py7zr Documentation

Release 0.18.4

Hiroshi Miura

Apr 17, 2022

CONTENTS

CHAPTER

ONE

USER GUIDE

The 7z file format is a popular archive and compression format in recent days. This module provides tools to read, write and list 7z file. Features is not implemented to update and append a 7z file. py7zr does not support self-extracting archive, aka. SFX file, and only support plain 7z archive file.

1.1 Getting started

1.1.1 Install

The py7zr is written by Python and can be downloaded from PyPI(aka. Python Package Index) using standard 'pip' command as like follows;

```
$ pip install py7zr
```

The py7zr depends on several external libraries. You should install these libraries with py7zr. There are `PyCryptodome`, `PyZstd`, `PyPPMd`, `bcj-cffi`, `texttable`, and `multivolumefile`. These pakcages are automatically installed when installing with `pip` command.

1.1.2 Dependencies

There are several dependencies to support algorithms and CLI expressions.

Package	Purpose
PyCryptodomex	7zAES encryption
PyZstd	ZStandard compression
PyPPMd	PPMd compression
Brotli	Brotli compression (CPython)
BrotliCFFI	Brotli compression (PyPy)
zipfile-deflate64	DEFLATE64 decompression
pybcj	BCJ filters
multivolumefile	Multi-volume archive read/write
texttable	CLI formatter

1.1.3 Run Command

'py7zr' is a command script. You can run extracting a target file target.7z then command line become as such as follows;

```
$ py7zr x target.7z
```

When you want to create an archive from a files and directory under the current directory 'd', command line become as such as follows;

```
$ py7zr c target.7z  d/
```

1.2 Command-Line Interfaces

The *py7zr* module provides a simple command-line interface to interact with 7z archives.

If you want to extract a 7z archive into the specified directory, use the *x* subcommand:

```
$ python -m py7zr x monty.7z target-dir/
$ py7zr x monty.7z
```

For a list of the files in a 7z archive, use the *l* subcommand:

```
$ python -m py7zr l monty.7z
$ py7zr l monty.7z
```

1.2.1 Command-line options

l <7z file>
: List files in a 7z file.

x <7z file> [<output_dir>]
: Extract 7z file into target directory.

c <7z file> <base_dir>
: Create 7zip archive from base_directory

a <7z file> <base_dir>
: Append files from base_dir to existent 7zip archive.

i <7z file>
: Show archive information of specified 7zip archive.

t <7z file>
: Test whether the 7z file is valid or not.

1.2.2 Common command options

-P --password
: Extract, list or create password protected archive. py7zr will prompt user input.

--verbose
: Show verbose debug log.

1.2.3 Create command options

-v | --volume {Size}[b|k|m|g]
: Create multi-volume archive with Size. Usable with 'c' sub-command.

1.3 Programming APIs

1.3.1 Extraction

Here is a several example for extraction from your python program. You can write it with very clean syntax because py7zr supports context maanager.

```
import py7zr
with py7zr.SevenZipFile("Archive.7z", 'r') as archive:
    archive.extractall(path="/tmp")
```

This example extract a 7-zip archive file "Archive.7z" into "/tmp" target directory.

1.3.2 Make archive

Here is a simple example to make 7-zip archive.

```
import py7zr
with py7zr.SevenZipFile("Archive.7z", 'w') as archive:
    archive.writeall("target/")
```

1.3.3 Append files to archive

Here is a simple example to append some files into existent 7-zip archive.

```
import py7zr
with py7zr.SevenZipFile("Archive.7z", 'a') as archive:
    archive.write("additional_file.txt")
```

1.3.4 Extraction from multi-volume archive

You should concatenate multi-volume archives into single archive file before call py7zr, or consider using files wrapping class that handle multiple files as a virtual single file, (ex. multivolumefile library)

```
import py7zr
filenames = ['example.7z.0001', 'example.7z.0002']
with open('result.7z', 'ab') as outfile:  # append in binary mode
    for fname in filenames:
        with open(fname, 'rb') as infile:        # open in binary mode also
            outfile.write(infile.read())
with py7zr.SevenZipFile("result.7z", "r") as archive:
    archive.extractall()
os.unlink("result.7z)
```

Here is another example. This example use multivolumefile library. The multivolumefile library is in pre-alpha status, so it is not recommend to use production system.

```
pip install py7zr multivolumefile
```

When there are files named, 'example.7z.0001', 'example.7z.0002', and so on, following code will extract multi-volume archive.

```
import multivolumefile
import py7zr
with multivolumefile.open('example.7z', mode='rb') as target_archive:
    with SevenZipFile(target_archive, 'r') as archive:
        archive.extractall()
```

If you want to create multi volume archive using multivolumefile library, following example do it for you.

```
import multivolumefile
import py7zr

target = pathlib.Path('/target/directory/')
with multivolumefile.open('example.7z', mode='wb', volume_size=10240) as target_archive:
    with SevenZipFile(target_archive, 'w') as archive:
        archive.writeall(target, 'target')
```

1.4 Presentation material

See `Introductory presentation(PDF)`, and `Introductory presentation(ODP)`.

CHAPTER

TWO

API DOCUMENTATION

2.1 py7zr — 7-Zip archive library

The module is built upon awesome development effort and knowledge of *pylzma* module and its *py7zlib.py* program by Joachim Bauch. Great appreciation for Joachim!

The module defines the following items:

exception py7zr.**Bad7zFile**

The error raised for bad 7z files.

class py7zr.**SevenZipFile**

The class for reading 7z files. See section *sevenzipfile-object*

class py7zr.**ArchiveInfo**

The class used to represent information about an information of an archive file. See section *archiveinfo-object*

class py7zr.**FileInfo**

The class used to represent information about a member of an archive file. See section

py7zr.**is_7zfile**(*filename*)

Returns `True` if *filename* is a valid 7z file based on its magic number, otherwise returns `False`. *filename* may be a file or file-like object too.

py7zr.**unpack_7zarchive**(*archive*, *path*, *extra=None*)

Helper function to intend to use with `shutil` module which offers a number of high-level operations on files and collections of files. Since `shutil` has a function to register decompressor of archive, you can register an helper function and then you can extract archive by calling `shutil.unpack_archive()`

```
shutil.register_unpack_format('7zip', ['.7z'], unpack_7zarchive)
shutil.unpack_archive(filename, [, extract_dir])
```

py7zr.**pack_7zarchive**(*archive*, *path*, *extra=None*)

Helper function to intend to use with `shutil` module which offers a number of high-level operations on files and collections of files. Since `shutil` has a function to register maker of archive, you can register an helper function and then you can produce archive by calling `shutil.make_archive()`

```
shutil.register_archive_format('7zip', pack_7zarchive, description='7zip archive')
shutil.make_archive(base_name, '7zip', base_dir)
```

See also:

(external link) shutil `shutil` module offers a number of high-level operations on files and collections of files.

2.2 Class description

2.2.1 ArchiveInfo Object

class py7zr.**ArchiveInfo**(*filename*, *stat*, *header_size*, *method_names*, *solid*, *blocks*, *uncompressed*)

Data only python object to hold information of archive. The object can be retrieved by *archiveinfo()* method of *SevenZipFile* object.

py7zr.**filename: str**

filename of 7zip archive. If SevenZipFile object is created from BinaryIO object, it becomes None.

py7zr.**stat: stat_result**

fstat object of 7zip archive. If SevenZipFile object is created from BinaryIO object, it becomes None.

py7zr.**header_size: int**

header size of 7zip archive.

py7zr.**method_names: List[str]**

list of method names used in 7zip archive. If method is not supported by py7zr, name has a postfix asterisk(*) mark.

py7zr.**solid: bool**

Whether is 7zip archive a solid compression or not.

py7zr.**blocks: int**

number of compression block(s)

py7zr.**uncompressed: int**

total uncompressed size of files in 7zip archive

2.2.2 SevenZipFile Object

class py7zr.**SevenZipFile**(*file*, *mode='r'*, *filters=None*, *dereference=False*, *password=None*)

Open a 7z file, where *file* can be a path to a file (a string), a file-like object or a *path-like object*.

