RarFile Documentation

Release 3.0

Marko Kreen

May 17, 2017

Contents

1 rarfile API documentation 3

2 rarfile FAQ 13

3 rarfile history 15

4 Indices and tables 21

Python Module Index 23

This is Python module for RAR archive reading. The interface is made as zipfile like as possible. Licensed under ISC license.

Features:

- Supports both RAR3 and RAR5 format archives.
- Supports multi volume archives.
- Supports Unicode filenames.
- Supports password-protected archives.
- Supports archive and file comments.
- Archive parsing and non-compressed files are handled in pure Python code.
- Compressed files are extracted by executing external tool: either `unrar` from RARLAB or `bsdtar` from libarchive.
- Works with both Python 2.7 and 3.x.

Documentation:

CHAPTER 1

rarfile API documentation

Table Of Contents

- *rarfile API documentation*
  - *Introduction*
  - *RarFile class*
  - *RarInfo class*
  - *RarExtFile class*
  - *Functions*
  - *Module Configuration*
  - *Constants*
  - *Exceptions*

Introduction

RAR archive reader.

This is Python module for Rar archive reading. The interface is made as `zipfile`-like as possible.

Basic logic:

- Parse archive structure with Python.
- Extract non-compressed files with Python
- Extract compressed files with unrar.
- Optionally write compressed data to temp file to speed up unrar, otherwise it needs to scan whole archive on each execution.

Example:

```
import rarfile

rf = rarfile.RarFile('myarchive.rar')
for f in rf.infolist():
    print f.filename, f.file_size
    if f.filename == 'README':
        print(rf.read(f))
```

Archive files can also be accessed via file-like object returned by *RarFile.open()*:

```
import rarfile

with rarfile.RarFile('archive.rar') as rf:
    with rf.open('README') as f:
        for ln in f:
            print(ln.strip())
```

There are few module-level parameters to tune behaviour, here they are with defaults, and reason to change it:

```
import rarfile

# Set to full path of unrar.exe if it is not in PATH
rarfile.UNRAR_TOOL = "unrar"

# Set to '\\' to be more compatible with old rarfile
rarfile.PATH_SEP = '/'
```

For more details, refer to source.

RarFile class

class rarfile.**RarFile**(*rarfile*, *mode='r'*, *charset=None*, *info_callback=None*, *crc_check=True*, *errors='stop'*)

Bases: object

Parse RAR structure, provide access to files in archive.

Open and parse a RAR archive.

Parameters

- **rarfile** – archive file name
- **mode** – only 'r' is supported.
- **charset** – fallback charset to use, if filenames are not already Unicode-enabled.
- **info_callback** – debug callback, gets to see all archive entries.
- **crc_check** – set to False to disable CRC checks
- **errors** – Either "stop" to quietly stop parsing on errors, or "strict" to raise errors. Default is "stop".

comment = None

Archive comment. Unicode string or None.

setpassword(*password*)
Sets the password to use when extracting.

needs_password()
Returns True if any archive entries require password for extraction.

namelist()
Return list of filenames in archive.

infolist()
Return RarInfo objects for all files/directories in archive.

volumelist()
Returns filenames of archive volumes.

In case of single-volume archive, the list contains just the name of main archive file.

getinfo(*fname*)
Return RarInfo for file.

open(*fname*, *mode='r'*, *psw=None*)
Returns file-like object (*RarExtFile*) from where the data can be read.

The object implements `io.RawIOBase` interface, so it can be further wrapped with `io.BufferedReader` and `io.TextIOWrapper`.

On older Python where io module is not available, it implements only .read(), .seek(), .tell() and .close() methods.

The object is seekable, although the seeking is fast only on uncompressed files, on compressed files the seeking is implemented by reading ahead and/or restarting the decompression.

Parameters

- **fname** – file name or RarInfo instance.
- **mode** – must be 'r'
- **psw** – password to use for extracting.

read(*fname*, *psw=None*)
Return uncompressed data for archive entry.

For longer files using *RarFile.open()* may be better idea.

Parameters

- **fname** – filename or RarInfo instance
- **psw** – password to use for extracting.

close()
Release open resources.

printdir()
Print archive file list to stdout.

extract(*member*, *path=None*, *pwd=None*)
Extract single file into current directory.

