13.4. `zipfile` — Work with ZIP archives¶

New in version 1.6.

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, or ZIP files which have appended comments (although it correctly handles comments added to individual archive members—for which see the *ZipInfo Objects* documentation). It can handle ZIP files that use the ZIP64 extensions (that is ZIP files that are more than 4 GByte 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.

For other archive formats, see the `bz2`, `gzip`, and `tarfile` modules.

The module defines the following items:

exception `zipfile.BadZipfile`¶

The error raised for bad ZIP files (old name: `zipfile.error`).

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*[, *date_time*]])¶

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`. This module does not currently handle ZIP files which have appended comments.

`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. No other compression methods are currently supported.

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.

13.4.1. ZipFile Objects¶

class `zipfile.ZipFile`(*file*[, *mode*[, *compression*[, *allowZip64*]]])

Open a ZIP file, where *file* can be either a path to a file (a string) or a file-like object. The *mode* parameter should be `'r'` to read an existing file, `'w'` to truncate and write a new file, or `'a'` to append to an existing file. 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`. Using

```
cat myzip.zip >> python.exe
```

also works, and at least **WinZip** can read such files. If *mode* is a and the file does not exist at all, it is created. *compression* is the ZIP compression method to use when writing the archive, and should be `ZIP_STORED` or `ZIP_DEFLATED`; unrecognized values will cause `RuntimeError` to be raised. If `ZIP_DEFLATED`

is specified but the zlib module is not available, RuntimeError is also raised. The default is ZIP_STORED. If *allowZip64* is True zipfile will create ZIP files that use the ZIP64 extensions when the zipfile is larger than 2 GB. If it is false (the default) zipfile will raise an exception when the ZIP file would require ZIP64 extensions. ZIP64 extensions are disabled by default because the default **zip** and **unzip** commands on Unix (the InfoZIP utilities) don't support these extensions.

Changed in version 2.6: If the file does not exist, it is created if the mode is 'a'.

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*[, *pwd*]])¶

Extract a member from the archive as a file-like object (ZipExtFile). *name* is the name of the file in the archive, or a ZipInfo object. The *mode* parameter, if included, must be one of the following: 'r' (the default), 'U', or 'rU'. Choosing 'U' or 'rU' will enable universal newline support in the read-only object. *pwd* is the password used for encrypted files. Calling open() on a closed ZipFile will raise a RuntimeError.

Note

The file-like object is read-only and provides the following methods: read(), readline(), readlines(), __iter__(), next().

Note

If the ZipFile was created by passing in a file-like object as the first argument to the constructor, then the object returned by open() shares the ZipFile's file pointer. Under these circumstances, the object returned by open() should not be used after any additional operations are performed on the ZipFile object. If the ZipFile was created by passing in a string (the filename) as the first argument to the constructor, then open() will create a new file object that will be held by the ZipExtFile, allowing it to operate independently of the ZipFile.

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.

New in version 2.6.

ZipFile.extract(*member*[, *path*[, *pwd*]])¶

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.

New in version 2.6.

ZipFile.extractall([*path*[, *members*[, *pwd*]]])¶

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 "..".

New in version 2.6.

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.

New in version 2.6.

ZipFile.read(*name*[, *pwd*])¶

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 closed ZipFile will raise a `RuntimeError`.

Changed in version 2.6: *pwd* was added, and *name* can now be a `ZipInfo` object.

`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`. Calling `testzip()` on a closed ZipFile will raise a `RuntimeError`.

`ZipFile.write`(*filename*[, *arcname*[, *compress_type*]])¶

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'` or `'a'` – calling `write()` on a ZipFile created with mode `'r'` will raise a `RuntimeError`. Calling `write()` on a closed ZipFile will raise a `RuntimeError`.

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 encoding 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.

`ZipFile.writestr`(*zinfo_or_arcname*, *bytes*)¶

Write the string *bytes* 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'` or `'a'` – calling `writestr()` on a ZipFile created with mode `'r'` will raise a `RuntimeError`. Calling `writestr()` on a closed ZipFile will raise a `RuntimeError`.

Note

When passing a `ZipInfo` instance as the *zinfo_or_acrname* 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`.

The following data attributes are also available:

`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 'a' or 'w', this should be a string no longer than 65535 bytes. Comments longer than this will be truncated in the written archive when `ZipFile.close()` is called.

13.4.2. PyZipFile Objects¶

The `PyZipFile` constructor takes the same parameters as the `ZipFile` constructor. Instances have one method in addition to those of `ZipFile` objects.

`PyZipFile.writepy`(*pathname*[, *basename*])¶

Search for files `*.py` and add the corresponding file to the archive. The corresponding file is a `*.pyo` file if available, else a `*.pyc` file, compiling if necessary. If the pathname is a file, the filename must end with `.py`, and just the (corresponding `*.py[co]`) file is added at the top level (no path information). If the 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 `*.py[co]` are added at the top level. If the directory is a package directory, then all `*.py[co]` 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. The `writepy()` method makes archives with file names like this:

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

13.4.3. 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.

Instances have the following attributes:

`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
1	Month (one-based)
2	Day of month (one-based)
3	Hours (zero-based)
4	Minutes (zero-based)
5	Seconds (zero-based)

`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.

Previous topic

13.3. `bz2` — Compression compatible with **bzip2**

Next topic

13.5. `tarfile` — Read and write tar archive files

This Page

- Show Source