The *mode* parameter should be `'r'` to read an existing file, `'w'` to truncate and write a new file, `'a'` to append to an existing file, or `'x'` to exclusively create and write a new file. If *mode* is `'x'` and *file* refers to an existing file, a `FileExistsError` will be raised. If *mode* is `'r'` or `'a'`, the file should be seekable.

The *filters* parameter controls the compression algorithms to use when writing files to the archive.

SevenZipFile class has a capability as context manager. It can handle 'with' statement.

If dereference is False, add symbolic and hard links to the archive. If it is True, add the content of the target files to the archive. This has no effect on systems that do not support symbolic links.

When password given, py7zr handles an archive as an encrypted one.

SevenZipFile.**close**()

Close the archive file and release internal buffers. You must call *close()* before exiting your program or most records will not be written.

SevenZipFile.**getnames**()

Return a list of archive files by name.

SevenZipFile.**needs_password**()

Return *True* if the archive is encrypted, or is going to create encrypted archive. Otherwise return *False*

SevenZipFile.**extractall**(*path=None*)

Extract all members from the archive to current working directory. *path* specifies a different directory to extract to.

SevenZipFile.**extract**(*path=None*, *targets=None*)

Extract specified pathspec archived files to current working directory. 'path' specifies a differenct directory to extract to.

'targets' is a list of archived files to be extracted. py7zr looks for files and directories as same as specified in 'targets'.

Once extract() called, the SevenZipFIle object become exhausted and EOF state. If you want to call read(), readall(), extract(), extractall() again, you should call reset() before it.

CAUTION when specifying files and not specifying parent directory, py7zr will fails with no such directory. When you want to extract file 'somedir/somefile' then pass a list: ['somedirectory', 'somedir/somefile'] as a target argument.

Please see 'tests/test_basic.py: test_py7zr_extract_and_getnames()' for example code.

```
filter_pattern = re.compile(r'scripts.*')
with SevenZipFile('archive.7z', 'r') as zip:
     allfiles = zip.getnames()
     targets = [f if filter_pattern.match(f) for f in allfiles]
with SevenZipFile('archive.7z', 'r') as zip:
     zip.extract(targets=targets)
```

SevenZipFile.**readall**()

Extract all members from the archive to memory and returns dictionary object. Returned dictionary has a form of Dict[filename: str, BinaryIO: io.BytesIO object]. Once readall() called, the SevenZipFIle object become exhausted and EOF state. If you want to call read(), readall(), extract(), extractall() again, you should call reset() before it. You can get extracted data from dictionary value as such

```
with SevenZipFile('archive.7z', 'r') as zip:
    for fname, bio in zip.readall().items():
        print('{:s}: {:X}...'.format(name, bio.read(10))
```

SevenZipFile.**read**(*targets=None*)

Extract specified list of target archived files to dictionary object. 'targets' is a list of archived files to be extracted. py7zr looks for files and directories as same as specified in 'targets'. When targets is None, it behave as same as readall(). Once read() called, the SevenZipFIle object become exhausted and EOF state. If you want to call read(), readall(), extract(), extractall() again, you should call reset() before it.

```
filter_pattern = re.compile(r'scripts.*')
with SevenZipFile('archive.7z', 'r') as zip:
     allfiles = zip.getnames()
     targets = [f for f in allfiles if filter_pattern.match(f)]
with SevenZipFile('archive.7z', 'r') as zip:
     for fname, bio in zip.read(targets).items():
         print('{:s}: {:X}...'.format(name, bio.read(10))
```

SevenZipFile.**list**()

Return a List[FileInfo].

SevenZipFile.**archiveinfo**()

Return a ArchiveInfo object.

SevenZipFile.**test**()

Read all the archive file and check a packed CRC. Return `True` if CRC check passed, and return `False` when detect defeat, or return `None` when the archive don't have a CRC record.

SevenZipFile.**testzip**()

Read all the files in the archive and check their CRCs. Return the name of the first bad file, or else return `None`. When the archive don't have a CRC record, it return `None`.

SevenZipFile.**write**(*filename*, *arcname=None*)

Write the file named *filename* to the archive, giving it the archive name *arcname* (by default, this will be the same as *filename*, but without a drive letter and with leading path separators removed). The archive must be open with mode `'w'`

SevenZipFile.**writeall**(*filename*, *arcname=None*)

Write the directory and its sub items recursively into the archive, giving the archive name *arcname* (by default, this will be the same as *filename*, but without a drive letter and with leading path seaprator removed).

If you want to store directories and files, putting *arcname* is good idea. When filename is 'C:/a/b/c' and arcname is 'c', with a file exist as 'C:/a/b/c/d.txt', then archive listed as ['c', 'c/d.txt'], the former as directory.

SevenZipFile.**set_encrypted_header**(*mode*)

Set header encryption mode. When encrypt header, set mode to *True*, otherwise *False*. Default is *False*.

SevenZipFile.**set_encoded_header_mode**(*mode*)

Set header encode mode. When encode header data, set mode to *True*, otherwise *False*. Default is *True*.

2.3 Compression Methods

'py7zr' supports algorithms and filters which lzma module and liblzma support. It also support BZip2 and Deflate that are implemented in python core libraries, and ZStandard with third party libraries. *py7zr*, python3 core lzma module and *liblzma* do not support some algorithms such as PPMd, BCJ2 and Deflate64.

Here is a table of algorithms.

#	Category	Algorithm	Note
1	• Compression • Decompression	LZMA2	default (LZMA2+BCJ)
2		LZMA	
3		Bzip2	
4		Deflate	
5		COPY	
6		PPMd	depend on pyppmd
7		ZStandard	depend on pyzstd
8		Brotli	depend on brotli,brotliCFFI
9	• Filter	BCJ	**(X86, ARM, PPC, ARMT, SPARC,** IA64) depend on bcj-cffi
10		Delta	
11	• Encryption • Decryption	7zAES	depend on pycryptodomex
12	• Unsupported	BCJ2	
13		Deflate64	

- A feature handling symbolic link is basically compatible with 'p7zip' implementation, but not work with original 7-zip because the original does not implement the feature.

2.4 Possible filters value

Here is a list of examples for possible filters values. You can use it when creating SevenZipFile object.

```
from py7zr import FILTER_LZMA, SevenZipFile

filters = [{'id': FILTER_LZMA}]
archive = SevenZipFile('target.7z', mode='w', filters=filters)
```

LZMA2 + Delta `[{'id': FILTER_DELTA}, {'id': FILTER_LZMA2, 'preset': PRESET_DEFAULT}]`

LZMA2 + BCJ `[{'id': FILTER_X86}, {'id': FILTER_LZMA2, 'preset': PRESET_DEFAULT}]`

LZMA2 + ARM `[{'id': FILTER_ARM}, {'id': FILTER_LZMA2, 'preset': PRESET_DEFAULT}]`

LZMA + BCJ `[{'id': FILTER_X86}, {'id': FILTER_LZMA}]`

LZMA2 `[{'id': FILTER_LZMA2, 'preset': PRESET_DEFAULT}]`

LZMA `[{'id': FILTER_LZMA}]`

BZip2 `[{'id': FILTER_BZIP2}]`

Deflate `[{'id': FILTER_DEFLATE}]`

ZStandard `[{'id': FILTER_ZSTD, 'level': 3}]`

PPMd `[{'id': FILTER_PPMD, 'order': 6, 'mem': 24}]`

`[{'id': FILTER_PPMD, 'order': 6, 'mem': "16m"}]`

Brolti `[{'id': FILTER_BROTLI, 'level': 11}]`

7zAES + LZMA2 + Delta `[{'id': FILTER_DELTA}, {'id': FILTER_LZMA2, 'preset': PRESET_DEFAULT}, {'id': FILTER_CRYPTO_AES256_SHA256}]`