Parameters

- **member** – filename or *RarInfo* instance
- **path** – optional destination path
- **pwd** – optional password to use

extractall(*path=None*, *members=None*, *pwd=None*)
Extract all files into current directory.

Parameters

- **path** – optional destination path
- **members** – optional filename or *RarInfo* instance list to extract
- **pwd** – optional password to use

testrar()
Let 'unrar' test the archive.

strerror()
Return error string if parsing failed or None if no problems.

RarInfo class

class rarfile.**RarInfo**
Bases: object

An entry in rar archive.

RAR3 extended timestamps are datetime.datetime objects without timezone. RAR5 extended timestamps are datetime.datetime objects with UTC timezone.

filename
File name with relative path. Path separator is '/'. Always unicode string.

date_time
File modification timestamp. As tuple of (year, month, day, hour, minute, second). RAR5 allows archives where it is missing, it's None then.

file_size
Uncompressed size.

compress_size
Compressed size.

compress_type
Compression method: one of *RAR_M0* .. *RAR_M5* constants.

extract_version
Minimal Rar version needed for decompressing. As (major*10 + minor), so 2.9 is 29.

RAR3: 10, 20, 29

RAR5 does not have such field in archive, it's simply set to 50.

host_os
Host OS type, one of RAR_OS_* constants.

RAR3: *RAR_OS_WIN32*, *RAR_OS_UNIX*, *RAR_OS_MSDOS*, *RAR_OS_OS2*, *RAR_OS_BEOS*.

RAR5: *RAR_OS_WIN32*, *RAR_OS_UNIX*.

mode
File attributes. May be either dos-style or unix-style, depending on host_os.

mtime
File modification time. Same value as *date_time* but as datetime.datetime object with extended precision.

ctime
Optional time field: creation time. As `datetime.datetime` object.

atime
Optional time field: last access time. As `datetime.datetime` object.

arctime
Optional time field: archival time. As `datetime.datetime` object. (RAR3-only)

CRC
CRC-32 of uncompressed file, unsigned int.

RAR5: may be None.

blake2sp_hash
Blake2SP hash over decompressed data. (RAR5-only)

comment
Optional file comment field. Unicode string. (RAR3-only)

file_redir
If not None, file is link of some sort. Contains tuple of (type, flags, target). (RAR5-only)

Type is one of constants:

RAR5_XREDIR_UNIX_SYMLINK unix symlink to target.

RAR5_XREDIR_WINDOWS_SYMLINK windows symlink to target.

RAR5_XREDIR_WINDOWS_JUNCTION windows junction.

RAR5_XREDIR_HARD_LINK hard link to target.

RAR5_XREDIR_FILE_COPY current file is copy of another archive entry.

Flags may contain `RAR5_XREDIR_ISDIR` bit.

volume
Volume nr, starting from 0.

volume_file
Volume file name, where file starts.

isdir()
Returns True if entry is a directory.

needs_password()
Returns True if data is stored password-protected.

RarExtFile class

class `rarfile.`**RarExtFile**(*parser*, *inf*)
Bases: `io.RawIOBase`

Base class for file-like object that *`RarFile.open()`* returns.

Provides public methods and common crc checking.

Behaviour:

- no short reads - .read() and .readinfo() read as much as requested.
- no internal buffer, use io.BufferedReader for that.

Open archive entry.

name = None
Filename of the archive entry

read(*cnt=None*)
Read all or specified amount of data from archive entry.

close()
Close open resources.

readinto(*buf*)
Zero-copy read directly into buffer.

Returns bytes read.

tell()
Return current reading position in uncompressed data.

seek(*ofs*, *whence=0*)
Seek in data.

On uncompressed files, the seeking works by actual seeks so it's fast. On compresses files its slow - forward seeking happends by reading ahead, backwards by re-opening and decompressing from the start.

readable()
Returns True

writable()
Returns False.

Writing is not supported.

seekable()
Returns True.

Seeking is supported, although it's slow on compressed files.

readall()
Read all remaining data

fileno()
Returns underlying file descriptor if one exists.

An IOError is raised if the IO object does not use a file descriptor.

flush()
Flush write buffers, if applicable.

