13.5. `zipfile` — Work with ZIP archives

Source code: Lib/zipfile.py

The ZIP file format is a common archive and compression standard. This module provides tools to create, read, write, append, and list a ZIP file. Any advanced use of this module will require an understanding of the format, as defined in PKZIP Application Note.

This module does not currently handle multi-disk ZIP files. It can handle ZIP files that use the ZIP64 extensions (that is ZIP files that are more than 4 GiB in size). It supports decryption of encrypted files in ZIP archives, but it currently cannot create an encrypted file. Decryption is extremely slow as it is implemented in native Python rather than C.

The module defines the following items:

exception `zipfile.`**`BadZipFile`**

The error raised for bad ZIP files.

New in version 3.2.

exception `zipfile.`**`BadZipfile`**

Alias of `BadZipFile`, for compatibility with older Python versions.

Deprecated since version 3.2.

exception `zipfile.`**`LargeZipFile`**

The error raised when a ZIP file would require ZIP64 functionality but that has not been enabled.

class `zipfile.`**`ZipFile`**

The class for reading and writing ZIP files. See section ZipFile Objects for constructor details.

class `zipfile.`**`PyZipFile`**

Class for creating ZIP archives containing Python libraries.

class `zipfile.`**`ZipInfo`**(*filename='NoName'*, *date_time=(1980*, *1*, *1*, *0*, *0*, *0)*)

Class used to represent information about a member of an archive. Instances of this class are returned by the `getinfo()` and `infolist()` methods of `ZipFile` objects. Most users of the `zipfile` module will not need to create these, but

only use those created by this module. *filename* should be the full name of the archive member, and *date_time* should be a tuple containing six fields which describe the time of the last modification to the file; the fields are described in section ZipInfo Objects.

`zipfile.`**`is_zipfile`**(*filename*)

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

Changed in version 3.1: Support for file and file-like objects.

`zipfile.`**`ZIP_STORED`**

The numeric constant for an uncompressed archive member.

`zipfile.`**`ZIP_DEFLATED`**

The numeric constant for the usual ZIP compression method. This requires the `zlib` module.

`zipfile.`**`ZIP_BZIP2`**

The numeric constant for the BZIP2 compression method. This requires the `bz2` module.

New in version 3.3.

`zipfile.`**`ZIP_LZMA`**

The numeric constant for the LZMA compression method. This requires the `lzma` module.

New in version 3.3.

> **Note:** The ZIP file format specification has included support for bzip2 compression since 2001, and for LZMA compression since 2006. However, some tools (including older Python releases) do not support these compression methods, and may either refuse to process the ZIP file altogether, or fail to extract individual files.

See also:

PKZIP Application Note

Documentation on the ZIP file format by Phil Katz, the creator of the format and algorithms used.

Info-ZIP Home Page

Information about the Info-ZIP project's ZIP archive programs and development libraries.

13.5.1. ZipFile Objects

class `zipfile.`**`ZipFile`**(*file*, *mode='r'*, *compression=ZIP_STORED*, *allowZip64=True*)

Open a ZIP 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 `'a'` and *file* refers to an existing ZIP file, then additional files are added to it. If *file* does not refer to a ZIP file, then a new ZIP archive is appended to the file. This is meant for adding a ZIP archive to another file (such as `python.exe`). If *mode* is `'a'` and the file does not exist at all, it is created. If *mode* is `'r'` or `'a'`, the file should be seekable. *compression* is the ZIP compression method to use when writing the archive, and should be `ZIP_STORED`, `ZIP_DEFLATED`, `ZIP_BZIP2` or `ZIP_LZMA`; unrecognized values will cause `NotImplementedError` to be raised. If `ZIP_DEFLATED`, `ZIP_BZIP2` or `ZIP_LZMA` is specified but the corresponding module (`zlib`, `bz2` or `lzma`) is not available, `RuntimeError` is raised. The default is `ZIP_STORED`. If *allowZip64* is `True` (the default) zipfile will create ZIP files that use the ZIP64 extensions when the zipfile is larger than 4 GiB. If it is false `zipfile` will raise an exception when the ZIP file would require ZIP64 extensions.

If the file is created with mode `'w'`, `'x'` or `'a'` and then `closed` without adding any files to the archive, the appropriate ZIP structures for an empty archive will be written to the file.

ZipFile is also a context manager and therefore supports the `with` statement. In the example, *myzip* is closed after the `with` statement's suite is finished—even if an exception occurs:

```
with ZipFile('spam.zip', 'w') as myzip:
    myzip.write('eggs.txt')
```

New in version 3.2: Added the ability to use `ZipFile` as a context manager.

Changed in version 3.3: Added support for `bzip2` and `lzma` compression.

