

## NAME

IO::Zlib - IO:: style interface to *Compress::Zlib*

## SYNOPSIS

With any version of Perl 5 you can use the basic OO interface:

```
use IO::Zlib;

$fh = new IO::Zlib;
if ($fh->open("file.gz", "rb")) {
    print <$fh>;
    $fh->close;
}

$fh = IO::Zlib->new("file.gz", "wb9");
if (defined $fh) {
    print $fh "bar\n";
    $fh->close;
}

$fh = IO::Zlib->new("file.gz", "rb");
if (defined $fh) {
    print <$fh>;
    undef $fh;          # automatically closes the file
}
```

With Perl 5.004 you can also use the TIEHANDLE interface to access compressed files just like ordinary files:

```
use IO::Zlib;

tie *FILE, 'IO::Zlib', "file.gz", "wb";
print FILE "line 1\nline2\n";

tie *FILE, 'IO::Zlib', "file.gz", "rb";
while (<FILE>) { print "LINE: ", $_ };
```

## DESCRIPTION

IO::Zlib provides an IO:: style interface to *Compress::Zlib* and hence to gzip/zlib compressed files. It provides many of the same methods as the *IO::Handle* interface.

Starting from IO::Zlib version 1.02, IO::Zlib can also use an external *gzip* command. The default behaviour is to try to use an external *gzip* if no *Compress::Zlib* can be loaded, unless explicitly disabled by

```
use IO::Zlib qw(:gzip_external 0);
```

If explicitly enabled by

```
use IO::Zlib qw(:gzip_external 1);
```

then the external *gzip* is used **instead** of *Compress::Zlib*.

## CONSTRUCTOR

`new ( [ARGS] )`

Creates an `IO::Zlib` object. If it receives any parameters, they are passed to the method `open`; if the open fails, the object is destroyed. Otherwise, it is returned to the caller.

## OBJECT METHODS

`open ( FILENAME, MODE )`

`open` takes two arguments. The first is the name of the file to open and the second is the open mode. The mode can be anything acceptable to *Compress::Zlib* and by extension anything acceptable to *zlib* (that basically means POSIX `fopen()` style mode strings plus an optional number to indicate the compression level).

`opened`

Returns true if the object currently refers to a opened file.

`close`

Close the file associated with the object and disassociate the file from the handle. Done automatically on destroy.

`getc`

Return the next character from the file, or undef if none remain.

`getline`

Return the next line from the file, or undef on end of string. Can safely be called in an array context. Currently ignores `$/` (`$INPUT_RECORD_SEPARATOR` or `$RS` when *English* is in use) and treats lines as delimited by `"\n"`.

`getlines`

Get all remaining lines from the file. It will croak() if accidentally called in a scalar context.

`print ( ARGS... )`

Print ARGS to the file.

`read ( BUF, NBYTES, [OFFSET] )`

Read some bytes from the file. Returns the number of bytes actually read, 0 on end-of-file, undef on error.

`eof`

Returns true if the handle is currently positioned at end of file?

`seek ( OFFSET, WHENCE )`

Seek to a given position in the stream. Not yet supported.

`tell`

Return the current position in the stream, as a numeric offset. Not yet supported.

`setpos ( POS )`

Set the current position, using the opaque value returned by `getpos ( )`. Not yet supported.

`getpos ( POS )`

Return the current position in the string, as an opaque object. Not yet supported.

## USING THE EXTERNAL GZIP

If the external *gzip* is used, the following `opens` are used:

```
open(FH, "gzip -dc $filename |") # for read opens
open(FH, " | gzip > $filename")  # for write opens
```

You can modify the 'commands' for example to hardwire an absolute path by e.g.

```
use IO::Zlib ':gzip_read_open'  => '/some/where/gunzip -c %s |';
use IO::Zlib ':gzip_write_open' => '| /some/where/gzip.exe > %s';
```

The `%s` is expanded to be the filename (`sprintf` is used, so be careful to escape any other `%` signs). The 'commands' are checked for sanity - they must contain the `%s`, and the read open must end with the pipe sign, and the write open must begin with the pipe sign.

## CLASS METHODS

`has_Compress_Zlib`

Returns true if `Compress::Zlib` is available. Note that this does not mean that `Compress::Zlib` is being used: see *gzip\_external* and *gzip\_used*.

`gzip_external`

Undef if an external *gzip* **can** be used if `Compress::Zlib` is not available (see *has\_Compress\_Zlib*), true if an external *gzip* is explicitly used, false if an external *gzip* must not be used. See *gzip\_used*.

`gzip_used`

True if an external *gzip* is being used, false if not.

`gzip_read_open`

Return the 'command' being used for opening a file for reading using an external *gzip*.

`gzip_write_open`

Return the 'command' being used for opening a file for writing using an external *gzip*.

## DIAGNOSTICS

`IO::Zlib::getlines`: must be called in list context

If you want read lines, you must read in list context.

`IO::Zlib::gzopen_external`: mode `'...'` is illegal

Use only modes `'rb'` or `'wb'` or `/wb[1-9]/`.

`IO::Zlib::import`: `'...'` is illegal

The known import symbols are the `:gzip_external`, `:gzip_read_open`, and `:gzip_write_open`. Anything else is not recognized.

`IO::Zlib::import`: `'gzip_external'` requires an argument

The `:gzip_external` requires one boolean argument.

`IO::Zlib::import`: `'gzip_read_open'` requires an argument

The `:gzip_read_open` requires one string argument.

`IO::Zlib::import`: `'gzip_read'` `'...'` is illegal

The `:gzip_read_open` argument must end with the pipe sign (`|`) and have the `%s` for the filename. See *USING THE EXTERNAL GZIP*.

`IO::Zlib::import`: `'gzip_write_open'` requires an argument

The `:gzip_write_open` requires one string argument.

IO::Zlib::import: 'gzip\_write\_open' '...' is illegal

The `:gzip_write_open` argument must begin with the pipe sign (`|`) and have the `%s` for the filename. An output redirect (`>`) is also often a good idea, depending on your operating system shell syntax. See *USING THE EXTERNAL GZIP*.

IO::Zlib::import: no Compress::Zlib and no external gzip

Given that we failed to load `Compress::Zlib` and that the use of an external *gzip* was disabled, IO::Zlib has not much chance of working.

IO::Zlib::open: needs a filename

No filename, no open.

IO::Zlib::READ: NBYTES must be specified

We must know how much to read.

IO::Zlib::WRITE: too long LENGTH

The LENGTH must be less than or equal to the buffer size.

## SEE ALSO

*perlfunc*, "I/O Operators" in *perlop*, *IO::Handle*, *Compress::Zlib*

## HISTORY

Created by Tom Hughes <*tom@compton.nu*>.

Support for external gzip added by Jarkko Hietaniemi <*jhi@iki.fi*>.

## COPYRIGHT

Copyright (c) 1998-2004 Tom Hughes <*tom@compton.nu*>. All rights reserved. This program is free software; you can redistribute it and/or modify it under the same terms as Perl itself.