This is not implemented for read-only and non-blocking streams.

isatty()
Return whether this is an 'interactive' stream.

Return False if it can't be determined.

next

readline()
Read and return a line from the stream.

If limit is specified, at most limit bytes will be read.

The line terminator is always b'n' for binary files; for text files, the newlines argument to open can be used to select the line terminator(s) recognized.

readlines()
Return a list of lines from the stream.

hint can be specified to control the number of lines read: no more lines will be read if the total size (in bytes/characters) of all lines so far exceeds hint.

truncate()
Truncate file to size bytes.

File pointer is left unchanged. Size defaults to the current IO position as reported by tell(). Returns the new size.

Functions

rarfile.**is_rarfile**(*xfile*)
Check quickly whether file is rar archive.

Module Configuration

rarfile.**UNRAR_TOOL = 'unrar'**
'unrar', 'rar' or full path to either one

rarfile.**DEFAULT_CHARSET = 'windows-1252'**
default fallback charset

rarfile.**TRY_ENCODINGS = ('utf8', 'utf-16le')**
list of encodings to try, with fallback to DEFAULT_CHARSET if none succeed

rarfile.**PATH_SEP = '/'**
Separator for path name components. RAR internally uses '\'. Use '/' to be similar with zipfile.

rarfile.**USE_EXTRACT_HACK = 1**
whether to speed up decompression by using tmp archive

rarfile.**HACK_SIZE_LIMIT = 20971520**
limit the filesize for tmp archive usage

Constants

rarfile.**RAR_M0**
No compression.

rarfile.**RAR_M1**
Compression level *-m1* - Fastest compression.

rarfile.**RAR_M2**
Compression level *-m2*.

rarfile.**RAR_M3**
Compression level *-m3*.

rarfile.**RAR_M4**
Compression level *-m4*.

rarfile.**RAR_M5**
Compression level *-m5* - Maximum compression.

rarfile.**RAR_OS_MSDOS**

rarfile.**RAR_OS_OS2**

rarfile.**RAR_OS_WIN32**

rarfile.**RAR_OS_UNIX**

rarfile.**RAR_OS_MACOS**

rarfile.**RAR_OS_BEOS**

Exceptions

class rarfile.**Error**
Bases: exceptions.Exception

Base class for rarfile errors.

class rarfile.**BadRarFile**
Bases: *rarfile.Error*

Incorrect data in archive.

class rarfile.**NotRarFile**
Bases: *rarfile.Error*

The file is not RAR archive.

class rarfile.**BadRarName**
Bases: *rarfile.Error*

Cannot guess multipart name components.

class rarfile.**NoRarEntry**
Bases: *rarfile.Error*

File not found in RAR

class rarfile.**PasswordRequired**
Bases: *rarfile.Error*

File requires password

class rarfile.**NeedFirstVolume**
Bases: *rarfile.Error*

Need to start from first volume.

class rarfile.**NoCrypto**
Bases: *rarfile.Error*

Cannot parse encrypted headers - no crypto available.

class rarfile.**RarExecError**
Bases: *rarfile.Error*

Problem reported by unrar/rar.

class rarfile.**RarWarning**
Bases: *rarfile.RarExecError*

Non-fatal error

class rarfile.**RarFatalError**
Bases: *rarfile.RarExecError*

Fatal error

class rarfile.**RarCRCError**
Bases: *rarfile.RarExecError*

CRC error during unpacking

class rarfile.**RarLockedArchiveError**
Bases: *rarfile.RarExecError*

Must not modify locked archive

class rarfile.**RarWriteError**
Bases: *rarfile.RarExecError*

Write error

class rarfile.**RarOpenError**
Bases: *rarfile.RarExecError*

Open error

class rarfile.**RarUserError**
Bases: *rarfile.RarExecError*

User error

class rarfile.**RarMemoryError**
Bases: *rarfile.RarExecError*

Memory error

class rarfile.**RarCreateError**
Bases: *rarfile.RarExecError*

Create error

class rarfile.**RarNoFilesError**
Bases: *rarfile.RarExecError*

No files that match pattern were found

class rarfile.**RarUserBreak**
Bases: *rarfile.RarExecError*

User stop

class rarfile.**RarWrongPassword**
Bases: *rarfile.RarExecError*

Incorrect password

class rarfile.**RarUnknownError**
Bases: *rarfile.RarExecError*

Unknown exit code

class rarfile.**RarSignalExit**
Bases: *rarfile.RarExecError*

Unrar exited with signal

class rarfile.**RarCannotExec**
Bases: *rarfile.RarExecError*

Executable not found.

