stereo
- Sample rate: 22.1 KHz
- Compression: MPEG Layer 3
```

Supported file formats

Total: 33 file formats.

Archive

- bzip2: bzip2 archive
- cab: Microsoft Cabinet archive
- gzip: gzip archive

- mar: Microsoft Archive
- tar: TAR archive
- zip: ZIP archive

Audio

- aiff: Audio Interchange File Format (AIFF)
- mpeg_audio: MPEG audio version 1, 2, 2.5
- real_audio: Real audio (.ra)
- sun_next_snd: Sun/NeXT audio

Container

- matroska: Matroska multimedia container
- ogg: Ogg multimedia container
- real_media: !RealMedia (rm) Container File
- riff: Microsoft RIFF container

Image

- bmp: Microsoft bitmap (BMP) picture
- gif: GIF picture
- ico: Microsoft Windows icon or cursor
- jpeg: JPEG picture
- pcx: PC Paintbrush (PCX) picture
- png: Portable Network Graphics (PNG) picture
- psd: Photoshop (PSD) picture
- targa: Truevision Targa Graphic (TGA)
- tiff: TIFF picture
- wmf: Microsoft Windows Metafile (WMF)
- xcf: Gimp (XCF) picture

Misc

- ole2: Microsoft Office document
- pcf: X11 Portable Compiled Font (pcf)
- torrent: Torrent metainfo file
- ttf: !TrueType font

Program

- exe: Microsoft Windows Portable Executable

Video

- asf: Advanced Streaming Format (ASF), used for WMV (video) and WMA (audio)
- flv: Macromedia Flash video
- mov: Apple !QuickTime movie

Command line options

Modes –mime and –type

Option –mime ask to just display file MIME type (works like UNIX "file –mime" program):

```
$ hachoir-metadata --mime logo-Kubuntu.png sheep_on_drugs.mp3 wormux_32x32_16c.ico
logo-Kubuntu.png: image/png
sheep_on_drugs.mp3: audio/mpeg
wormux_32x32_16c.ico: image/x-ico
```

Option –file display short description of file type (works like UNIX "file" program):

```
$ hachoir-metadata --type logo-Kubuntu.png sheep_on_drugs.mp3 wormux_32x32_16c.ico
logo-Kubuntu.png: PNG picture: 331x90x8 (alpha layer)
sheep_on_drugs.mp3: MPEG v1 layer III, 128.0 Kbit/sec, 44.1 KHz, Joint stereo
wormux_32x32_16c.ico: Microsoft Windows icon: 16x16x32
```

Modes –mime and –type

Option `--mime` ask to just display file MIME type:

```
$ hachoir-metadata --mime logo-Kubuntu.png sheep_on_drugs.mp3 wormux_32x32_16c.ico
logo-Kubuntu.png: image/png
sheep_on_drugs.mp3: audio/mpeg
wormux_32x32_16c.ico: image/x-ico
```

(it works like UNIX "file –mime" program)

Option `--file` display short description of file type:

```
$ hachoir-metadata --type logo-Kubuntu.png sheep_on_drugs.mp3 wormux_32x32_16c.ico
logo-Kubuntu.png: PNG picture: 331x90x8 (alpha layer)
sheep_on_drugs.mp3: MPEG v1 layer III, 128.0 Kbit/sec, 44.1 KHz, Joint stereo
wormux_32x32_16c.ico: Microsoft Windows icon: 16x16x32
```

(it works like UNIX "file" program)

Filter metadatas with –level

hachoir-metadata is a too much verbose by default:

```
$ hachoir-metadata logo-Kubuntu.png
Image:
- Image width: 331
- Image height: 90
- Bits/pixel: 8
- Image format: Color index
- Creation date: 2006-05-26 09:41:46
- Compression: deflate
- MIME type: image/png
- Endian: Big endian
```

You can skip useless information (here, only until level 7):

```
$ hachoir-metadata --level=7 logo-Kubuntu.png
Image:
- Image width: 331
- Image height: 90
- Bits/pixel: 8
- Image format: Color index
- Creation date: 2006-05-26 09:41:46
- Compression: deflate
```

Example to get most importation informations (level 3):