Changed in version 3.4: ZIP64 extensions are enabled by default.

Changed in version 3.5: Added support for writing to unseekable streams. Added support for the `'x'` mode.

Changed in version 3.6: Previously, a plain `RuntimeError` was raised for unrecognized compression values.

Changed in version 3.6.2: The *file* parameter accepts a path-like object.

`ZipFile.`**`close`**()
Close the archive file. You must call `close()` before exiting your program or essential records will not be written.

`ZipFile.`**`getinfo`**(*name*)
Return a `ZipInfo` object with information about the archive member *name*. Calling `getinfo()` for a name not currently contained in the archive will raise a `KeyError`.

`ZipFile.`**`infolist`**()
Return a list containing a `ZipInfo` object for each member of the archive. The objects are in the same order as their entries in the actual ZIP file on disk if an existing archive was opened.

`ZipFile.`**`namelist`**()
Return a list of archive members by name.

`ZipFile.`**`open`**(*name*, *mode='r'*, *pwd=None*, *, *force_zip64=False*)
Access a member of the archive as a binary file-like object. *name* can be either the name of a file within the archive or a `ZipInfo` object. The *mode* parameter, if included, must be `'r'` (the default) or `'w'`. *pwd* is the password used to decrypt encrypted ZIP files.

`open()` is also a context manager and therefore supports the `with` statement:

```
with ZipFile('spam.zip') as myzip:
    with myzip.open('eggs.txt') as myfile:
        print(myfile.read())
```

With *mode* `'r'` the file-like object (`ZipExtFile`) is read-only and provides the following methods: `read()`, `readline()`, `readlines()`, `__iter__()`, `__next__()`. These objects can operate independently of the ZipFile.

With `mode='w'`, a writable file handle is returned, which supports the `write()` method. While a writable file handle is open, attempting to read or write other files in the ZIP file will raise a `ValueError`.

When writing a file, if the file size is not known in advance but may exceed 2 GiB, pass `force_zip64=True` to ensure that the header format is capable of

supporting large files. If the file size is known in advance, construct a `ZipInfo` object with `file_size` set, and use that as the *name* parameter.

> **Note:** The `open()`, `read()` and `extract()` methods can take a filename or a `ZipInfo` object. You will appreciate this when trying to read a ZIP file that contains members with duplicate names.

Changed in version 3.6: Removed support of `mode='U'`. Use `io.TextIOWrapper` for reading compressed text files in universal newlines mode.

Changed in version 3.6: `open()` can now be used to write files into the archive with the `mode='w'` option.

Changed in version 3.6: Calling `open()` on a closed ZipFile will raise a `ValueError`. Previously, a `RuntimeError` was raised.

`ZipFile.`**`extract`**(*member*, *path=None*, *pwd=None*)

Extract a member from the archive to the current working directory; *member* must be its full name or a `ZipInfo` object. Its file information is extracted as accurately as possible. *path* specifies a different directory to extract to. *member* can be a filename or a `ZipInfo` object. *pwd* is the password used for encrypted files.

Returns the normalized path created (a directory or new file).

> **Note:** If a member filename is an absolute path, a drive/UNC sharepoint and leading (back)slashes will be stripped, e.g.: `///foo/bar` becomes `foo/bar` on Unix, and `C:\foo\bar` becomes `foo\bar` on Windows. And all `".."` components in a member filename will be removed, e.g.: `../../foo../../ba..r` becomes `foo../ba..r`. On Windows illegal characters (`:`, `<`, `>`, `|`, `"`, `?`, and `*`) replaced by underscore (`_`).

Changed in version 3.6: Calling `extract()` on a closed ZipFile will raise a `ValueError`. Previously, a `RuntimeError` was raised.

Changed in version 3.6.2: The *path* parameter accepts a path-like object.

`ZipFile.`**`extractall`**(*path=None*, *members=None*, *pwd=None*)

Extract all members from the archive to the current working directory. *path* specifies a different directory to extract to. *members* is optional and must be a subset of the list returned by `namelist()`. *pwd* is the password used for encrypted files.

Warning: Never extract archives from untrusted sources without prior inspection. It is possible that files are created outside of *path*, e.g. members that have absolute filenames starting with `"/"` or filenames with two dots `".."`. This module attempts to prevent that. See `extract()` note.

Changed in version 3.6: Calling `extractall()` on a closed ZipFile will raise a `ValueError`. Previously, a `RuntimeError` was raised.

Changed in version 3.6.2: The *path* parameter accepts a path-like object.

`ZipFile.`**`printdir`**()

Print a table of contents for the archive to `sys.stdout`.

`ZipFile.`**`setpassword`**(*pwd*)

Set *pwd* as default password to extract encrypted files.