CHAPTER 2

rarfile FAQ

Table of Contents

- *rarfile FAQ*
  - *What are the dependencies?*
  - *Does it parse `unrar` output to get archive contents?*
  - *Will rarfile support wrapping unrarlib/unrar.dll/unrar.so in the future?*
  - *How can I get it work on Windows?*
  - *How to avoid the need for user to manually install rarfile/unrar?*
  - *Will it support creating RAR archives?*
  - *What is the USE_EXTRACT_HACK?*

What are the dependencies?

It depends on `unrar` command-line utility to do the actual decompression. Note that by default it expect it to be in `PATH`. If unrar launching fails, you need to fix this.

Alternatively, *`rarfile`* can use bsdtar from libarchive as decompression backend, but that is a bit problematic as bsdtar does not support all RAR features.

It depends on cryptography or PyCrypto modules to process archives with password-protected headers.

Does it parse `unrar` output to get archive contents?

No, *`rarfile`* parses RAR structure in Python code. Also it can read uncompressed files from archive without external utility.

Will rarfile support wrapping unrarlib/unrar.dll/unrar.so in the future?

No. The current architecture - parsing in Python and decompression with command line tools work well across all interesting operating systems (Windows/Linux/MacOS), wrapping a library does not bring any advantages.

Simple execution of command-line tools is also legally simpler situation than linking with external library.

How can I get it work on Windows?

On Windows the `unrar.exe` is not in `PATH` so simple `Popen("unrar ..")` does not work. It can be solved several ways:

1. Add location of `unrar.exe` to PATH.
2. Set *`rarfile.UNRAR_TOOL`* to full path of `unrar.exe`.
3. Copy `unrar.exe` to your program directory.
4. Copy `unrar.exe` to system directory that is in PATH, eg. `C:\Windows`.

How to avoid the need for user to manually install rarfile/unrar?

Include `rarfile.py` and/or `unrar` with your application.

Will it support creating RAR archives?

No. RARLAB is not interested in RAR becoming open format and specifically discourages writing RAR creation software.

In the meantime use either Zip (better compatibility) or 7z (better compression) format for your own archives.

What is the USE_EXTRACT_HACK?

RarFile uses `unrar` to extract compressed files. But when extracting single file from archive containing many entries, `unrar` needs to parse whole archive until it finds the right entry. This makes random-access to entries slow. To avoid that, RarFile remembers location of compressed data for each entry and on read it copies it to temporary archive containing only data for that one file, thus making `unrar` fast.

The logic is only activated for entries smaller than *`rarfile.HACK_SIZE_LIMIT`* (20M by default). Bigger files are accessed directly from RAR.

Note - it only works for non-solid archives. So if you care about random access to files in your archive, do not create solid archives.

CHAPTER 3

rarfile history

Version 3.0 (2016-12-27)

New feature:

- Support RAR5 archive format. It is actually completely different archive format from RAR3 one, only is uses same file extension and tools are old one.

  Except incompatibilies noted below, most of code should notice no change, existing *RarInfo* fields will continue using RAR3-compatible values (eg. *RarInfo.host_os*). RAR5-specific values will use new fields.

Incompatibilities between rarfile v2.x and 3.x:

- Default *PATH_SEP* is now '/' instead '\'.
- Removed NEED_COMMENTS option, comments are always extracted.
- Removed UNICODE_COMMENTS option, they are always decoded.
- Removed USE_DATETIME option, *RarInfo.date_time* is always tuple, *RarInfo.mtime*, *RarInfo.atime*, *RarInfo.ctime* and *RarInfo.arctime* are always datetime.datetime objects.

Fixes:

- Fixed bug when calling rarfp.open() on a RarInfo structure.

Cleanups:

- Code refactor to allow 2 different file format parsers.
- Code cleanups to pass modern linters.
- New testing and linting setup based on Tox.
- Use setuptools instead distutils for install.