```
$ hachoir-metadata --level=3 logo-Kubuntu.png
Image:
- Image width: 331
- Image height: 90
- Bits/pixel: 8
- Image format: Color index
```

Getting help: –help

Use `--help` option to get full option list.

See also

Used by

hachoir-metadata library is used by:

- Plone4artist
- amplee (implementation of the Atom Publishing Protocol, APP)
- django-massmedia (Calloway open source library)
- pyrenamer

Informations

- (fr) DCMI Metadata Terms: Classification of meta-datas done by the //Dublin Core//
- (fr) Dublin Core article on Openweb website
- (fr) avi_ogminfo : Informations about AVI and OGM files

- (en) Xesam (was Wasabi): common interface between programs extracting metadata

Libraries

- (fr|en) MediaInfo (GPL v2, C++)
- (en) Mutagen: audio metadata tag reader and writer (Python)
- (en) getid3: Library written in PHP to extact meta-datas from several multimedia file formats (and not only MP3)
- (fr|en) libextractor: Library dedicated to meta-data extraction. See also: (en) Bader's Python binding
- (en) Kaa (part of Freevo), it replaces mmpython (Media Metadata for Python) (dead project)
- (en) ExifTool: Perl library to read and write metadata

Programs

- jpeginfo
- ogginfo
- mkvinfo
- mp3info

Programs using metadata

- Programs using metadata:
  - GLScube
  - Beagle (Kerry)
  - Beagle++
  - Nepomuk
- Extractors:
  - Tracker
  - Strigi
- Other: Lucene (full text search)

Metadata examples

hachoir-metadata (version 0.10) output examples.

Video

AVI

Common:

- Duration: 1 hour 38 min 4 sec

- Image width: 576
- Image height: 240
- Frame rate: 25.0 fps
- Bit rate: 989.9 Kbit/sec
- Producer: Nandub v1.0rc2
- Comment: Has audio/video index (5.7 MB)
- MIME type: video/x-msvideo
- Endian: Little endian

Video stream:

- Duration: 1 hour 38 min 4 sec
- Image width: 576
- Image height: 240
- Bits/pixel: 24
- Compression: XviD MPEG-4 (fourcc:"xvid")
- Frame rate: 25.0 fps

Audio stream:

- Duration: 1 hour 38 min 4 sec
- Channel: stereo
- Sample rate: 44.1 kHz
- Compression: MPEG Layer 3
- Bit rate: 128.0 Kbit/sec

WMV

Common:

- Title:
- Author: Nippon Television Network Corporation[[NTV]
- Duration: 1 min 47 sec 258 ms
- Creation date: 2003-06-16 07:57:23.235000
- Copyright: [C]]Nippon Television Network Corporation[NTV] 2003
- Bit rate: 276.9 Kbit/sec (max)
- Comment: Is seekable
- MIME type: video/x-ms-wmv
- Endian: Little endian

Audio stream !#1:

- Channel: stereo
- Sample rate: 8.0 kHz

- Bits/sample: 16 bits
- Compression: Windows Media Audio V7 / V8 / V9
- Bit rate: 13.0 Kbit/sec

Video stream !#1:

- Image width: 200
- Image height: 150
- Bits/pixel: 24
- Compression: Windows Media Video V7
- Bit rate: 16.3 Kbit/sec

Video stream !#2:

- Image width: 200
- Image height: 150
- Bits/pixel: 24
- Compression: Windows Media Video V7
- Bit rate: 36.3 Kbit/sec

Video stream !#3:

- Image width: 200
- Image height: 150
- Bits/pixel: 24
- Compression: Windows Media Video V7
- Bit rate: 211.3 Kbit/sec

MKV

Common:

- Duration: 17 sec 844 ms
- Creation date: 2006-08-16 11:04:36
- Producer: mkvmerge v1.7.0 ('What Do You Take Me For') built on Jun 7 2006 08:33:28
- Producer: libebml v0.7.7 + libmatroska v0.8.0
- MIME type: video/x-matroska
- Endian: Big endian

Video stream:

- Language: French
- Image width: 384
- Image height: 288
- Compression: V_MPEG4/ISO/AVC

Audio stream:

- Title: travail = aliénation (extrait)
- Language: French
- Channel: mono
- Sample rate: 44.1 kHz
- Compression: A_VORBIS

FLV

Common:

- Duration: 46 sec 942 ms
- Bit rate: 287.4 Kbit/sec
- Producer: !YouTube, Inc.
- Producer: !YouTube Metadata Injector.
- Format version: Macromedia Flash video version 1
- MIME type: video/x-flv
- Endian: Big endian

Metadata:

- Channel: mono
- Sample rate: 22.1 kHz
- Bits/sample: 16 bits
- Compression: MPEG-2 layer III, 64.0 Kbit/sec, 22.1 kHz

Metadata:

- Compression: Sorensen H.263

Audio

MP3

Metadata:

- Title: 07. motorbike
- Author: Sheep On Drugs
- Album: Bilmusik vol 1. Stainless Steel Providers
- Duration: 1 sec 301 ms
- Music genre: Car music
- Track number: 7
- Track total: 13
- Channel: Joint stereo
- Sample rate: 44.1 kHz

- Bits/sample: 16 bits
- Compression rate: 11.0x
- Creation date: 2003
- Bit rate: 128.0 Kbit/sec (constant)
- Comment: Stainless Steel Provider is compilated to the car of Twinstar.
- Format version: MPEG version 1 layer III
- MIME type: audio/mpeg
- Endian: Big endian

Ogg Vorbis

Common:

- Title: La mouche
- Album: Dans le caillou
- Duration: 2 min 59 sec 893 ms
- Music genre: Chanson
- Track number: 6
- Artist: Karpatt
- Creation date: 2004
- Producer: Xiph.Org libVorbis I 20050304
- MIME type: audio/vorbis
- Endian: Little endian

Audio:

- Channel: stereo
- Sample rate: 44.1 kHz
- Compression: Vorbis
- Bit rate: 128.0 Kbit/sec
- Format version: Vorbis version 0

Picture

JPEG

Common:

- Image width: 2048
- Image height: 1536
- Image orientation: Horizontal (normal)
- Bits/pixel: 24

- Pixel format: YCbCr
- Compression rate: 15.5x
- Camera aperture: 3
- Camera focal: 2.8
- Camera exposure: 1/60.1
- Camera model: E3100
- Camera manufacturer: NIKON
- Compression: JPEG (Baseline)
- Producer: E3100v1.2
- Comment: JPEG quality: 85%
- Format version: JFIF 1.01
- MIME type: image/jpeg
- Endian: Big endian

PNG

Metadata:

- Image width: 331
- Image height: 90
- Bits/pixel: 32
- Pixel format: RGBA
- Compression rate: 12.0x
- Creation date: 2006-05-26 09:41:46
- Compression: deflate
- MIME type: image/png
- Endian: Big endian

ICO

Common:

- MIME type: image/x-ico
- Endian: Little endian

Icon !#1 (16x16):

- Image width: 16
- Image height: 16
- Bits/pixel: 32
- Compression rate: 0.9x

- Compression: Uncompressed (RGB)

Archive

CAB

Common:

- Compression: LZX (level 16)
- Comment: 1 folders, 6 files
- Format version: Microsoft Cabinet version 0x0103
- MIME type: application/vnd.ms-cab-compressed
- Endian: Little endian

File "fontinst.inf":

- File name: fontinst.inf
- File size: 64 bytes
- Creation date: 1998-11-10 16:09:52

File "Georgiaz.TTF":

- File name: Georgiaz.TTF
- File size: 155.1 KB
- File attributes: archive
- Creation date: 1998-11-10 14:00:02

File "Georgiab.TTF":

- File name: Georgiab.TTF
- File size: 136.3 KB
- File attributes: archive
- Creation date: 1998-11-10 14:00:02

Misc

TTF

Metadata:

- Title: !DejaVu Serif
- Author: !DejaVu fonts team
- Version: 2.7
- Creation date: 2006-07-06 17:29:52
- Last modification: 2006-07-06 17:29:52
- Copyright: Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved.nDejaVu changes are in public domain

- Copyright: !http://dejavu.sourceforge.net/wiki/index.php/License
- URL: !http://dejavu.sourceforge.net
- Comment: Smallest readable size in pixels: 8 pixels
- Comment: Font direction: Mixed directional
- MIME type: application/octet-stream
- Endian: Big endian

EXE (PE)

Metadata:

- Title: EULA
- Author: Dell Inc
- Version: 1.00
- Creation date: 2006-08-09 03:23:10
- Comment: CPU: Intel 80386
- Comment: Subsystem: Windows/GUI
- Format version: Portable Executable: Windows application
- MIME type: application/x-dosexec
- Endian: Little endian

Torrent

Metadata:

- File name: debian-31r4-i386-binary-1.iso
- File size: 638.7 MB
- Creation date: 2006-11-16 21:44:37
- URL: !http://bttracker.acc.umu.se:6969/announce
- Comment: “Debian CD from cdimage.debian.org”
- Comment: Piece length: 512.0 KB
- MIME type: application/x-bittorrent
- Endian: Little endian

hachoir-urwid program

hachoir-urwid is a binary file explorer based on Hachoir library to parse the files. Using this tool you can exactly know the meaning of each bit/byte of your files. With direction keys, you can navigate in the field tree. The key ‘h’ will disable ‘human display’ and switch to ‘raw display’. It’s sometime useful when you would like to compare hexadecimal data and Hachoir reprensentation.

hachoir-urwid is the most sexy user interface based on hachoir.parser to explore a binary file.

Command line options

- `--preload=10`: Load 10 fields when loading a new field set
- `--path="/header/bpp"`: Open the specified path and focus on the field
- `--parser=PARSERID`: Force a parser (and skip parser validation)

Usefull keys

Move:

- up/down: move up/down
- home: go to parent
- end: go to the last field of a field set
- left/right: horizontal scrolling

Setup display:

- h: most important option :-) switch between human display (default) and raw value
- v / d / s: show or hide field value / description / size
- a: switch between relative (default) and absolute address
- b: switch between address in decimal (default) and hexadecimal

Interaction:

- enter: on a field set, expand/collaspe the children
- space: parse a file/stream contained in the current field

Application:

- q: quit
- < / >: previous / next tab
- – / -: move separator vertically
- esc or CTRL+W: close current tab
- F1: display help

Help

Command line options: use –help option.

In hachoir-use, use F1 key to get help (keyboard keys).

hachoir-subfile program

hachoir-subfile is a tool based on hachoir-parser to find subfiles in any binary stream.

Changelog

Version 0.5.3 (2008-04-01):

- Catch StreamError on file copy
- Use "#!/usr/bin/env python" as shebang for FreeBSD

Version 0.5.2 (2007-07-13):

- Fix shebang: use "#!/usr/bin/python"
- Only import hachoir_core.profiler with –profiler command line option is used, so hachoir-subfile do not depends on 'profiler' Python module

Version 0.5.1 (2007-07-12):

- Fix setup.py: also install script 'hachoir-subfile'

Version 0.5 (2007-07-11):

- Publication of the first public version

Usage

Search JPEG images:

```
hachoir-subfile input --parser=jpeg
```

Search images:

```
hachoir-subfile input --category=image
```

Search images, videos and SWF files:

```
hachoir-subfile input --category=image,video --parser=swf
```

Search all subfiles and store them in `/tmp/subfiles/`:

```
hachoir-subfile input /tmp/subfiles/
```

Other options:

- –offset: start search at specified offset in bytes
- –size: limit search to specified size in bytes

Search speed is proportional to the number of used parsers.

How does it work?

Find file start

To find file start, hachoir-subfile use "magic number": short string (2 to 16 bytes) typical to a file format. Examples:

- "MZ" for MS-Dos (and Windows) executable
- "xFFxD8xFF" for JPEG
- "FAT16 " for FAT16 file system

When a magic number is found, a parser of hachoir parser is opened. The validate() method is used to make sure that the file is in the right file format. Some values of the header are tested:

- TAR archive: check magic number, check first file entry (user/group identifier, file size)
- SWF animation: check magic number, check version, check rectangle padding value
- etc.

Find file length

To find (guess) file length, each parser requires a method called "createContentSize()". Examples:

- RIFF container: read "/filesize" field value
- JPEG picture: search "xFFxD9" (End of image chunk) string
- etc.

Examples

Find files in a hard drive:

```
$ hachoir-subfile /dev/sda --size=34200100 --quiet
[+] Start search (32.6 MB)

[+] Found file at 0: MS-DOS hard drive with Master Boot Record (MBR)
[+] Found file at 32256: FAT16 filesystem
[+] Found file at 346112 size=308280 (301.1 KB): Microsoft Bitmap version 3
[+] Found file at 32157696: MS-DOS executable
[+] Found file at 32483328: MS-DOS executable
[+] Found file at 32800768: MS-DOS executable
[+] Found file at 32851968: MS-DOS executable
[+] Found file at 32872448: MS-DOS executable
[+] Found file at 33058816: MS-DOS executable
[+] Found file at 33112064: MS-DOS executable
[+] Found file at 33142784: MS-DOS executable
[+] Found file at 33949936: Microsoft Windows Portable Executable: Intel 80386 or␣
↪greater

[+] Search done -- offset=34200100 (32.6 MB)
Total time: 20.08 sec -- 1.6 MB/sec
```

PowerPoint document:

```
$ hachoir-subfile chiens.PPS
[+] Start search (828.5 KB)

[+] Found file at 0: Microsoft Office document
[+] Found file at 537 size=28449 (27.8 KB): JPEG picture: 433x300 pixels
[+] Found file at 29011 size=34761 (33.9 KB): JPEG picture: 433x300 pixels
[+] Found file at 63797 size=40326 (39.4 KB): JPEG picture: 433x300 pixels
[+] Found file at 104148 size=30641 (29.9 KB): JPEG picture: 433x300 pixels
[+] Found file at 134814 size=22782 (22.2 KB): JPEG picture: 384x325 pixels
[+] Found file at 157621 size=24744 (24.2 KB): JPEG picture: 443x313 pixels
[+] Found file at 182390 size=27241 (26.6 KB): JPEG picture: 443x290 pixels
[+] Found file at 209656 size=27407 (26.8 KB): JPEG picture: 443x336 pixels
[+] Found file at 237088 size=30088 (29.4 KB): JPEG picture: 388x336 pixels
```

```
[+] Found file at 267201 size=30239 (29.5 KB): JPEG picture: 366x336 pixels
[+] Found file at 297465 size=81634 (79.7 KB): JPEG picture: 630x472 pixels
[+] Found file at 379124 size=36142 (35.3 KB): JPEG picture: 599x432 pixels
[+] Found file at 415291 size=28801 (28.1 KB): JPEG picture: 443x303 pixels
[+] Found file at 444117 size=28283 (27.6 KB): JPEG picture: 433x300 pixels
[+] Found file at 472425 size=95913 (93.7 KB): PNG picture: 433x431x8
[+] Found file at 568363 size=219252 (214.1 KB): PNG picture: 532x390x8
[+] Found file at 811308 size=20644 (20.2 KB): Microsoft Windows Metafile (WMF)␣
↪picture

[+] Search done -- offset=848384 (828.5 KB)
Total time: 1.30 sec -- 635.1 KB/sec
```

Filter

It's possible to filter files using your own function. Example to skip images smaller than 256x256:

```
from hachoir_metadata import extractMetadata
import sys

def metadataFilter(parser):
    try:
        metadata = extractMetadata(parser)
    except HachoirError, err:
        metadata = None

    if metadata:
        if hasattr(metadata, "width") and metadata.width[0]] < 256:
            print("Skip picture with width < 256 pixels", file=sys.stderr)
            return False
        if hasattr(metadata, "height") and metadata.height[[0]|< 256:
            print("Skip picture with height < 256 pixels", file=sys.stderr)
            return False
    return True

subfile = HachoirSubfile(...)
subfile.filter = metadataFilter
```

hachoir-grep program

hachoir-grep is an experimantal search program based on Hachoir. Its goal is to search a substring in a binary file or list all strings.

Examples

List all strings:

```
$ .hachoir-grep --all sheep_on_drugs.mp3
0:ID3
10:TAL
17:Bilmusik vol 1. Stainless Steel Providers
(...)
```

```
143:COM
150:eng
154:Stainless Steel Provider is compilated to the car of Twinstar.
```

Search substring:

```
$ hachoir-grep "il" sheep_on_drugs.mp3
17:Bilmusik vol 1. Stainless Steel Providers
154:Stainless Steel Provider is compilated to the car of Twinstar.
```

Search substring with case sensitive and display string path:

```
$ hachoir-grep --path --case Car sheep_on_drugs.mp3
78:/id3v2/field[2]/content/text:Car music
```

Other options:

- `--no-value`: Don't display the string value
- `--no-addr`: Don't display string address
- `--percent`: Display search percent (on stderr)
- Get full option list using –help

Comparaison to grep and strings

The difference between hachoir-grep and grep or strings are:

- hachoir-grep don't parse non-string data
- hachoir-grep cares about string charset (ISO-8859-1, UTF-8, UTF-16, etc.) but search is charset independant
- hachoir-grep can display string address

Drawbacks:

- hachoir-grep is slow and might use a lot of memory
- results depend on the quality of Hachoir parsers

hachoir-strip program

hachoir-strip is an experimental program based on Hachoir library: it removes "useless" informations from a file. Don't use it to create smaller file, you have better to recompress your data :-) hachoir-strip can be used if you would like to remove spy information which can be used to know the origin of a file.

Examples

Our victim:

```
$ hachoir-metadata KDE_Click.wav.new
Common:
- Creation date: 2001-02-21   <== here they are
- Producer: Sound Forge 4.5   <== spy informations :-)
- MIME type: audio/x-wav
```

```
- Endian: Little endian
Audio:
- Duration: 39 ms
...
```

Clean up the file:

```
$ hachoir-strip KDE_Click.wav
[+] Process file KDE_Click.wav
Remove field /info
Remove 56 bytes (3.1%)
Save new file into KDE_Click.wav.new

$ hachoir-metadata KDE_Click.wav.new
Common:
- MIME type: audio/x-wav
- Endian: Little endian
Audio:
- Duration: 39 ms
...
```

So hachoir-strip removed creation date (2001-02-21) and producer (software used to record/edit the sound: Sound Forge 4.5). The file is also 56 bytes smaller.

Options

You can select field types to remove using –strip:

- (default): remove all useless fields
- `--strip=useless`: remove really useless fields (eg. padding)
- `--strip=metadata`: remove metadata like ID3 tags and EXIF and IPTC metadatas
- `--strip=index`: remove video index

You can combine options with comma: `--strip="useless,metadata"`.

CHAPTER 2

Developer Guide

Hachoir3 for developers

Download code and run tests

To clone Git repository, type:

```
git clone https://github.com/haypo/hachoir3
```

Enter hachoir3 directory:

```
cd hachoir3
```

To run tests, type tox:

```
tox
```

If tox is not already installed, install tox:

```
python3 -m pip install tox
```

See also Hachoir3 on the Travis CI.

Why using Hachoir parsers?

Why using slow Python code instead of fast hardcoded C code? Hachoir has many interesting features:

- Autofix: Hachoir is able to open invalid / truncated files
- Lazy: Open a file is very fast since no information is read from file, data are read and/or computed when the user ask for it
- Types: Hachoir has many predefined field types (integer, bit, string, etc.) and supports string with charset (ISO-8859-1, UTF-8, UTF-16, ...)

- Addresses and sizes are stored in bit, so flags are stored as classic fields
- Endian: You have to set endian once, and then number are converted in the right endian
- Editor: Using Hachoir representation of data, you can edit, insert, remove data and then save in a new file.

Hachoir Metadata Example

hachoir-metadata example:

```
from hachoir.parser import createParser
from hachoir.metadata import extractMetadata
from sys import argv, stderr, exit

if len(argv) != 2:
    print("usage: %s filename" % argv[0], file=stderr)
    exit(1)
filename = argv[1]
parser = createParser(filename)
if not parser:
    print("Unable to parse file", file=stderr)
    exit(1)

with parser:
    try:
        metadata = extractMetadata(parser)
    except Exception as err:
        print("Metadata extraction error: %s" % err)
        metadata = None
if not metadata:
    print("Unable to extract metadata")
    exit(1)

for line in metadata.exportPlaintext():
    print(line)
```

hachoir.stream: Stream manipulation

To split data we first need is to get data :-) So this section presents the "hachoir.stream" API.

In most cases we work on files using the FileInputStream function. This function takes one argument: a Unicode filename. But for practical reasons we will use StringInputStream function in this documentation:

```
>>> data = b"point\0\3\0\2\0"
>>> from hachoir.stream import StringInputStream, LITTLE_ENDIAN
>>> stream = StringInputStream(data)
>>> stream.source
'<string>'
>>> len(data), stream.size
(10, 80)
>>> data[1:6], stream.readBytes(8, 5)
(b'oint\x00', b'oint\x00')
>>> data[6:8], stream.readBits(6*8, 16, LITTLE_ENDIAN)
(b'\x03\x00', 3)
>>> data[8:10], stream.readBits(8*8, 16, LITTLE_ENDIAN)
(b'\x02\x00', 2)
```

First big difference between a string and a Hachoir stream is that sizes and addresses are written in bits and not bytes. The difference is a factor of eight, that's why we write "6*8" to get the sixth byte for example. You don't need to know anything else to use Hachoir, so let's play with fields!

hachoir.field: Field manipulation

Basic parser

We will parse the data used in the last section:

```
>>> from hachoir.field import Parser, CString, UInt16
>>> class Point(Parser):
...     endian = LITTLE_ENDIAN
...     def createFields(self):
...         yield CString(self, "name", "Point name")
...         yield UInt16(self, "x", "X coordinate")
...         yield UInt16(self, "y", "Y coordinate")
...
>>> point = Point(stream)
>>> for field in point:
...     print("%s) %s=%s" % (field.address, field.name, field.display))
...
0) name="point"
48) x=3
64) y=2
```

point is a the root of our field tree. This tree is really simple, it just has one level and three fields: name, x and y. Hachoir stores a lot of information in each field. In this example we just show the address, name and display attributes. But a field has more attributes:

```
>>> x = point["x"]
>>> "%s = %s" % (x.path, x.value)
'/x = 3'
>>> x.parent == point
True
>>> x.description
'X coordinate'
>>> x.index
1
>>> x.address, x.absolute_address
(48, 48)
```

The index is not the index of a field in a parent field list, '1' means that it's the second since the index starts at zero.

Parser with sub-field sets

After learning basic API, let's see a more complex parser: parser with sub-field sets:

```
>>> from hachoir.field import FieldSet, UInt8, Character, String
>>> class Entry(FieldSet):
...     def createFields(self):
...         yield Character(self, "letter")
...         yield UInt8(self, "code")
...
>>> class MyFormat(Parser):
```

```
...     endian = LITTLE_ENDIAN
...     def createFields(self):
...         yield String(self, "signature", 3, charset="ASCII")
...         yield UInt8(self, "count")
...         for index in range(self["count"].value):
...             yield Entry(self, "point[]")
...
>>> data = b"MYF\3a\0b\2c\0"
>>> stream = StringInputStream(data)
>>> root = MyFormat(stream)
```

This example presents many interesting features of Hachoir. First of all, you can see that you can have two or more levels of fields. Here we have a tree with two levels:

```
>>> def displayTree(parent):
...     for field in parent:
...         print(field.path)
...         if field.is_field_set: displayTree(field)
...
>>> displayTree(root)
/signature
/count
/point[0]
/point[0]/letter
/point[0]/code
/point[1]
/point[1]/letter
/point[1]/code
/point[2]
/point[2]/letter
/point[2]/code
```

A field set is also a field, so it has the same attributes than another field (name, address, size, path, etc.) but has some new attributes like stream or root.

Lazy feature

Hachoir is written in Python so it should be slow and eat a lot of CPU and memory, and it does. But in most cases, you don't need to explore an entire field set and read all values; you just need to read some values of some specific fields. Hachoir is really lazy: no field is parsed before you ask for it, no value is read from stream before you read a value, etc. To inspect this behaviour, you can watch "current_length" (number of read fields) and "current_size" (current size in bits of a field set):

```
>>> root = MyFormat(stream)  # Rebuild our parser
>>> (root.current_length, root.current_size)
(0, 0)
>>> print(root["signature"].display)
"MYF"
>>> (root.current_length, root.current_size, root["signature"].size)
(1, 24, 24)
```

Just after its creation, a parser is empty (0 fields). When we read the first field, its size becomes the size of the first field. Some operations requires to read more fields:

```
>>> print(root["point[0]/letter"].display)
'a'
```

```
>>> (root.current_length, root.current_size)
(3, 48)
```

Reading point[0] needs to read field "count". So root now contains three fields.

List of field types

Number:

- Bit: one bit (True/False) ;
- Bits: unsigned number with a size in bits ;
- Bytes: vector of know bytes (e.g. file signature) ;
- UInt8, UInt16, UInt24, UInt32, UInt64: unsigned number (size: 8, 16, ... bits) ;
- Int8, Int16, Int24, Int32, Int64: signed number (size: 8, 16, ... bits) ;
- Float32, Float64, Float80: IEEE 754 floating point number (32, 64, 80 bits) ;

Text:

- Character: 8 bits ASCII character ;
- String: fixed length string ;
- CString: string ending with nul byte ("\0") ;
- UnixLine: string ending with new line character ("\n") ;
- PascalString8, PascalString16 and PascalString32: string prefixed with length in a unsigned 8 / 16 / 32 bits integer (use parent endian) ;

Timestamp (date and time):

- TimestampUnix32, TimestampUnix64: 32/64 bits UNIX, number of seconds since the January 1st 1970 ;
- TimestampMac32: 32-bit Mac, number of seconds since the January 1st 1904 ;
- TimestampWin64: 64-bit Windows, number of 1/10 microseconds since the January 1st 1600 ;
- DateTimeMSDOS32 and TimeDateMSDOS32: 32-bit MS-DOS structure, since the January 1st 1980.

Timedelta (duration):

- TimedeltaWin64: 64-bit Windows, number of 1/10 microseconds

Padding and raw bytes:

- PaddingBits/PaddingBytes: padding with a size in bits/bytes ;
- NullBits/NullBytes: null padding with a size in bits/bytes ;
- RawBits/RawBytes: unknown content with a size in bits/bytes.
- SubFile: a file contained in the stream ;

To create your own type, you can use:

- GenericInteger: integer ;
- GenericString: string ;
- FieldSet: Set of other fields ;
- Parser: The main class to parse a stream.

Field class

Read only attributes:

- name (str): Name of the field, is unique in parent field set
- address (long): address in bits relative to parent address
- absolute_address (long): address in bits relative to input stream
- parent (GenericFieldSet): parent field (is a field set)
- is_field_set (bool) <~~~ can be replaced: the field contains other fields?
- index (int): index of the field in parent field set (first index is 0)

Read only and lazy attributes:

- size (long), cached: size of the field in bits
- description (strlunicode), cached: informal description
- display (unicode): value with human representation as unicode string
- raw_display (unicode): value with raw representation as unicode string
- path (str): concatenation with slash separator of all field name from the root field

Method that can be replaced:

- createDescription(): create value of 'description' attribute
- createValue(): create value of 'value' attribute
- createDisplay(): create value of 'display' attribute
- _createInputStream(): create an InputStream containing the field content

Aliases (method):

- __str__() <=> read display attribute
- __unicode__() <=> read display attribute
- __getitem__(key): alias to getField(key, False)

Other methods:

- static_size: helper to compute field size. If the value is an integer, the type has constant size. If it's a function, the size depends of the arguments.
- hasValue(): check if the field has a value or not (default: self.value is not None)
- getField(key, const=True): get the field with specified key, if const is True the field set will not be changed
- __contains__(key)
- getSubIStream(): return a tagged InputStream containing the field content
- setSubIStream(): helper to replace _createInputStream (the old one is passed to the new one to allow chaining)

Field set class

Read only attributes:

- endian: value is BIG_ENDIAN or LITTLE_ENDIAN, the way the bits are written in input stream <~~ can be replaced

- stream (InputStream): input stream
- root (FieldSet): root of all fields
- eof (bool): End Of File: are we at the end of the input stream?
- done (bool): The parser is done or not?

Read only and lazy attributes:

- current_size (long): Current size in bits
- current_length (long): Current number of children

Methods:

- connectEvent(event, handler, local=True): connect an handler to an event
- raiseEvent(event, *args): raise an event
- reset(): clear all caches but keep its size if we know it
- setUniqueFieldName(): for field with name ending with "[]", replaces "[]" with an unique identifier like, "item[]" => "item[0]".
- seekBit(address, ...): create a field to seek to specified address or returns None if we are already there
- seekByte(address, ...): create a field to seek to specified address or returns None if we are already there
- replaceField(name, fields): replace a field with one or more fields <~~~ I don't like this method :-(
- getFieldByAddress(address, feed=True): get the field at the specified address
- writeFieldsIn(old, address, new): helper for replaceField() <~~~ can be an helper?
- getFieldType(): get the field type as a short string. The type may contains extra informations like the string charset.

Lazy methods:

- array(): create a FakeArray to easily get a field by its index (see FakeArray API to get more details)
- __len__(): number of children in the field set
- readFirstFields(number): read first 'number' fields, returns number of new fields
- readMoreFields(number): read more 'number' fields, returns number of new fields
- __iter__(): iterate over children
- createFields(): main function of the parser, create the fields. Don't call this function directly.

Hachoir internals

When is a field really created?

A field is created when someone ask to access it, or when another field is asked and the field is before it. So if you use a field in your field set constructor, one or more fields will be created. Example:

```
>>> from hachoir.field import Parser, Int8
>>> from hachoir.core.endian import BIG_ENDIAN
>>> class Point(Parser):
...     endian = BIG_ENDIAN
...     def __init__(self, stream):
```

```
...         Parser.__init__(self, stream)
...         if self["color"].value == -1:
...             self._description += " (no color)"
...
...     def createFields(self):
...         yield Int8(self, "color", "Point color (-1 for none)")
...         yield Int8(self, "use_3d", "Does it use Z axis?")
...         yield Int8(self, "x", "X axis value")
...         yield Int8(self, "y", "Y axis value")
...         if self["use_3d"] == 1:
...             yield Int8(self, "z", "Z axis value")
...
```

In the constructor, the field "color" is asked. So the field list will contains one field (color):

```
>>> from hachoir.stream import StringInputStream
>>> stream = StringInputStream(b"\x2A\x00\x04\x05")
>>> p = Point(stream)
>>> p.current_length
1
```

If you access another field, the field list will grow up until the requested field is reached:

```
>>> x = p["x"].value
>>> p.current_length
3
```

Some field set methods which create new fields:

- `__getitem__()`: feed field list until requested field is reached (or raise MissingField exception) ;
- `__len__()`: create all fields ;
- `__iter__()`: begin to iterate in existing fields, and the iterate in new fields until all fields are created ;
- `__contains__()`: feed field list until requested field is reached, may create all fields if the field doesn't exist ;
- `readFirstFields()` ;
- `readMoreFields()`.

When is the size really computed?

The size attribute also interact with field list creation, but it's mechanism is little bit more complex. By default, the whole field list have to be built before size value can be read. But you can specify field list size:

- if field list is not dynamic (e.g. doesn't depend on flag), use class attribute `static_size` ;
- otherwise you can set _size instance attribute in the constructor.

Two examples:

```
>>> from hachoir.field import Parser
>>> class FourBytes(Parser):
...     endian = BIG_ENDIAN
...     static_size = 32
...     def createFields(self):
...         yield Integer(self, "four", "uint32")
...
```