7zAES + LZMA2 + BCJ `[{'id': FILTER_X86}, {'id': FILTER_LZMA2, 'preset': PRESET_DEFAULT}, {'id': FILTER_CRYPTO_AES256_SHA256}]`

7zAES + LZMA `[{'id': FILTER_LZMA}, {'id': FILTER_CRYPTO_AES256_SHA256}]`

7zAES + Deflate `[{'id': FILTER_DEFLATE}, {'id': FILTER_CRYPTO_AES256_SHA256}]`

7zAES + BZip2 `[{'id': FILTER_BZIP2}, {'id': FILTER_CRYPTO_AES256_SHA256}]`

7zAES + ZStandard `[{'id': FILTER_ZSTD}, {'id': FILTER_CRYPTO_AES256_SHA256}]`

CHAPTER

THREE

CONTRIBUTOR GUIDE

3.1 Development environment

If you're reading this, you're probably interested in contributing to py7zr. Thank you very much! The purpose of this guide is to get you to the point where you can make improvements to the py7zr and share them with the rest of the team.

3.1.1 Setup Python

The py7zr is written in the Python programming language. Python installation for various platforms with various ways. You need to install Python environment which support *pip* command. Venv/Virtualenv is recommended for development.

We have a test suite with python 3.6, 3.7, 3.8 and pypy3. If you want to run all the test with these versions and variant on your local, you should install these versions. You can run test with CI environment on Github actions.

3.1.2 Get Early Feedback

If you are contributing, do not feel the need to sit on your contribution until it is perfectly polished and complete. It helps everyone involved for you to seek feedback as early as you possibly can. Submitting an early, unfinished version of your contribution for feedback in no way prejudices your chances of getting that contribution accepted, and can save you from putting a lot of work into a contribution that is not suitable for the project.

3.2 Code Contributions

3.2.1 Steps submitting code

When contributing code, you'll want to follow this checklist:

1. Fork the repository on GitHub.
2. Run the tox tests to confirm they all pass on your system. If they don't, you'll need to investigate why they fail. If you're unable to diagnose this yourself, raise it as a bug report.
3. Write tests that demonstrate your bug or feature. Ensure that they fail.
4. Make your change.
5. Run the entire test suite again using tox, confirming that all tests pass including the ones you just added.
6. Send a GitHub Pull Request to the main repository's master branch. GitHub Pull Requests are the expected method of code collaboration on this project.

3.2.2 Code review

Contribution will not be merged until they have been code reviewed. There are limited reviewer in the team, reviews from other contributors are also welcome. You should implemented a review feedback unless you strongly object to it.

3.2.3 Code style

The py7zr uses the PEP8 code style. In addition to the standard PEP8, we have an extended guidelines

- line length should not exceed 125 charactors.
- It also use MyPy static type check enforcement.

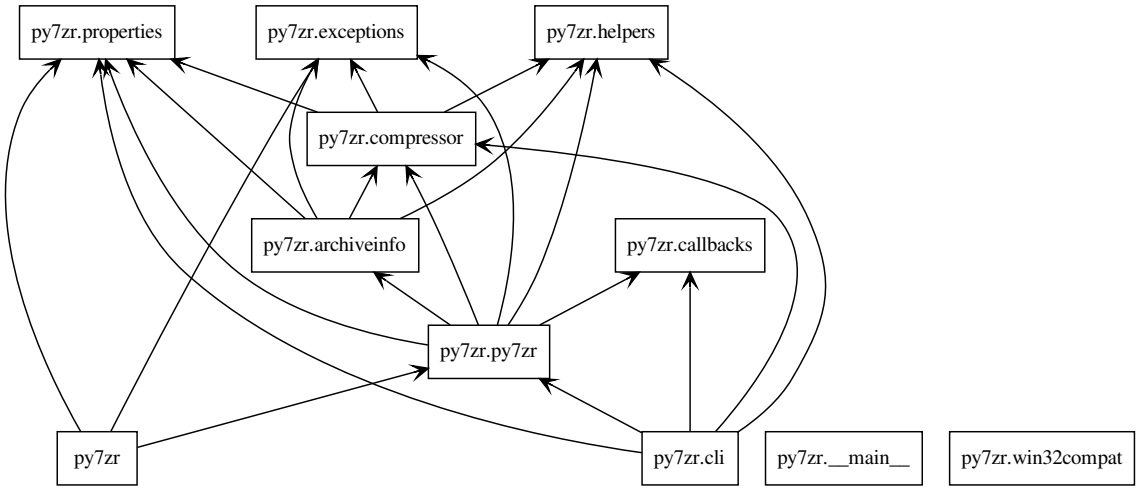
3.3 Class and module design

The py7zr take class design that categorized into several sub modules to reflect its role.

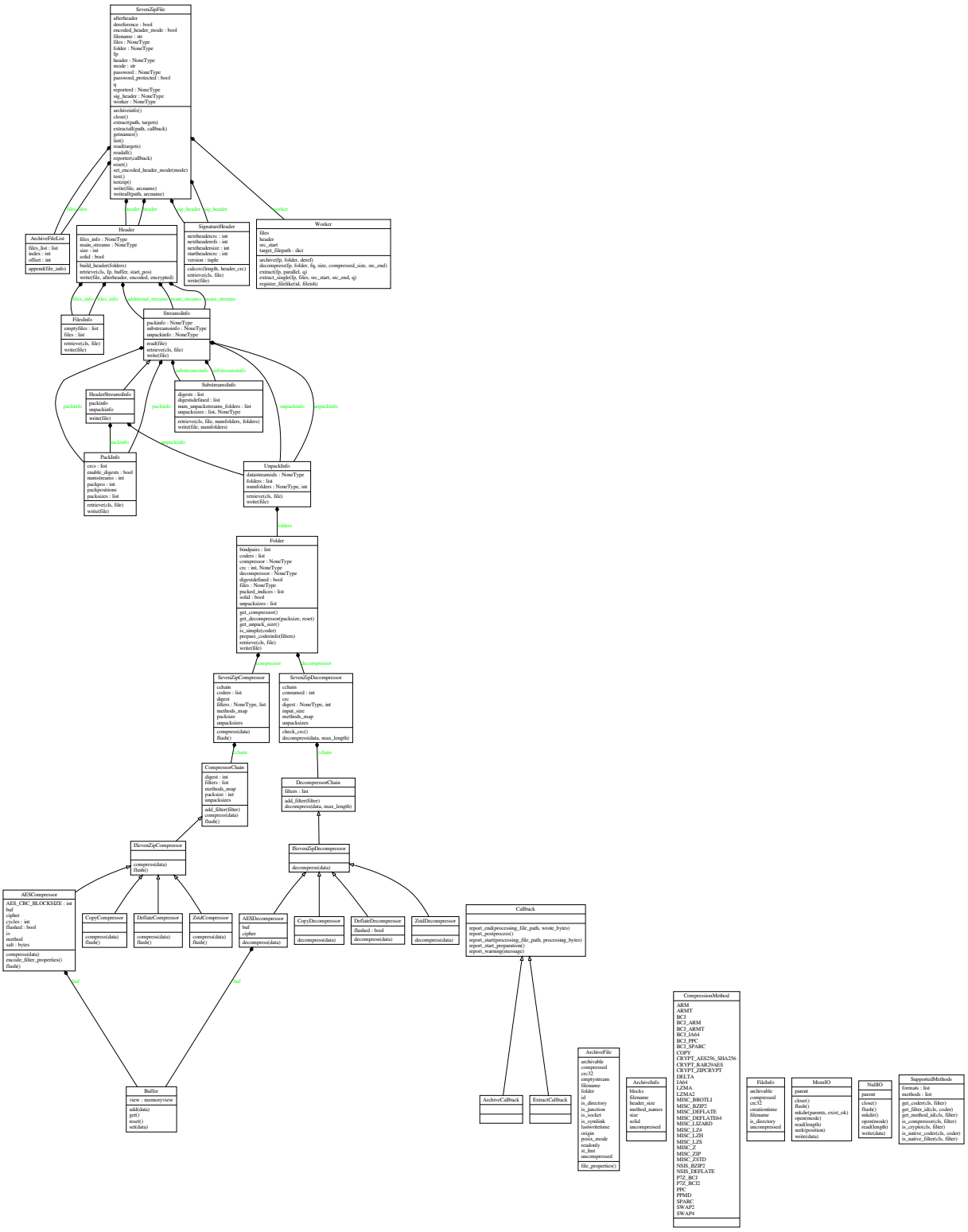
The main class is py7zr.SevenZipFile() class which provide API for library users. The main internal classes are in the submodule py7zr.archiveinfo, which takes class structure as same as .7z file format structure.