Version 2.8 (2016-06-07)

- Fix: support solid archives from in-memory file object. Full archive will be written out to temp file. [#21]
- Fix: ask unrar stop switches scanning, to handle archive names starting with "-". (Alexander Shadchin) [#12]
- Fix: add missing _parse_error variable to RarFile object. (Gregory Mazzola) [#20]
- Fix: return proper boolean from *RarInfo.needs_password()*. [#22]
- Fix: do not insert non-string rarfile into exception string. (Tim Muller) [#23]
- Fix: make *RarFile.extract()* and *RarFile.testrar()* support in-memory archives.
- Use cryptography module as preferred crypto backend. PyCrypto will be used as fallback.
- Cleanup: remove compat code for Python 2.4/2.5/2.6.

Version 2.7 (2014-11-23)

- Allow use of bsdtar as decompression backend. It sits on top of libarchive, which has support for reading RAR archives.

  Limitations of `libarchive` RAR backend:

  - Does not support solid archives.
  - Does not support password-protected archives.
  - Does not support "parsing filters" used for audio/image/executable data, so few non-solid, non-encrypted archives also fail.

  Now *rarfile* checks if `unrar` and if not then tries `bsdtar`. If that works, then keeps using it. If not then configuration stays with `unrar` which will then appear in error messages.
- Both *RarFile* and *is_rarfile()* now accept file-like object. Eg. `io.BytesIO`. Only requirement is that the object must be seekable. This mirrors similar funtionality in zipfile.

  Based on patch by Chase Zhang.
- Uniform error handling. *RarFile* accepts `errors="strict"` argument.

  Allow user to tune whether parsing and missing file errors will raise exception. If error is not raised, the error string can be queried with *RarFile.strerror()* method.

Version 2.6 (2013-04-10)

- Add context manager support for *RarFile* class. Both *RarFile* and *RarExtFile* support `with` statement now. (Wentao Han)
- *RarFile.volumelist()* method, returns filenames of archive volumes.
- Re-throw clearer error in case `unrar` is not found in `PATH`.
- Sync new unrar4.x error code from `rar.txt`.
- Use Sphinx for documentation, push docs to rtfd.org

Version 2.5 (2012-01-19)

Fixes:

- *RarExtFile.read()* and *RarExtFile.readinto()* now do looping read to work properly on short reads. Important for Python 3.2+ where read from pipe can return short result even on blocking file descriptor.
- Proper error reporting in *RarFile.extract()*, *RarFile.extractall()* and *RarFile.testrar()*.
- *RarExtFile.read()* from unrar pipe: prefer to return unrar error code, if thats not available, do own error checks.
- Avoid string addition in *RarExtFile.read()*, instead use always list+join to merge multi-part reads.
- dumprar: dont re-encode byte strings (Python 2.x). This avoids unneccessary failure when printing invalid unicode.

Version 2.4 (2011-11-05)

Fixes:

- `USE_DATETIME`: survive bad values from RAR
- Fix bug in corrupt unicode filename handling
- dumprar: make unicode chars work with both pipe and console

Version 2.3 (2011-07-03)

Features:

- Support .seek() method on file streams. (Kristian Larsson)
- Support .readinto() method on file streams. Optimized implementation is available on Python 2.6+ where `memoryview` is available.
- Support file comments - *RarInfo.comment* contains decompressed data if available.
- File objects returned by *RarFile.open()* are `io.RawIOBase`-compatible. They can further wrapped with `io.BufferedReader` and `io.TextIOWrapper`.
- Now .getinfo() uses dict lookup instead of sequential scan when searching archive entry. This speeds up processing for archives that have many entries.
- Option `UNICODE_COMMENTS` to decode both archive and file comments to unicode. It uses *TRY_ENCODINGS* for list of encodings to try. If off, comments are left as byte strings. Default: 0
- Option *PATH_SEP* to change path separator. Default: `r'\'`, set `rarfile.PATH_SEP='/'` to be compatibe with zipfile.
- Option `USE_DATETIME` to convert timestamps to datetime objects. Default: 0, timestamps are tuples.
- Option *TRY_ENCODINGS* to allow tuning attempted encoding list.
- Reorder *RarInfo* fiels to better show zipfile-compatible fields.
- Standard regtests to make sure various features work

Compatibility:

- Drop `RarInfo.unicode_filename`, plain *`RarInfo.filename`* is already unicode since 2.0.
- .read(-1) reads now until EOF. Previously it returned empty buffer.