`ZipFile.`**`read`**(*name*, *pwd=None*)

Return the bytes of the file *name* in the archive. *name* is the name of the file in the archive, or a `ZipInfo` object. The archive must be open for read or append. *pwd* is the password used for encrypted files and, if specified, it will override the default password set with `setpassword()`. Calling `read()` on a ZipFile that uses a compression method other than `ZIP_STORED`, `ZIP_DEFLATED`, `ZIP_BZIP2` or `ZIP_LZMA` will raise a `NotImplementedError`. An error will also be raised if the corresponding compression module is not available.

Changed in version 3.6: Calling `read()` on a closed ZipFile will raise a `ValueError`. Previously, a `RuntimeError` was raised.

`ZipFile.`**`testzip`**()

Read all the files in the archive and check their CRC's and file headers. Return the name of the first bad file, or else return `None`.

Changed in version 3.6: Calling `testfile()` on a closed ZipFile will raise a `ValueError`. Previously, a `RuntimeError` was raised.

`ZipFile.`**`write`**(*filename*, *arcname=None*, *compress_type=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). If given, *compress_type* overrides the value given for the *compression* parameter to the constructor for the new entry. The archive must be open with mode `'w'`, `'x'` or `'a'`.

Note: There is no official file name encoding for ZIP files. If you have unicode file names, you must convert them to byte strings in your desired en-

coding before passing them to `write()`. WinZip interprets all file names as encoded in CP437, also known as DOS Latin.

Note: Archive names should be relative to the archive root, that is, they should not start with a path separator.

Note: If `arcname` (or `filename`, if `arcname` is not given) contains a null byte, the name of the file in the archive will be truncated at the null byte.

Changed in version 3.6: Calling `write()` on a ZipFile created with mode `'r'` or a closed ZipFile will raise a `ValueError`. Previously, a `RuntimeError` was raised.

`ZipFile.`**`writestr`**(*zinfo_or_arcname*, *data*[, *compress_type*])

Write the string *data* to the archive; *zinfo_or_arcname* is either the file name it will be given in the archive, or a `ZipInfo` instance. If it's an instance, at least the filename, date, and time must be given. If it's a name, the date and time is set to the current date and time. The archive must be opened with mode `'w'`, `'x'` or `'a'`.

If given, *compress_type* overrides the value given for the *compression* parameter to the constructor for the new entry, or in the *zinfo_or_arcname* (if that is a `ZipInfo` instance).

Note: When passing a `ZipInfo` instance as the *zinfo_or_arcname* parameter, the compression method used will be that specified in the *compress_type* member of the given `ZipInfo` instance. By default, the `ZipInfo` constructor sets this member to `ZIP_STORED`.

Changed in version 3.2: The *compress_type* argument.

Changed in version 3.6: Calling `writestr()` on a ZipFile created with mode `'r'` or a closed ZipFile will raise a `ValueError`. Previously, a `RuntimeError` was raised.

The following data attributes are also available:

`ZipFile.`**`filename`**

Name of the ZIP file.

`ZipFile.`**`debug`**

The level of debug output to use. This may be set from `0` (the default, no output) to `3` (the most output). Debugging information is written to `sys.stdout`.

`ZipFile.`**`comment`**

The comment text associated with the ZIP file. If assigning a comment to a `ZipFile` instance created with mode `'w'`, `'x'` or `'a'`, this should be a string no longer than 65535 bytes. Comments longer than this will be truncated in the written archive when `close()` is called.

13.5.2. PyZipFile Objects

The `PyZipFile` constructor takes the same parameters as the `ZipFile` constructor, and one additional parameter, *optimize*.

class `zipfile.`**`PyZipFile`**(*file*, *mode='r'*, *compression=ZIP_STORED*, *allowZip64=True*, *optimize=-1*)

New in version 3.2: The *optimize* parameter.

Changed in version 3.4: ZIP64 extensions are enabled by default.

Instances have one method in addition to those of `ZipFile` objects:

`writepy`(*pathname*, *basename=''*, *filterfunc=None*)

Search for files `*.py` and add the corresponding file to the archive.

If the *optimize* parameter to `PyZipFile` was not given or `-1`, the corresponding file is a `*.pyc` file, compiling if necessary.

If the *optimize* parameter to `PyZipFile` was `0`, `1` or `2`, only files with that optimization level (see `compile()`) are added to the archive, compiling if necessary.

