13.2. `gzip` — Support for **gzip** files

Source code: Lib/gzip.py

This module provides a simple interface to compress and decompress files just like the GNU programs **gzip** and **gunzip** would.

The data compression is provided by the `zlib` module.

The `gzip` module provides the `GzipFile` class, as well as the `open()`, `compress()` and `decompress()` convenience functions. The `GzipFile` class reads and writes **gzip**-format files, automatically compressing or decompressing the data so that it looks like an ordinary file object.

Note that additional file formats which can be decompressed by the **gzip** and **gunzip** programs, such as those produced by **compress** and **pack**, are not supported by this module.

The module defines the following items:

`gzip.`**`open`**(*filename*, *mode='rb'*, *compresslevel=9*, *encoding=None*, *errors=None*, *newline=None*)

Open a gzip-compressed file in binary or text mode, returning a file object.

The *filename* argument can be an actual filename (a `str` or `bytes` object), or an existing file object to read from or write to.

The *mode* argument can be any of `'r'`, `'rb'`, `'a'`, `'ab'`, `'w'`, `'wb'`, `'x'` or `'xb'` for binary mode, or `'rt'`, `'at'`, `'wt'`, or `'xt'` for text mode. The default is `'rb'`.

The *compresslevel* argument is an integer from 0 to 9, as for the `GzipFile` constructor.

For binary mode, this function is equivalent to the `GzipFile` constructor: `GzipFile(filename, mode, compresslevel)`. In this case, the *encoding*, *errors* and *newline* arguments must not be provided.

For text mode, a `GzipFile` object is created, and wrapped in an `io.TextIOWrapper` instance with the specified encoding, error handling behavior, and line ending(s).

Changed in version 3.3: Added support for *filename* being a file object, support for text mode, and the *encoding*, *errors* and *newline* arguments.

Changed in version 3.4: Added support for the `'x'`, `'xb'` and `'xt'` modes.

Changed in version 3.6: Accepts a path-like object.

class `gzip.`**`GzipFile`**(*filename=None*, *mode=None*, *compresslevel=9*, *fileobj=None*, *mtime=None*)

Constructor for the `GzipFile` class, which simulates most of the methods of a file object, with the exception of the `truncate()` method. At least one of *fileobj* and *filename* must be given a non-trivial value.

The new class instance is based on *fileobj*, which can be a regular file, an `io.BytesIO` object, or any other object which simulates a file. It defaults to `None`, in which case *filename* is opened to provide a file object.

When *fileobj* is not `None`, the *filename* argument is only used to be included in the **gzip** file header, which may include the original filename of the uncompressed file. It defaults to the filename of *fileobj*, if discernible; otherwise, it defaults to the empty string, and in this case the original filename is not included in the header.

The *mode* argument can be any of `'r'`, `'rb'`, `'a'`, `'ab'`, `'w'`, `'wb'`, `'x'`, or `'xb'`, depending on whether the file will be read or written. The default is the mode of *fileobj* if discernible; otherwise, the default is `'rb'`.

Note that the file is always opened in binary mode. To open a compressed file in text mode, use `open()` (or wrap your `GzipFile` with an `io.TextIOWrapper`).

The *compresslevel* argument is an integer from `0` to `9` controlling the level of compression; `1` is fastest and produces the least compression, and `9` is slowest and produces the most compression. `0` is no compression. The default is `9`.

The *mtime* argument is an optional numeric timestamp to be written to the last modification time field in the stream when compressing. It should only be provided in compression mode. If omitted or `None`, the current time is used. See the `mtime` attribute for more details.

Calling a `GzipFile` object's `close()` method does not close *fileobj*, since you might wish to append more material after the compressed data. This also allows you to pass an `io.BytesIO` object opened for writing as *fileobj*, and retrieve the resulting memory buffer using the `io.BytesIO` object's `getvalue()` method.

`GzipFile` supports the `io.BufferedIOBase` interface, including iteration and the `with` statement. Only the `truncate()` method isn't implemented.

`GzipFile` also provides the following method and attribute:

`peek`(*n*)

Read *n* uncompressed bytes without advancing the file position. At most one single read on the compressed stream is done to satisfy the call. The number of bytes returned may be more or less than requested.

> **Note:** While calling `peek()` does not change the file position of the `GzipFile`, it may change the position of the underlying file object (e.g. if the `GzipFile` was constructed with the *fileobj* parameter).

New in version 3.2.

`mtime`

When decompressing, the value of the last modification time field in the most recently read header may be read from this attribute, as an integer. The initial value before reading any headers is `None`.

All **gzip** compressed streams are required to contain this timestamp field. Some programs, such as **gunzip**, make use of the timestamp. The format is the same as the return value of `time.time()` and the `st_mtime` attribute of the object returned by `os.stat()`.

Changed in version 3.1: Support for the `with` statement was added, along with the *mtime* constructor argument and `mtime` attribute.

Changed in version 3.2: Support for zero-padded and unseekable files was added.

Changed in version 3.3: The `io.BufferedIOBase.read1()` method is now implemented.

Changed in version 3.4: Added support for the `'x'` and `'xb'` modes.

Changed in version 3.5: Added support for writing arbitrary bytes-like objects. The `read()` method now accepts an argument of `None`.

Changed in version 3.6: Accepts a path-like object.

`gzip.`**`compress`**(*data*, *compresslevel=9*)

Compress the *data*, returning a `bytes` object containing the compressed data. *compresslevel* has the same meaning as in the `GzipFile` constructor above.

New in version 3.2.

`gzip.`**`decompress`**(*data*)

Decompress the *data*, returning a `bytes` object containing the uncompressed data.

New in version 3.2.

13.2.1. Examples of usage

Example of how to read a compressed file:

```
import gzip
with gzip.open('/home/joe/file.txt.gz', 'rb') as f:
    file_content = f.read()
```

Example of how to create a compressed GZIP file:

```
import gzip
content = b"Lots of content here"
with gzip.open('/home/joe/file.txt.gz', 'wb') as f:
    f.write(content)
```

Example of how to GZIP compress an existing file:

```
import gzip
import shutil
with open('/home/joe/file.txt', 'rb') as f_in:
    with gzip.open('/home/joe/file.txt.gz', 'wb') as f_out:
        shutil.copyfileobj(f_in, f_out)
```

Example of how to GZIP compress a binary string:

```
import gzip
s_in = b"Lots of content here"
s_out = gzip.compress(s_in)
```

See also:

Module `zlib`
The basic data compression module needed to support the **gzip** file format.