Fixes:

- Make encrypted headers work with Python 3.x bytes() and with old 2.x 'sha' module.
- Simplify `subprocess.Popen` usage when launching `unrar`. Previously it tried to optimize and work around OS/Python bugs, but this is not maintainable.
- Use temp rar file hack on multi-volume archives too.
- Always .wait() on unrar, to avoid zombies
- Convert struct.error to BadRarFile
- Plug some fd leaks. Affected: Jython, PyPy.
- Broken archives are handled more robustly.

Version 2.2 (2010-08-19)

Fixes:

- Relaxed volume naming. Now it just calculates new volume name by finding number in old one and increasing it, without any expectations what that number should be.
- Files with 4G of compressed data in one colume were handled wrong. Fix.
- DOS timestamp seconds need to be multiplied with 2.
- Correct EXTTIME parsing.

Cleanups:

- Compressed size is per-volume, sum them together, so that user sees complete compressed size for files split over several volumes.
- dumprar: Show unknown bits.
- Use `struct.Struct` to cache unpack formats.
- Support missing `os.devnull`. (Python 2.3)

Version 2.1 (2010-07-31)

Features:

- Minimal implmentation for *`RarFile.extract()`*, *`RarFile.extractall()`*, *`RarFile.testrar()`*. They are simple shortcuts to `unrar` invocation.
- Accept *`RarInfo`* object where filename is expected.
- Include `dumprar.py` in .tgz. It can be used to visualize RAR structure and test module.
- Support for encrypted file headers.

Fixes:

- Don't read past ENDARC, there could be non-RAR data there.
- RAR 2.x: It does not write ENDARC, but our volume code expected it. Fix that.

- RAR 2.x: Support more than 200 old-style volumes.

Cleanups:

- Load comment only when requested.
- Cleanup of internal config variables. They should have now final names.
- *RarFile.open()*: Add mode=r argument to match zipfile.
- Doc and comments cleanup, minimize duplication.
- Common wrappers for both compressed and uncompressed files, now *RarFile.open()* also does CRC-checking.

Version 2.0 (2010-04-29)

Features:

- Python 3 support. Still works with 2.x.
- Parses extended time fields. (.mtime, .ctime, .atime)
- *RarFile.open()* method. This makes possible to process large entries that do not fit into memory.
- Supports password-protected archives.
- Supports archive comments.

Cleanups:

- Uses `subprocess` module to launch `unrar`.
- .filename is always Unicode string, .unicode_filename is now deprecated.
- .CRC is unsigned again, as python3 crc32() is unsigned.

Version 1.1 (2008-08-31)

Fixes:

- Replace `os.tempnam()` with `tempfile.mkstemp()`. (Jason Moiron)
- Fix infinite loop in _extract_hack on unexpected EOF
- *RarInfo.CRC* is now signed value to match crc32()
- *RarFile.read()* now checks file crc

Cleanups:

- more docstrings
- throw proper exceptions (subclasses of *rarfile.Error*)
- RarInfo has fields pre-initialized, so they appear in help()
- rename RarInfo.data to RarInfo.header_data
- dont use “print” when header parsing fails
- use try/finally to delete temp rar

Version 1.0 (2005-08-08)

- First release.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

rarfile, 3

Index

A

arctime (rarfile.RarInfo attribute), 7
atime (rarfile.RarInfo attribute), 7

B

BadRarFile (class in rarfile), 10
BadRarName (class in rarfile), 10
blake2sp_hash (rarfile.RarInfo attribute), 7

C

close() (rarfile.RarExtFile method), 8
close() (rarfile.RarFile method), 5
comment (rarfile.RarFile attribute), 4
comment (rarfile.RarInfo attribute), 7
compress_size (rarfile.RarInfo attribute), 6
compress_type (rarfile.RarInfo attribute), 6
CRC (rarfile.RarInfo attribute), 7
ctime (rarfile.RarInfo attribute), 6