```
>>> class DynamicSize(Parser):
...     endian = BIG_ENDIAN
...     def __init__(self, stream, nb_items):
...         Parser.__init__(self, stream)
...         assert 0 < nb_items
...         self.nb_items = nb_items
...         self._size = nb_items * 32   # 32 is the size of one item
...
...     def createFields(self):
...         for index in range(self.nb_items):
...             yield Integer(self, "item[]", "uint32")
...
>>> a = FourBytes(stream)
>>> b = DynamicSize(stream, 1)
>>> a.size, b.size
(32, 32)
>>> # Check that nothing is really read from stream
... a.current_length, b.current_length
(0, 0)
```

When the value of a field is read?

When a field is created, the value of the field doesn't exist (equals to None). The value is really read when you read the field value using 'value' or 'display' field attributes. The value is then stored in the field.

Details about field name

The name of a field have to be unique in a field set because it is used as key in the field list. The argument 'name' of the Field constructor can be changed in the constructor, but should not (and cannot) be changed after that.

For arrays, you can use the 'magic' suffix « [] » (e.g. "item[]") which will be replaced by « [index] » where the number index is a counter starting a zero.

Endian

The "endian" is the way in which ''bytes" are stored. There are two important orders:

- « Big endian » in which *most* significant byte (*big* number) are written first (PowerPC / Motorola CPUs). It's also the network byte order ;
- « Little endian » in which *least* significant byte (*little* number) are written first (Intel x86 CPUs).

The number 0x1020 is stored "0x10 0x20" in big endian and "0x20 0x10" in little endian.

The endian is global to a FieldSet and is a class attribute. Allowed values:

- BIG_ENDIAN ;
- NETWORK_ENDIAN (alias of BIG_ENDIAN) ;
- LITTLE_ENDIAN.

Example to set endian:

```
>>> from hachoir.core.endian import LITTLE_ENDIAN
>>> class UseLittleEndian(Parser):
...     endian = LITTLE_ENDIAN
...
```

For sub-field sets, if endian is not specified, parent endian will be used.

Explore a field set using it's path

Fields are stored in a tree. To explore the tree you have different tools:

- attribute *root* of a field which go to tree root ;
- attribute *parent* go to field parent (is None for tree root) ;
- and you can specify a path in *__getitem__()* argument.

There are different valid syntaxes for a path:

- path to a child of current node: `field["content"]` ;
- path to a child of the parent: `field["../brother"]` ;
- path from the root: `field["/header/key"]`.

hachoir.parser module

`hachoir.parser` is a package of most common file format parsers written for Hachoir framework. Not all parsers are complete, some are very good and other are poor: only parser first level of the tree for example.

A perfect parser have no "raw" field: with a perfect parser you are able to know *each* bit meaning. Some good (but not perfect ;-)) parsers:

- Matroska video
- Microsoft RIFF (AVI video, WAV audio, CDA file)
- PNG picture
- TAR and ZIP archive

Parser list

Archive

- 7zip: Compressed archive in 7z format
- ace: ACE archive
- bom_store: Apple bill-of-materials file
- bzip2: bzip2 archive
- cab: Microsoft Cabinet archive
- gzip: gzip archive
- mar: Microsoft Archive
- mozilla_ar: Mozilla Archive

- prs_pak: Parallel Realities Starfighter .pak archive
- rar: Roshal archive (RAR)
- rpm: RPM package
- tar: TAR archive
- unix_archive: Unix archive
- zip: ZIP archive
- zlib: ZLIB Data

Audio

- aiff: Audio Interchange File Format (AIFF)
- fasttracker2: FastTracker2 module
- flac: FLAC audio
- itunesdb: iPod iTunesDB file
- midi: MIDI audio
- mod: Uncompressed amiga module
- mpeg_audio: MPEG audio version 1, 2, 2.5
- ptm: PolyTracker module (v1.17)
- real_audio: Real audio (.ra)
- s3m: ScreamTracker3 module
- sun_next_snd: Sun/NeXT audio

Container

- asn1: Abstract Syntax Notation One (ASN.1)
- matroska: Matroska multimedia container
- ogg: Ogg multimedia container
- ogg_stream: Ogg logical stream
- real_media: RealMedia (rm) Container File
- riff: Microsoft RIFF container
- swf: Macromedia Flash data

File System

- ext2: EXT2/EXT3 file system
- fat12: FAT12 filesystem
- fat16: FAT16 filesystem
- fat32: FAT32 filesystem
- iso9660: ISO 9660 file system

- linux_swap: Linux swap file
- msdos_harddrive: MS-DOS hard drive with Master Boot Record (MBR)
- ntfs: NTFS file system
- reiserfs: ReiserFS file system

Game

- blp1: Blizzard Image Format, version 1
- blp2: Blizzard Image Format, version 2
- lucasarts_font: LucasArts Font
- spiderman_video: The Amazing Spider-Man vs. The Kingpin (Sega CD) FMV video
- zsnes: ZSNES Save State File (only version 143)

Image

- bmp: Microsoft bitmap (BMP) picture
- gif: GIF picture
- ico: Microsoft Windows icon or cursor
- jpeg: JPEG picture
- pcx: PC Paintbrush (PCX) picture
- png: Portable Network Graphics (PNG) picture
- psd: Photoshop (PSD) picture
- targa: Truevision Targa Graphic (TGA)
- tiff: TIFF picture
- wmf: Microsoft Windows Metafile (WMF)
- xcf: Gimp (XCF) picture

Misc

- 3do: renderdroid 3d model.
- 3ds: 3D Studio Max model
- bplist: Apple/NeXT Binary Property List
- chm: Microsoft's HTML Help (.chm)
- dsstore: Mac OS X DS_Store
- gnomekeyring: Gnome keyring
- hlp: Microsoft Windows Help (HLP)
- lnk: Windows Shortcut (.lnk)
- mapsforge_map: Mapsforge map file
- mstask: .job 'at' file parser from ms windows

- ole2: Microsoft Office document
- pcf: X11 Portable Compiled Font (pcf)
- pdf: Portable Document Format (PDF) document
- tcpdump: Tcpdump file (network)
- torrent: Torrent metainfo file
- ttf: TrueType font

Program

- elf: ELF Unix/BSD program/library
- exe: Microsoft Windows Portable Executable
- java_class: Compiled Java class
- java_serialized: Serialized Java object
- macho: Mach-O program/library
- macho_fat: Mach-O fat program/library
- nds_file: Nintendo DS game file
- pifv: EFI Platform Initialization Firmware Volume
- prc: Palm Resource File
- python: Compiled Python script (.pyc/.pyo files)

Video

- asf: Advanced Streaming Format (ASF), used for WMV (video) and WMA (audio)
- flv: Macromedia Flash video
- mov: Apple QuickTime movie
- mpeg_ts: MPEG-2 Transport Stream
- mpeg_video: MPEG video, version 1 or 2

Total: 91 parsers

hachoir.regex module

`hachoir.regex` is a Python library for regular expression (regex or regexp) manupulation. You can use a|b (or) and a+b (and) operators. Expressions are optimized during the construction: merge ranges, simplify repetitions, etc. It also contains a class for pattern matching allowing to search multiple strings and regex at the same time.

Regex examples

Regex are optimized during their creation:

```
>>> from hachoir.regex import parse, createRange, createString
>>> createString("bike") + createString("motor")
<RegexString 'bikemotor'>
>>> parse('(foo|fooo|foot|football)')
<RegexAnd 'foo(|[ot]|tball)'>
```

Create character range:

```
>>> regex = createString("1") | createString("3")
>>> regex
<RegexRange '[13]'>
>>> regex |= createRange("2", "4")
>>> regex
<RegexRange '[1-4]'>
```

As you can see, you can use classic "a|b" (or) and "a+b" (and) Python operators. Example of regular expressions using repetition:

```
>>> parse("(a{2,}){3,4}")
<RegexRepeat 'a{6,}'>
>>> parse("(a*|b)*")
<RegexRepeat '[ab]*'>
>>> parse("(a*|b|){4,5}")
<RegexRepeat '(a+|b){0,5}'>
```

Compute minimum/maximum matched pattern:

```
>>> r=parse('(cat|horse)')
>>> r.minLength(), r.maxLength()
(3, 5)
>>> r=parse('(a{2,}|b+)')
>>> r.minLength(), r.maxLength()
(1, None)
```

Pattern maching

Use PatternMaching if you would like to find many strings or regex in a string. Use addString() and addRegex() to add your patterns:

```
>>> from hachoir.regex import PatternMatching
>>> p = PatternMatching()
>>> p.addString("a")
>>> p.addString("b")
>>> p.addRegex("[cd]")
```

And then use search() to find all patterns:

```
>>> for start, end, item in p.search("a b c d"):
...     print("%s..%s: %s" % (start, end, item))
...
0..1: a
2..3: b
4..5: [cd]
6..7: [cd]
```

You can also attach an objet to a pattern with 'user' (user data) argument:

```
>>> p = PatternMatching()
>>> p.addString("un", 1)
>>> p.addString("deux", 2)
>>> for start, end, item in p.search("un deux"):
...     print("%r at %s: user=%r" % (item, start, item.user))
...
<StringPattern 'un'> at 0: user=1
<StringPattern 'deux'> at 3: user=2
```

Create regular expressions

There is two ways to create regular expressions: use string or directly use the API.

Atom classes:

- RegexEmpty: empty regex (match nothing)
- RegexStart, RegexEnd, RegexDot: symbols ^, $ and .
- RegexString
- RegexRange: character range like [a-z] or [^0-9]
- RegexAnd
- RegexOr
- RegexRepeat

All classes are based on Regex class.

Create regex with string

```
>>> from hachoir.regex import parse
>>> parse('')
<RegexEmpty ''>
>>> parse('abc')
<RegexString 'abc'>
>>> parse('[bc]d')
<RegexAnd '[bc]d'>
>>> parse('a(b|[cd]|(e|f))g')
<RegexAnd 'a[b-f]g'>
>>> parse('([a-z]|[b-])')
<RegexRange '[a-z-]'>
>>> parse('^^..$$')
<RegexAnd '^..$'>
>>> parse('chats?')
<RegexAnd 'chats?'>
>>> parse(' +abc')
<RegexAnd ' +abc'>
```

Create regex with the API

```
>>> from hachoir.regex import createString, createRange
>>> createString('')
<RegexEmpty ''>
```

```
>>> createString('abc')
<RegexString 'abc'>
>>> createRange('a', 'b', 'c')
<RegexRange '[a-c]'>
>>> createRange('a', 'b', 'c', exclude=True)
<RegexRange '[^a-c]'>
```

Manipulate regular expressions

Convert to string:

```
>>> from hachoir.regex import createRange, createString
>>> str(createString('abc'))
'abc'
>>> repr(createString('abc'))
"<RegexString 'abc'>"
```

Operatiors “and” and “or”:

```
>>> createString("bike") & createString("motor")
<RegexString 'bikemotor'>
>>> createString("bike") | createString("motor")
<RegexOr '(bike|motor)'>
```

You can also use operator “+”, it’s just an alias to a & b:

```
>>> createString("big ") + createString("bike")
<RegexString 'big bike'>
```

Compute minimum/maximum matched pattern:

```
>>> r=parse('(cat|horse)')
>>> r.minLength(), r.maxLength()
(3, 5)
```

Optimizations

The library includes many optimization to keep small and fast expressions.

Group prefix:

```
>>> createString("blue") | createString("brown")
<RegexAnd 'b(lue|rown)'>
>>> createString("moto") | parse("mot.")
<RegexAnd 'mot.'>
>>> parse("(ma|mb|mc)")
<RegexAnd 'm[a-c]'>
>>> parse("(maa|mbb|mcc)")
<RegexAnd 'm(aa|bb|cc)'>
```

Merge ranges:

```
>>> from hachoir.regex import createRange
>>> regex = createString("1") | createString("3"); regex
<RegexRange '[13]'>
```

```
>>> regex = regex | createRange("2"); regex
<RegexRange '[1-3]'>
>>> regex = regex | createString("0"); regex
<RegexRange '[0-3]'>
>>> regex = regex | createRange("5", "6"); regex
<RegexRange '[0-356]'>
>>> regex = regex | createRange("4"); regex
<RegexRange '[0-6]'>
```

PatternMaching class

Use PatternMaching if you would like to find many strings or regex in a string. Use addString() and addRegex() to add your patterns:

```
>>> from hachoir.regex import PatternMatching
>>> p = PatternMatching()
>>> p.addString("a")
>>> p.addString("b")
>>> p.addRegex("[cd]")
```

And then use search() to find all patterns:

```
>>> for start, end, item in p.search("a b c d"):
...     print("%s..%s: %s" % (start, end, item))
...
0..1: a
2..3: b
4..5: [cd]
6..7: [cd]
```

Item is a Pattern object, not the matched string. To be exact, it's a StringPattern for string and a RegexPattern for regex. You can associate an "user" value to each Pattern object:

```
>>> p2 = PatternMatching()
>>> p2.addString("un", 1)
>>> p2.addString("deux", 2)
>>> p2.addRegex("(trois|three)", 3)
>>> for start, end, item in p2.search("un deux trois"):
...     print("%r at %s: user=%r" % (item, start, item.user))
...
<StringPattern 'un'> at 0: user=1
<StringPattern 'deux'> at 3: user=2
<RegexPattern 't(rois|hree)'> at 8: user=3
```

You can associate any Python object to an item, not only an integer!

hachoir.editor module

Hachoir editor is a Python library based on Hachoir core used to edit binary files.

Today, only one program uses it: *hachoir-strip* (remove "useless" information to make a file smaller).

Example: gzip, remove filename

```
from hachoir.parser import createParser
from hachoir.editor import createEditor
from hachoir.field import writeIntoFile

parser = createParser("file.gz")
with parser:
    editor = createEditor(parser)
    del editor["filename"]
    editor["has_filename"].value = False
    writeIntoFile(editor, "noname.gz")
```

Example: gzip, add extra

```
from hachoir.parser import createParser
from hachoir.editor import createEditor
from hachoir.field import writeIntoFile
from hachoir.editor import EditableInteger, EditableBytes

parser = createParser("file.gz")
with parser:
    editor = createEditor(parser)
    extra = "abcd"
    editor["has_extra"].value = True
    editor.insertAfter("os",
                       EditableInteger(editor, "extra_length", False,
                                       16, len(extra)),
                       EditableBytes(editor, "extra", extra))
    writeIntoFile(editor, "file_extra.gz")
```

Example: zip, set comment