Another important submodule is py7zr.compressor module that hold all related compression and encryption proxy classes for corresponding libraries to convert various interfaces into common ISevenZipCompressor() and ISevenZipDecompressor() interface.

All UI related classes and functions are separated from core modules. cli submodule is a place for command line functions and pretty printings.

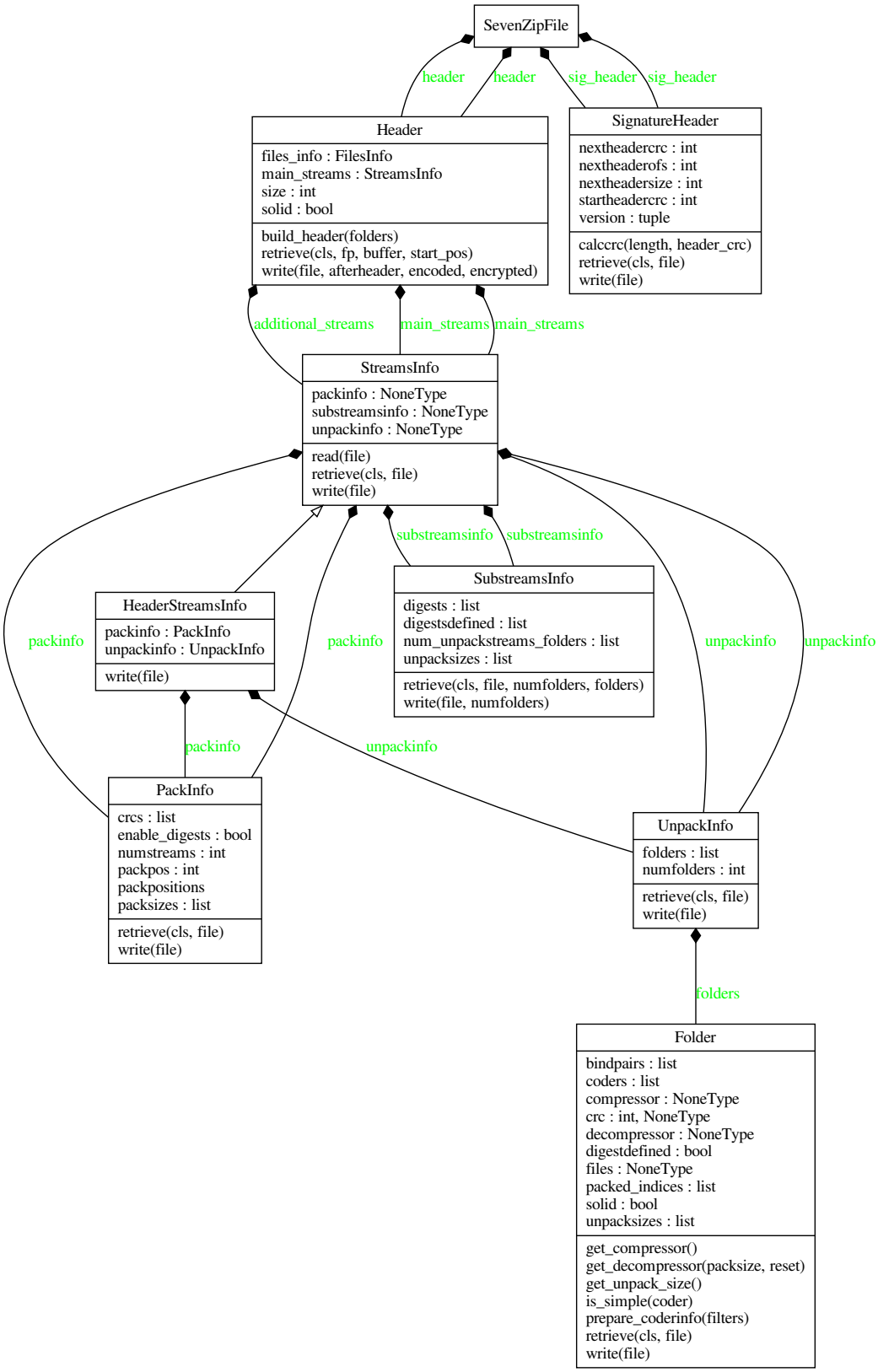


Here is a whole classes diagram. There are part by part descriptions at Next sections.