D

date_time (rarfile.RarInfo attribute), 6
DEFAULT_CHARSET (in module rarfile), 9

E

Error (class in rarfile), 10
extract() (rarfile.RarFile method), 5
extract_version (rarfile.RarInfo attribute), 6
extractall() (rarfile.RarFile method), 5

F

file_redir (rarfile.RarInfo attribute), 7
file_size (rarfile.RarInfo attribute), 6
filename (rarfile.RarInfo attribute), 6
fileno() (rarfile.RarExtFile method), 8
flush() (rarfile.RarExtFile method), 8

G

getinfo() (rarfile.RarFile method), 5

H

HACK_SIZE_LIMIT (in module rarfile), 9
host_os (rarfile.RarInfo attribute), 6

I

infolist() (rarfile.RarFile method), 5
is_rarfile() (in module rarfile), 9
isatty() (rarfile.RarExtFile method), 8
isdir() (rarfile.RarInfo method), 7

M

mode (rarfile.RarInfo attribute), 6
mtime (rarfile.RarInfo attribute), 6

N

name (rarfile.RarExtFile attribute), 8
namelist() (rarfile.RarFile method), 5
NeedFirstVolume (class in rarfile), 10
needs_password() (rarfile.RarFile method), 5
needs_password() (rarfile.RarInfo method), 7
next (rarfile.RarExtFile attribute), 8
NoCrypto (class in rarfile), 10
NoRarEntry (class in rarfile), 10
NotRarFile (class in rarfile), 10

O

open() (rarfile.RarFile method), 5

P

PasswordRequired (class in rarfile), 10
PATH_SEP (in module rarfile), 9
printdir() (rarfile.RarFile method), 5

R

RAR_M0 (in module rarfile), 9
RAR_M1 (in module rarfile), 9
RAR_M2 (in module rarfile), 9
RAR_M3 (in module rarfile), 9
RAR_M4 (in module rarfile), 9

RAR_M5 (in module rarfile), 9
RAR_OS_BEOS (in module rarfile), 10
RAR_OS_MACOS (in module rarfile), 10
RAR_OS_MSDOS (in module rarfile), 10
RAR_OS_OS2 (in module rarfile), 10
RAR_OS_UNIX (in module rarfile), 10
RAR_OS_WIN32 (in module rarfile), 10
RarCannotExec (class in rarfile), 11
RarCRCError (class in rarfile), 11
RarCreateError (class in rarfile), 11
RarExecError (class in rarfile), 10
RarExtFile (class in rarfile), 7
RarFatalError (class in rarfile), 10
RarFile (class in rarfile), 4
rarfile (module), 3
RarInfo (class in rarfile), 6
RarLockedArchiveError (class in rarfile), 11
RarMemoryError (class in rarfile), 11
RarNoFilesError (class in rarfile), 11
RarOpenError (class in rarfile), 11
RarSignalExit (class in rarfile), 11
RarUnknownError (class in rarfile), 11
RarUserBreak (class in rarfile), 11
RarUserError (class in rarfile), 11
RarWarning (class in rarfile), 10
RarWriteError (class in rarfile), 11
RarWrongPassword (class in rarfile), 11
read() (rarfile.RarExtFile method), 8
read() (rarfile.RarFile method), 5
readable() (rarfile.RarExtFile method), 8
readall() (rarfile.RarExtFile method), 8
readinto() (rarfile.RarExtFile method), 8
readline() (rarfile.RarExtFile method), 8
readlines() (rarfile.RarExtFile method), 8

S

seek() (rarfile.RarExtFile method), 8
seekable() (rarfile.RarExtFile method), 8
setpassword() (rarfile.RarFile method), 4
strerror() (rarfile.RarFile method), 6

T

tell() (rarfile.RarExtFile method), 8
testrar() (rarfile.RarFile method), 6
truncate() (rarfile.RarExtFile method), 9
TRY_ENCODINGS (in module rarfile), 9

U

UNRAR_TOOL (in module rarfile), 9
USE_EXTRACT_HACK (in module rarfile), 9

V

volume (rarfile.RarInfo attribute), 7
volume_file (rarfile.RarInfo attribute), 7
volumelist() (rarfile.RarFile method), 5

W

writable() (rarfile.RarExtFile method), 8