```
from hachoir.parser import createParser
from hachoir.editor import createEditor
from hachoir.field import writeIntoFile

parser = createParser("file.zip")
with parser:
    editor = createEditor(parser)
    editor["end_central_directory/comment"].value = "new comment"
    writeIntoFile(editor, "file_comment.zip")
```

CHAPTER 3

Others pages

Contact

Mailing list

Mailing list: `hachoir AT lists.tuxfamily.org`

- Read archives on mail-archives.com
- Mailing list archives on Gmane.org: Read mailing list using HTTP, NNTP or RSS
- To subscribe, send an email with subject `subscribe` (and empty body) to `hachoir-request@lists.tux(...).org`
- To unsubscribe, send an email with subject `unsubscribe` (and empty body) to `hachoir-request@lists.tux(...).org`
- You have to subscribe to post email.
- Created October 22nd 2006

Hack Hachoir

Run tests

Using tox

Install tox (`pip install tox`) and then run tox:

```
tox
```

Manually

Run tests manually:

```
python3 runtests.py
```

Hachoir Authors

Team:

- Julien Muchembled <jm AT jm10 DOT no-ip DOT com>
- Victor Stinner aka haypo <victor DOT stinner AT gmail DOT com>
- Robert Xiao aka nneonneo <nneonneo AT gmail.com> - improve SeekableFieldSet

Packagers:

- Arnaud Pithon aka bildoon <apithon AT free DOT fr> - ArchLinux package (v0.5.2 and svn)
- Emmanuel Garette aka GnunuX <gnunux AT laposte DOT net> - ArchLinux package (v0.5.2)
- Michael Scherer aka misc <misc AT mandriva DOT org> - Mandriva package (v0.5.2)
- Michel Casabona aka plumbear <michel DOT casabona AT free DOT fr> - Debian package (v0.5.2)
- Richard Demongeot <richard AT demongeot DOT biz> - Debian package (v0.5.2)
- Thomas de Grenier de Latour TGL <degrenier AT easyconnect DOT fr> - Gentoo ebuild

Contributors

- Alexandre Boeglin <alex AT boeglin DOT org> - PIFV parser
- Aurélien Jacobs <aurel AT gnuage DOT org> - AVI parser big contributor
- Christophe Fergeau <teuf AT gnome.org> - Improve iTunesDB parser
- Christophe Gisquet <christophe.gisquet AT free.fr> - Write RAR parser
- Cyril Zorin <cyril.zorin AT gmail.com> - Author of 3DO parser
- Elie Roudninski aka adema <liliroud AT hotmail.com> - Started Gtk GUI
- Feth Arezki <feth AT arezki.net> - Fix hachoir-metadata-qt to save the current directory
- Frédéric Weisbecker <chantecode AT gmail.com> - Author of ReiserFS parser
- Gottfried Ganßauge <ganssauge AT gmx DOT de> - Fix Win32 curses issues
- Jason Gorski <jgorski AT gmail.com> - Author of zsnes parser
- Jean-Marc Libs <jeanmarc.libs AT gmail.com> - KDE plugin
- Mickaël Kenikssi <k DOT mickael AT gmail DOT com> - Write CDDA parser
- Mike Melanson <mike AT multimedia.cx> - Write RealMedia and SpiderMan video parser
- Olivier Schwab <le.poulpe303 AT laposte.net> - Write 7-zip parser
- Pierre Thierry <nowhere.man AT levallois.eu.org> - KDE plugin
- Robert Xiao aka nneonneo <nneonneo AT gmail.com> - improve LNK parser
- Romain Hérault <romain DOT herault AT gmail DOT com> - Author of iTunesDB parser
- Sebastien Ponce <sebastien.ponce AT cern.ch> - ActionScript (for SWF) parser

- Thomas Pabst <thomas.pabst AT gmail.com> - Gnome plugin
- Thomas de Grenier de Latour aka TGL <degrenier AT easyconnect DOT fr> - Java class parser

Changelog

hachoir 3.0a3

Parsers:

- Fix ELF parser (on Python 3)

hachoir 3.0a2 (2017-02-24)

Parsers:

- Add initial parser for Mapsforge .map files (only version 3 supported)
- Add parser for PAK files from "Project: Starfighter" game
- Add OS X `.bom` parser
- mov parser: add `traf` entry
- Update iTunesDB Parser
- 7zip: Improve and expand 7zip parser
- tar: Support ustar prefix field in tar archives.
- Enhance the Java class parser.
- Added more field to exif extractor
- Add WIP Mach-O parser
- ntfs improvements: parse non-resident runlists
- ext2: support ext4, massive parser improvements

  Huge changeset that may break backwards compatibility, for better consistency and deeper parsing. Basic rundown:

  - Support many more (new) flags added in ext4 and beyond
  - Create nice option displays for flags
  - Improve handling of groups using SeekableFieldSet
  - Parse (demarcate) inode data blocks
  - Consistently use lower case for flag names
- Enhance mpeg_ts parser to support MTS/M2TS; add MIME type

New features:

- Python parser supports Python 3.3-3.7 .pyc files.
- metadata: get comment from ZIP
- Support InputIOStream.read(0)

- Add a close() method and support for the context manager protocol (`with obj: ...`) to parsers, input and output streams.
- Add more file extensions for PE program parser.
- ZIP: add MIME type for Android APK, `.apk` file.
- Add editable field for TimestampMac32 type

Bugfixes:

- Issue #2: Fix saving a filed into a file in urwid
  - FileFromInputStream: fix comparison between None and an int
  - InputIOStream: open the file in binary mode
- Fix OutputStream.writeBits() (was broken since the migration to Python 3)
- Fix ResourceWarning warnings on files: use the new close() methods and context managers.
- Fix a few pending warnings on StopIteration.
- Fixup and relocate hachoir-wx, which now works mostly properly.
- Fix hachoir-parser matroska SimpleBlock
- Fix Mac timestamp name

Remove the unmaintained experimental HTTP interface.

hachoir 3.0a1 (2017-01-09)

Changes:

- metadata: support TIFF picture

Big refactoring:

- First release of the Python 3 port
- The 7 Hachoir subprojects (core, editor, metadata, parser, regex, subfile, urwid) which lived in different directories are merged again into one big unique Python 3 module: hachoir. For example, "hachoir_parser" becomes "hachoir.parser".
- The project moved from Bitbucket (Mercurial repository) to GitHub (Git repository). The Mercurial history since 2007 was kept.
- Reorganize tests into a new tests/ subdirectory. Copy test files directly into the Git repository, instead of relying on an old FTP server which is not convenient. For example, it's now possible to add the required test file in a Git commit. So it's more convenient for pull requests as well.
- Port code to Python 3: "for field in parser: yield field" becomes "yield from parser".
- Fix PEP 8 issues: most of the code does now respect the latest PEP 8 coding style.
- Enable Travis CI on the project: tests are on Python 3.5, check also pep8 and documentation.
- Copy old wiki pages and documentation splitted into many subdirectories into a single consistent Sphinx documentation in the doc/ subdirectory. Publish the documentation online at http://hachoir3.readthedocs.io/