SevenZipFile
afterheader
dereference : bool
encoded_header_mode : bool
filename : str
files : NoneType
folder : NoneType
fp
header : NoneType
mode : str
password : NoneType
password_protected : bool
q
reporterd : NoneType
sig_header : NoneType
worker : NoneType
archiveinfo()
close()
extract(path, targets)
extractall(path, callback)
getnames()
list()
read(targets)
readall()
reporter(callback)
reset()
set_encoded_header_mode(mode)
test()
testzip()
write(file, arcname)
writeall(path, arcname)
file_info
header_header
sig_header
sig_header
worker
ArchiveFileList
files_list : list
index : int
offset : int
append(file_info)
Header
files_info : NoneType
main_streams : NoneType
size : int
solid : bool
build_header(folders)
retrieve(cls, fp, buffer, start_pos)
write(file, afterheader, encoded, encrypted)
SignatureHeader
nextheadercrc : int
nextheadersize : int
nextheaderofs : int
startheadercrc : int
version : tuple
calccrc(length, header_crc)
retrieve(cls, file)
write(file)
Worker
files
header
src_start
target_filepath : dict
archive(fp, folder, deref)
decompress(fp, folder, fq, size, compressed_size, src_end)
extract(fp, parallel, q)
extract_single(fp, files, src_start, src_end, q)
register_filelike(id, fileish)
files_info
files_info
additional_streams
main_streams
main_streams
FilesInfo
emptyfiles : list
files : list
retrieve(cls, file)
write(file)
StreamsInfo
packinfo : NoneType
substreamsinfo : NoneType
unpackinfo : NoneType
read(file)
retrieve(cls, file)
write(file)
substreamsinfo
substreamsinfo
HeaderStreamsInfo
packinfo
unpackinfo
write(file)
SubstreamsInfo
digests : list
digestsdefined : list
num_unpackstreams_folders : list
unpacksizes : list, NoneType
retrieve(cls, file, numfolders, folders)
write(file, numfolders)
packinfo
packinfo
unpackinfo
unpackinfo
packinfo
unpackinfo
PackInfo
crcs : list
enable_digests : bool
numstreams : int
packpos : int
packsizeitems
packsizes : list
retrieve(cls, file)
write(file)
UnpackInfo
datastreamidx : NoneType
folders : list
numfolders : NoneType, int
retrieve(cls, file)
write(file)
folders
Folder
bindpairs : list
coders : list
compressor : NoneType
crc : int, NoneType
decompressor : NoneType
digestsdefined : bool
files : NoneType
packed_indices : list
solid : bool
unpacksizes : list
get_compressor()
get_decompressor(packsize, reset)
get_unpack_size()
is_simple(coder)
prepare_coderinfo(filters)
retrieve(cls, file)
write(file)
compressor
decompressor
SevenZipCompressor
cchain
coders : list
digest
filters : NoneType, list
methods_map
packsize
unpacksizes
compress(data)
flush()
SevenZipDecompressor
cchain
consumed : int
crc
digest : NoneType, int
input_size
methods_map
unpacksizes
check_crc()
decompress(data, max_length)
cchain
cchain
CompressorChain
digest : int
filters : list
methods_map
packsize : int
unpacksizes
add_filter(filter)
compress(data)
flush()
DecompressorChain
filters : list
add_filter(filter)
decompress(data, max_length)
ISevenZipCompressor
compress(data)
flush()
ISevenZipDecompressor
decompress(data)
AESCompressor
AES_CBC_BLOCKSIZE : int
buf
cipher
cycles : int
flushed : bool
iv
method
salt : bytes
compress(data)
encode_filter_properties()
flush()
CopyCompressor
compress(data)
flush()
DeflateCompressor
compress(data)
flush()
ZstdCompressor
compress(data)
flush()
AESDecompressor
buf
cipher
decompress(data)
CopyDecompressor
decompress(data)
DeflateDecompressor
flushed : bool
decompress(data)
ZstdDecompressor
decompress(data)
buf
buf
Buffer
view : memoryview
add(data)
get()
reset()
set(data)
Callback
report_end(processing_file_path, wrote_bytes)
report_postprocess()
report_start(processing_file_path, processing_bytes)
report_start_preparation()
report_warning(message)
ArchiveCallback
ExtractCallback
ArchiveFile
archivable
compressed
crc32
emptystream
filename
folder
id
is_directory
is_junction
is_socket
is_symlink
lastwritetime
origin
posix_mode
readonly
st_fmt
uncompressed
file_properties()
ArchiveInfo
blocks
filename
header_size
method_names
size
solid
uncompressed
CompressionMethod
ARM
ARMT
BCJ
BCJ_ARM
BCJ_ARMT
BCJ_IA64
BCJ_PPC
BCJ_SPARC
COPY
CRYPT_AES256_SHA256
CRYPT_RAR29AES
CRYPT_ZIPCRYPT
DELTA
IA64
LZMA
LZMA2
MISC_BROTLI
MISC_BZIP2
MISC_DEFLATE
MISC_DEFLATE64
MISC_LZIARD
MISC_LZ4
MISC_LZH
MISC_LZS
MISC_Z
MISC_ZIP
MISC_ZSTD
NSIS_BZIP2
NSIS_DEFLATE
PTZ_BCJ
PTZ_BCJ2
PPC
PPMD
SPARC
SWAP2
SWAP4
FileInfo
archivable
compressed
crc32
creationtime
filename
is_directory
uncompressed
MemIO
parent
close()
flush()
mkdir(parents, exist_ok)
open(mode)
read(length)
seek(position)
write(data)
NullIO
parent
close()
flush()
mkdir()
open(mode)
read(length)
write(data)
SupportedMethods
formats : list
methods : list
get_coder(cls, filter)
get_filter_id(cls, coder)
get_method_id(cls, filter)
is_compressor(cls, filter)
is_crypto(cls, filter)
is_native_coder(cls, coder)
is_native_filter(cls, filter)

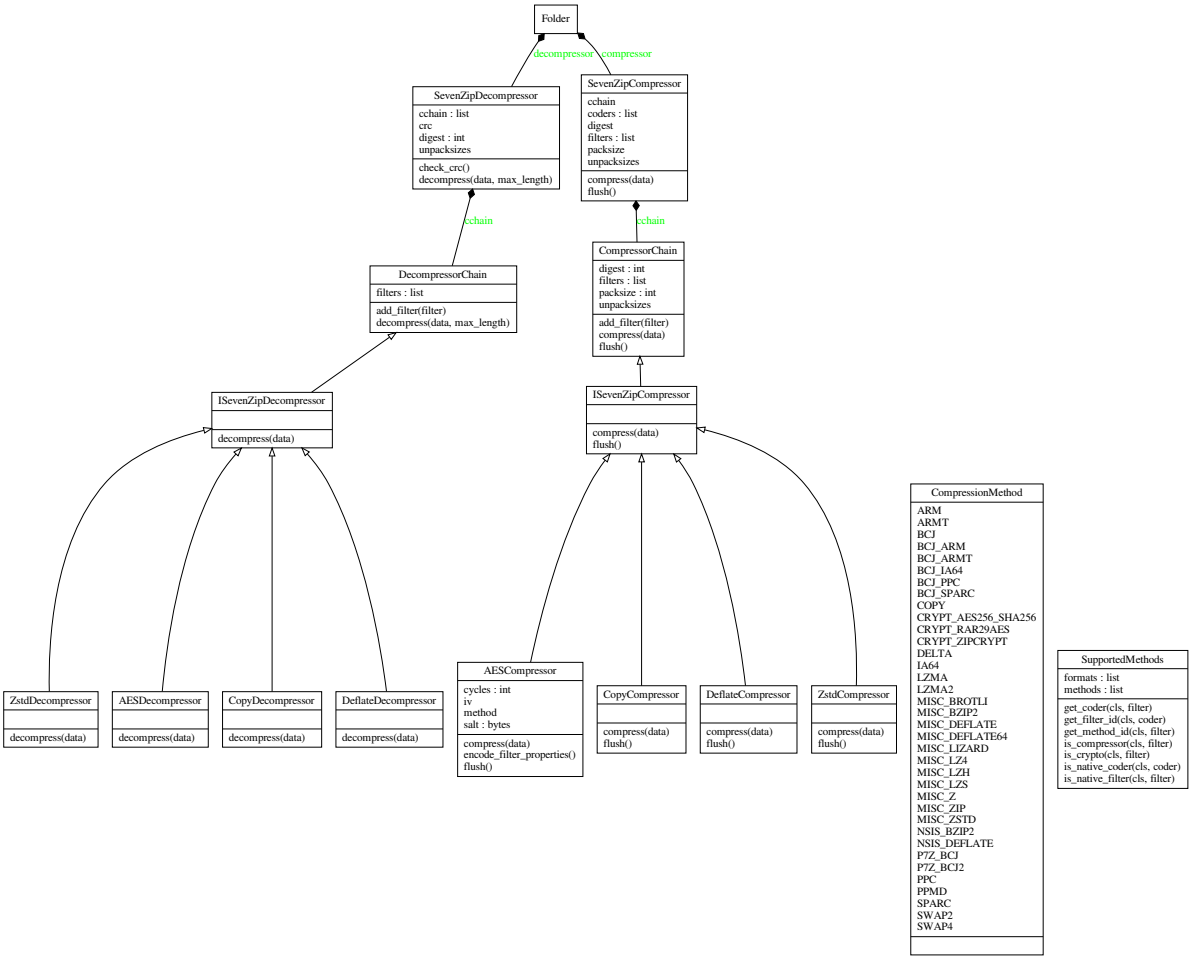
3.3.1 Header classes

Header related classes are in py7zr.archiveinfo submodule.