If *pathname* is a file, the filename must end with `.py`, and just the (corresponding `*.pyc`) file is added at the top level (no path information). If *pathname* is a file that does not end with `.py`, a `RuntimeError` will be raised. If it is a directory, and the directory is not a package directory, then all the files `*.pyc` are added at the top level. If the directory is a package directory, then all `*.pyc` are added under the package name as a file path, and if any subdirectories are package directories, all of these are added recursively.

basename is intended for internal use only.

filterfunc, if given, must be a function taking a single string argument. It will be passed each path (including each individual full file path) before it is added to the archive. If *filterfunc* returns a false value, the path will not be added, and if it is a directory its contents will be ignored. For example, if our

test files are all either in `test` directories or start with the string `test_`, we can use a *filterfunc* to exclude them:

```
>>> zf = PyZipFile('myprog.zip')
>>> def notests(s):
...     fn = os.path.basename(s)
...     return (not (fn == 'test' or fn.startswith('test_')))
>>> zf.writepy('myprog', filterfunc=notests)
```

The `writepy()` method makes archives with file names like this:

```
string.pyc                      # Top level name
test/__init__.pyc               # Package directory
test/testall.pyc                # Module test.testall
test/bogus/__init__.pyc         # Subpackage directory
test/bogus/myfile.pyc           # Submodule test.bogus.myfile
```

New in version 3.4: The *filterfunc* parameter.

Changed in version 3.6.2: The *pathname* parameter accepts a path-like object.

13.5.3. ZipInfo Objects

Instances of the `ZipInfo` class are returned by the `getinfo()` and `infolist()` methods of `ZipFile` objects. Each object stores information about a single member of the ZIP archive.

There is one classmethod to make a `ZipInfo` instance for a filesystem file:

classmethod `ZipInfo.`**`from_file`**(*filename*, *arcname=None*)

Construct a `ZipInfo` instance for a file on the filesystem, in preparation for adding it to a zip file.

filename should be the path to a file or directory on the filesystem.

If *arcname* is specified, it is used as the name within the archive. If *arcname* is not specified, the name will be the same as *filename*, but with any drive letter and leading path separators removed.

New in version 3.6.

Changed in version 3.6.2: The *filename* parameter accepts a path-like object.

Instances have the following methods and attributes:

`ZipInfo.`**`is_dir`**()

Return `True` if this archive member is a directory.

This uses the entry's name: directories should always end with `/`.

New in version 3.6.

`ZipInfo.`**`filename`**

Name of the file in the archive.

`ZipInfo.`**`date_time`**

The time and date of the last modification to the archive member. This is a tuple of six values:

Index	Value
`0`	Year (>= 1980)
`1`	Month (one-based)
`2`	Day of month (one-based)
`3`	Hours (zero-based)
`4`	Minutes (zero-based)
`5`	Seconds (zero-based)

Note: The ZIP file format does not support timestamps before 1980.

`ZipInfo.`**`compress_type`**

Type of compression for the archive member.

`ZipInfo.`**`comment`**

Comment for the individual archive member.

`ZipInfo.`**`extra`**

Expansion field data. The PKZIP Application Note contains some comments on the internal structure of the data contained in this string.

`ZipInfo.`**`create_system`**

System which created ZIP archive.

`ZipInfo.`**`create_version`**

PKZIP version which created ZIP archive.

`ZipInfo.`**`extract_version`**

PKZIP version needed to extract archive.

`ZipInfo.`**`reserved`**
Must be zero.

`ZipInfo.`**`flag_bits`**
ZIP flag bits.

`ZipInfo.`**`volume`**
Volume number of file header.

`ZipInfo.`**`internal_attr`**
Internal attributes.

`ZipInfo.`**`external_attr`**
External file attributes.

`ZipInfo.`**`header_offset`**
Byte offset to the file header.

`ZipInfo.`**`CRC`**
CRC-32 of the uncompressed file.

`ZipInfo.`**`compress_size`**
Size of the compressed data.

`ZipInfo.`**`file_size`**
Size of the uncompressed file.

13.5.4. Command-Line Interface

The `zipfile` module provides a simple command-line interface to interact with ZIP archives.

If you want to create a new ZIP archive, specify its name after the `-c` option and then list the filename(s) that should be included:

```
$ python -m zipfile -c monty.zip spam.txt eggs.txt
```

Passing a directory is also acceptable:

```
$ python -m zipfile -c monty.zip life-of-brian_1979/
```

If you want to extract a ZIP archive into the specified directory, use the `-e` option:

```
$ python -m zipfile -e monty.zip target-dir/
```

For a list of the files in a ZIP archive, use the `-l` option:

```
$ python -m zipfile -l monty.zip
```

13.5.4.1. Command-line options

`-l` `<zipfile>`
List files in a zipfile.

`-c` `<zipfile> <source1> ... <sourceN>`
Create zipfile from source files.

`-e` `<zipfile> <output_dir>`
Extract zipfile into target directory.

`-t` `<zipfile>`
Test whether the zipfile is valid or not.