SevenZipFile
header
header
sig_header
sig_header
Header
files_info : FilesInfo
main_streams : StreamsInfo
size : int
solid : bool
build_header(folders)
retrieve(cls, fp, buffer, start_pos)
write(file, afterheader, encoded, encrypted)
SignatureHeader
nextheadercrc : int
nextheaderofs : int
nextheadersize : int
startheadercrc : int
version : tuple
calccrc(length, header_crc)
retrieve(cls, file)
write(file)
additional_streams
main_streams
main_streams
StreamsInfo
packinfo : NoneType
substreamsinfo : NoneType
unpackinfo : NoneType
read(file)
retrieve(cls, file)
write(file)
substreamsinfo
substreamsinfo
SubstreamsInfo
digests : list
digestsdefined : list
num_unpackstreams_folders : list
unpacksizes : list
retrieve(cls, file, numfolders, folders)
write(file, numfolders)
HeaderStreamsInfo
packinfo : PackInfo
unpackinfo : UnpackInfo
write(file)
packinfo
packinfo
unpackinfo
unpackinfo
packinfo
unpackinfo
PackInfo
crcs : list
enable_digests : bool
numstreams : int
packpos : int
packpositions
packsizes : list
retrieve(cls, file)
write(file)
UnpackInfo
folders : list
numfolders : int
retrieve(cls, file)
write(file)
folders
Folder
bindpairs : list
coders : list
compressor : NoneType
crc : int, NoneType
decompressor : NoneType
digestdefined : bool
files : NoneType
packed_indices : list
solid : bool
unpacksizes : list
get_compressor()
get_decompressor(packsize, reset)
get_unpack_size()
is_simple(coder)
prepare_coderinfo(filters)
retrieve(cls, file)
write(file)

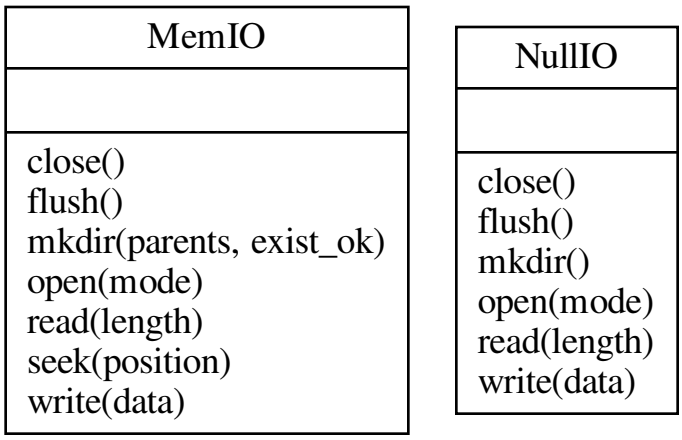
3.3.2 Compressor classes

There are compression related classes in py7zr.compressor submodule.



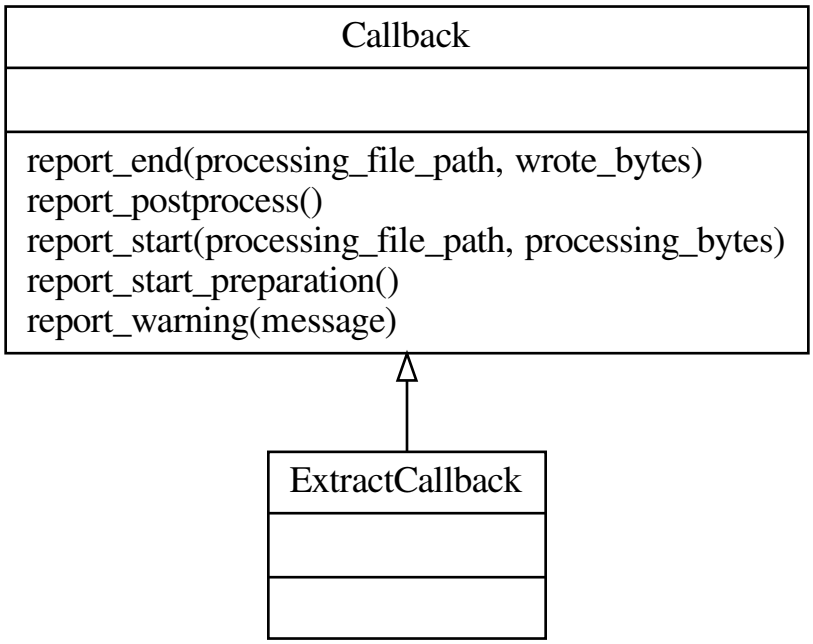
3.3.3 IO Abstraction classes

There are two IO abstraction classes to provide Mem API and check method.



3.3.4 Callback classes

Here is a callback interface class. ExtractCallback class is a concrete class used in CLI.



3.4 Classes details

Here is a detailed interface documentation for implementer.

3.4.1 ArchiveFile Objects

Read 7zip format archives.

class py7zr.py7zr.**ArchiveFile**(*id: int*, *file_info: Dict[str, Any]*)

Represent each files metadata inside archive file. It holds file properties; filename, permissions, and type whether it is directory, link or normal file.

Instances of the *ArchiveFile* class are returned by iterating `files_list` of *SevenZipFile* objects. Each object stores information about a single member of the 7z archive. Most of users use `extractall()`.

The class also hold an archive parameter where file is exist in archive file folder(container).

property archivable: bool

File has a Windows *archive* flag.

property compressed: Optional[int]

Compressed size

property crc32: Optional[int]

CRC of archived file(optional)

property emptystream: bool

True if file is empty(0-byte file), otherwise False

file_properties() → Dict[str, Any]

Return file properties as a hash object. Following keys are included: 'readonly', 'is_directory', 'posix_mode', 'archivable', 'emptystream', 'filename', 'creationtime', 'lastaccesstime', 'lastwritetime', 'attributes'

property filename: str

return filename of archive file.

has_strdata() → bool

True if file content is set by writestr() method otherwise False.

property is_directory: bool

True if file is a directory, otherwise False.

property is_junction: bool

True if file is a junction/reparse point on windows, otherwise False.

property is_socket: bool

True if file is a socket, otherwise False.

property is_symlink: bool

True if file is a symbolic link, otherwise False.

property lastwritetime: Optional[*py7zr.helpers.ArchiveTimestamp*]

Return last written timestamp of a file.

property posix_mode: Optional[int]

posix mode when a member has a unix extension property, or None :return: Return file stat mode can be set by os.chmod()

property readonly: bool

True if file is readonly, otherwise False.

property st_fmt: Optional[int]

Returns Return the portion of the file mode that describes the file type

class py7zr.py7zr.**ArchiveFileList**(*offset: int = 0*)

Iteratable container of ArchiveFile.

class py7zr.py7zr.**ArchiveInfo**(*filename: str*, *stat: os.stat_result*, *header_size: int*, *method_names: List[str]*, *solid: bool*, *blocks: int*, *uncompressed: List[int]*)

Hold archive information

class py7zr.py7zr.**FileInfo**(*filename*, *compressed*, *uncompressed*, *archivable*, *is_directory*, *creationtime*, *crc32*)

Hold archived file information.

class py7zr.py7zr.**SevenZipFile**(*file: Union[BinaryIO, str, pathlib.Path]*, *mode: str = 'r'*, ***, *filters: Optional[List[Dict[str, int]]] = None*, *dereference=False*, *password: Optional[str] = None*, *header_encryption: bool = False*, *blocksize: Optional[int] = None*, *mp: bool = False*)

The SevenZipFile Class provides an interface to 7z archives.

close()

Flush all the data into archive and close it. When close py7zr start reading target and writing actual archive file.

extractall(*path: Optional[Any] = None*, *callback: Optional[py7zr.callbacks.ExtractCallback] = None*) → None

Extract all members from the archive to the current working directory and set owner, modification time and permissions on directories afterwards. `path' specifies a different directory to extract to.

getnames() → List[str]

Return the members of the archive as a list of their names. It has the same order as the list returned by getmembers().

list() → List[*py7zr.py7zr.FileInfo*]

Returns contents information

reset() → None

When read mode, it reset file pointer, decompress worker and decompressor

write(*file: Union[pathlib.Path, str]*, *arcname: Optional[str] = None*)

Write single target file into archive(Not implemented yet).

writeall(*path: Union[pathlib.Path, str]*, *arcname: Optional[str] = None*)

Write files in target path into archive.

class py7zr.py7zr.**Worker**(*files*, *src_start: int*, *header*, *mp=False*)

Extract worker class to invoke handler

archive(*fp: BinaryIO*, *files*, *folder*, *deref=False*)

Run archive task for specified 7zip folder.

decompress(*fp: BinaryIO*, *folder*, *fq: IO[Any]*, *size: int*, *compressed_size: Optional[int]*, *src_end: int*) → int

decompressor wrapper called from extract method.

Parameters

- **fp** – archive source file pointer
- **folder** – Folder object that have decompressor object.
- **fq** – output file pathlib.Path
- **size** – uncompressed size of target file.
- **compressed_size** – compressed size of target file.
- **src_end** – end position of the folder

:returns None

extract(*fp: BinaryIO*, *parallel: bool*, *skip_notarget=True*, *q=None*) → None

Extract worker method to handle 7zip folder and decompress each files.

extract_single(*fp: Union[BinaryIO, str]*, *files*, *src_start: int*, *src_end: int*, *q: Optional[queue.Queue]*, *exc_q: Optional[queue.Queue] = None*, *skip_notarget=True*) → None

Single thread extractor that takes file lists in single 7zip folder.

register_filelike(*id: int*, *fileish: Optional[Union[*py7zr.helpers.MemIO*, pathlib.Path]]*) → None

register file-ish to worker.

py7zr.py7zr.**is_7zfile**(*file: Union[BinaryIO, str, pathlib.Path]*) → bool

Quickly see if a file is a 7Z file by checking the magic number. The file argument may be a filename or file-like object too.

py7zr.py7zr.**pack_7zarchive**(*base_name*, *base_dir*, *owner=None*, *group=None*, *dry_run=None*, *logger=None*)

Function for registering with shutil.register_archive_format()

py7zr.py7zr.**unpack_7zarchive**(*archive*, *path*, *extra=None*)

Function for registering with shutil.register_unpack_format()

3.4.2 archiveinfo module

class py7zr.archiveinfo.**Bond**(*incoder*, *outcoder*)

Represent bindings between two methods. bonds[i] = (incoder, outstream) means methods[i].stream[outstream] output data go to method[incoder].stream[0]

class py7zr.archiveinfo.**FilesInfo**

holds file properties

class py7zr.archiveinfo.**Folder**

a "Folder" represents a stream of compressed data. coders: list of coder num_coders: length of coders coder: hash list keys of coders: method, numinstreams, numoutstreams, properties unpacksizes: uncompressed sizes of outstreams

class py7zr.archiveinfo.**Header**

the archive header

class py7zr.archiveinfo.**HeaderStreamsInfo**

Header version of StreamsInfo

class py7zr.archiveinfo.**PackInfo**
information about packed streams

class py7zr.archiveinfo.**SignatureHeader**
The SignatureHeader class hold information of a signature header of archive.

class py7zr.archiveinfo.**StreamsInfo**
information about compressed streams

class py7zr.archiveinfo.**SubstreamsInfo**
defines the substreams of a folder

class py7zr.archiveinfo.**UnpackInfo**
combines multiple folders

class py7zr.archiveinfo.**WriteWithCrc**(*fp: BinaryIO*)
Thin wrapper for file object to calculate crc32 when write called.

py7zr.archiveinfo.**read_real_uint64**(*file: BinaryIO*) → Tuple[int, bytes]
read 8 bytes, return unpacked value as a little endian unsigned long long, and raw data.

py7zr.archiveinfo.**read_uint32**(*file: BinaryIO*) → Tuple[int, bytes]
read 4 bytes, return unpacked value as a little endian unsigned long, and raw data.

py7zr.archiveinfo.**read_uint64**(*file: BinaryIO*) → int
read UINT64, definition show in write_uint64()

py7zr.archiveinfo.**read_utf16**(*file: BinaryIO*) → str
read a utf-16 string from file

py7zr.archiveinfo.**write_real_uint64**(*file: BinaryIO*, *value: int*)
write 8 bytes, as an unsigned long long.

py7zr.archiveinfo.**write_uint32**(*file: BinaryIO*, *value*)
write uint32 value in 4 bytes.

py7zr.archiveinfo.**write_uint64**(*file: BinaryIO*, *value: int*)
UINT64 means real UINT64 encoded with the following scheme:

Size of encoding sequence depends from first byte:
First_Byte Extra_Bytes Value
(binary)
0xxxxxxx : (xxxxxxx)
10xxxxxx BYTE y[1] : (xxxxxx << (8 * 1)) + y
110xxxxx BYTE y[2] : (xxxxx << (8 * 2)) + y
...
1111110x BYTE y[6] : (x << (8 * 6)) + y
11111110 BYTE y[7] : y
11111111 BYTE y[8] : y

py7zr.archiveinfo.**write_utf16**(*file: BinaryIO*, *val: str*)
write a utf-16 string to file

3.4.3 compressor module

class py7zr.compressor.**AESCompressor**(*password: str*, *blocksize: Optional[int] = None*)

AES Compression(Encryption) class. It accept pre-processing filter which may be a LZMA compression.

compress(*data*)

Compression + AES encryption with 16byte alignment.

flush()

Flush output buffer(interface) :return: output data

class py7zr.compressor.**AESDecompressor**(*aes_properties: bytes*, *password: str*, *blocksize: Optional[int] = None*)

Decrypt data

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**BCJDecoder**(*size: int*)

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**BCJEncoder**

compress(*data: Union[bytes, bytearray, memoryview]*) → bytes

Compress data (interface) :param data: input data :return: output data

flush()

Flush output buffer(interface) :return: output data

class py7zr.compressor.**BcjArmDecoder**(*size: int*)

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**BcjArmEncoder**

compress(*data: Union[bytes, bytearray, memoryview]*) → bytes

Compress data (interface) :param data: input data :return: output data

flush()

Flush output buffer(interface) :return: output data

class py7zr.compressor.**BcjArmtDecoder**(*size: int*)

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**BcjArmtEncoder**

compress(*data: Union[bytes, bytearray, memoryview]*) → bytes

Compress data (interface) :param data: input data :return: output data

flush()

Flush output buffer(interface) :return: output data

class py7zr.compressor.**BcjPpcDecoder**(*size: int*)

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**BcjPpcEncoder**

compress(*data: Union[bytes, bytearray, memoryview]*) → bytes

Compress data (interface) :param data: input data :return: output data

flush()

Flush output buffer(interface) :return: output data

class py7zr.compressor.**BcjSparcDecoder**(*size: int*)

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**BcjSparcEncoder**

compress(*data: Union[bytes, bytearray, memoryview]*) → bytes

Compress data (interface) :param data: input data :return: output data

flush()

Flush output buffer(interface) :return: output data

class py7zr.compressor.**BrotliCompressor**(*level*)

compress(*data: Union[bytes, bytearray, memoryview]*) → bytes

Compress data (interface) :param data: input data :return: output data

flush() → bytes

Flush output buffer(interface) :return: output data

class py7zr.compressor.**BrotliDecompressor**(*properties: bytes*, *block_size: int*)

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*)

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**CopyCompressor**

compress(*data: Union[bytes, bytearray, memoryview]*) → bytes

Compress data (interface) :param data: input data :return: output data

flush()

Flush output buffer(interface) :return: output data

class py7zr.compressor.**CopyDecompressor**

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**Deflate64Compressor**

class py7zr.compressor.**Deflate64Decompressor**

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**DeflateCompressor**

compress(*data*)

Compress data (interface) :param data: input data :return: output data

flush()

Flush output buffer(interface) :return: output data

class py7zr.compressor.**DeflateDecompressor**

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**ISevenZipCompressor**

abstract compress(*data: Union[bytes, bytearray, memoryview]*) → bytes

Compress data (interface) :param data: input data :return: output data

abstract flush() → bytes

Flush output buffer(interface) :return: output data

class py7zr.compressor.**ISevenZipDecompressor**

abstract decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**LZMA1Decompressor**(*filters*, *unpacksize*)

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**MethodsType**(*value*)

An enumeration.

class py7zr.compressor.**PpmdCompressor**(*properties: bytes*)

Compress with PPMd compression algorithm

compress(*data: Union[bytes, bytearray, memoryview]*) → bytes

Compress data (interface) :param data: input data :return: output data

flush()

Flush output buffer(interface) :return: output data

class py7zr.compressor.**PpmdDecompressor**(*properties: bytes*, *blocksize: Optional[int] = None*)

Decompress PPMd compressed data

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length=- 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

class py7zr.compressor.**SevenZipCompressor**(*filters=None*, *password=None*, *blocksize: Optional[int] = None*)

Main compressor object to configured for each 7zip folder.

class py7zr.compressor.**SevenZipDecompressor**(*coders: List[Dict[str, Any]]*, *packsize: int*, *unpacksizes: List[int]*, *crc: Optional[int]*, *password: Optional[str] = None*, *blocksize: Optional[int] = None*)

Main decompressor object which is properly configured and bind to each 7zip folder. because 7zip folder can have a custom compression method

class py7zr.compressor.**SupportedMethods**

Hold list of methods.

class py7zr.compressor.**ZstdCompressor**(*level: int*)

compress(*data: Union[bytes, bytearray, memoryview]*) → bytes

Compress data (interface) :param data: input data :return: output data

flush() → bytes

Flush output buffer(interface) :return: output data

class py7zr.compressor.**ZstdDecompressor**(*properties: bytes*, *blocksize: int*)

decompress(*data: Union[bytes, bytearray, memoryview]*, *max_length: int = - 1*) → bytes

Decompress data (interface) :param data: input data :param max_length: maximum length of output data when it can respect, otherwise ignore. :return: output data

3.4.4 helpers module

class py7zr.helpers.**ArchiveTimestamp**

Windows FILETIME timestamp.

as_datetime()

Convert FILETIME to Python datetime object.

totimestamp() → float

Convert 7z FILETIME to Python timestamp.

exception py7zr.helpers.**BufferOverflow**

class py7zr.helpers.**LocalTimezone**

dst(*dt*)

datetime -> DST offset as timedelta positive east of UTC.

fromutc(*dt*)

datetime in UTC -> datetime in local time.

tzname(*dt*)

datetime -> string name of time zone.

utcoffset(*dt*)

datetime -> timedelta showing offset from UTC, negative values indicating West of UTC

class py7zr.helpers.**MemIO**(*buf: BinaryIO*)

pathlib.Path-like IO class to write memory(io.Bytes)

class py7zr.helpers.**NullIO**

pathlib.Path-like IO class of /dev/null

class py7zr.helpers.**UTC**

dst(*dt*)

datetime -> DST offset as timedelta positive east of UTC.

tzname(*dt*)

datetime -> string name of time zone.

utcoffset(*dt*)

datetime -> timedelta showing offset from UTC, negative values indicating West of UTC

py7zr.helpers.**calculate_crc32**(*data: bytes*, *value: int = 0*, *blocksize: int = 1048576*) → int

Calculate CRC32 of strings with arbitrary lengths.

py7zr.helpers.**calculate_key**(*password: bytes*, *cycles: int*, *salt: bytes*, *digest: str*) → bytes

Calculate 7zip AES encryption key. Concat values in order to reduce number of calls of Hash.update().

py7zr.helpers.**filetime_to_dt**(*ft*)

Convert Windows NTFS file time into python datetime object.

py7zr.helpers.**islink**(*path*)

Cross-platform islink implementation. Supports Windows NT symbolic links and reparse points.

py7zr.helpers.**readlink**(*path: Union[str, pathlib.Path]*, *, *dir_fd=None*) → Union[str, pathlib.Path]

Cross-platform compat implementation of os.readlink and Path.readlink(). Supports Windows NT symbolic links and reparse points. When called with path argument as pathlike(str), return result as a pathlike(str). When called with Path object, return also Path object. When called with path argument as bytes, return result as a bytes.

CHAPTER

FOUR

AUTHORS

py7zr is written and maintained by Hiroshi Miura <miurahr@linux.com>

Contributors, listed alphabetically, are:

- @andrebrait – Fix exception for empty 7z file (#118)
- Joachim Bauch – pylzma orginator
- Kazuya Fujioka – Fix zero file problem
- Kyle Altendorf – Fix multithreading problem (#82)
- @padremayi – Fix crash on wrong crationtime in archive (#275)
- @royopa – Fix typo (#108)
- @Zoynels – Mmeory IO API(#111, #119)

CHAPTER

FIVE

GLOSSARY

binary file A *file object* able to read and write *bytes-like objects*. Examples of binary files are files opened in binary mode (`'rb'`, `'wb'` or `'rb+'`), `sys.stdin.buffer`, `sys.stdout.buffer`, and instances of `io.BytesIO` and `gzip.GzipFile`.

See also *text file* for a file object able to read and write `str` objects.

bytes-like object An object that supports the *bufferobjects* and can export a C-*contiguous* buffer. This includes all `bytes`, `bytearray`, and `array.array` objects, as well as many common `memoryview` objects. Bytes-like objects can be used for various operations that work with binary data; these include compression, saving to a binary file, and sending over a socket.

Some operations need the binary data to be mutable. The documentation often refers to these as "read-write bytes-like objects". Example mutable buffer objects include `bytearray` and a `memoryview` of a `bytearray`. Other operations require the binary data to be stored in immutable objects ("read-only bytes-like objects"); examples of these include `bytes` and a `memoryview` of a `bytes` object.

contiguous A buffer is considered contiguous exactly if it is either *C-contiguous* or *Fortran contiguous*. Zero-dimensional buffers are C and Fortran contiguous. In one-dimensional arrays, the items must be laid out in memory next to each other, in order of increasing indexes starting from zero. In multidimensional C-contiguous arrays, the last index varies the fastest when visiting items in order of memory address. However, in Fortran contiguous arrays, the first index varies the fastest.

file object An object exposing a file-oriented API (with methods such as `read()` or `write()`) to an underlying resource. Depending on the way it was created, a file object can mediate access to a real on-disk file or to another type of storage or communication device (for example standard input/output, in-memory buffers, sockets, pipes, etc.). File objects are also called *file-like objects* or *streams*.

There are actually three categories of file objects: raw *binary files*, buffered *binary files* and *text files*. Their interfaces are defined in the `io` module. The canonical way to create a file object is by using the `open()` function.

file-like object A synonym for *file object*.

text file A *file object* able to read and write `str` objects. Often, a text file actually accesses a byte-oriented datastream and handles the text encoding automatically. Examples of text files are files opened in text mode (`'r'` or `'w'`), `sys.stdin`, `sys.stdout`, and instances of `io.StringIO`.

See also *binary file* for a file object able to read and write *bytes-like objects*.

path-like object An object representing a file system path. A path-like object is either a `str` or `bytes` object representing a path, or an object implementing the `os.PathLike` protocol. An object that supports the `os.PathLike` protocol can be converted to a `str` or `bytes` file system path by calling the `os.fspath()` function; `os.fsdecode()` and `os.fsencode()` can be used to guarantee a `str` or `bytes` result instead, respectively. Introduced by **PEP 519**.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

X

Z