

## NAME

IO::Compress::Zip - Write zip files/buffers

## SYNOPSIS

```
use IO::Compress::Zip qw(zip $ZipError) ;

my $status = zip $input => $output [,OPTS]
    or die "zip failed: $ZipError\n";

my $z = new IO::Compress::Zip $output [,OPTS]
    or die "zip failed: $ZipError\n";

$z->print($string);
$z->printf($format, $string);
$z->write($string);
$z->syswrite($string [, $length, $offset]);
$z->flush();
$z->tell();
$z->eof();
$z->seek($position, $whence);
$z->binmode();
$z->fileno();
$z->opened();
$z->autoflush();
$z->input_line_number();
$z->newStream( [OPTS] );

$z->deflateParams();

$z->close() ;

$ZipError ;

# IO::File mode

print $z $string;
printf $z $format, $string;
tell $z
eof $z
seek $z, $position, $whence
binmode $z
fileno $z
close $z ;
```

## DESCRIPTION

This module provides a Perl interface that allows writing zip compressed data to files or buffer.

The primary purpose of this module is to provide streaming write access to zip files and buffers. It is not a general-purpose file archiver. If that is what you want, check out `Archive::Zip`.

At present three compression methods are supported by IO::Compress::Zip, namely Store (no compression at all), Deflate and Bzip2.

Note that to create Bzip2 content, the module `IO::Compress::Bzip2` must be installed.

For reading zip files/buffers, see the companion module `IO::Uncompress::Unzip`.

## Functional Interface

A top-level function, `zip`, is provided to carry out "one-shot" compression between buffers and/or files. For finer control over the compression process, see the *OO Interface* section.

```
use IO::Compress::Zip qw(zip $ZipError) ;

zip $input => $output [,OPTS]
    or die "zip failed: $ZipError\n";
```

The functional interface needs Perl5.005 or better.

### `zip $input => $output [, OPTS]`

`zip` expects at least two parameters, `$input` and `$output`.

#### The `$input` parameter

The parameter, `$input`, is used to define the source of the uncompressed data.

It can take one of the following forms:

##### A filename

If the `$input` parameter is a simple scalar, it is assumed to be a filename. This file will be opened for reading and the input data will be read from it.

##### A filehandle

If the `$input` parameter is a filehandle, the input data will be read from it. The string '-' can be used as an alias for standard input.

##### A scalar reference

If `$input` is a scalar reference, the input data will be read from `$$input`.

##### An array reference

If `$input` is an array reference, each element in the array must be a filename.

The input data will be read from each file in turn.

The complete array will be walked to ensure that it only contains valid filenames before any data is compressed.

##### An Input FileGlob string

If `$input` is a string that is delimited by the characters "<" and ">" `zip` will assume that it is an *input fileglob string*. The input is the list of files that match the fileglob.

If the fileglob does not match any files ...

See *File::GlobMapper* for more details.

If the `$input` parameter is any other type, `undef` will be returned.

In addition, if `$input` is a simple filename, the default values for the `Name`, `Time`, `ExtAttr` and `exTime` options will be sourced from that file.

If you do not want to use these defaults they can be overridden by explicitly setting the `Name`, `Time`, `ExtAttr` and `exTime` options or by setting the `Minimal` parameter.

## The \$output parameter

The parameter `$output` is used to control the destination of the compressed data. This parameter can take one of these forms.

### A filename

If the `$output` parameter is a simple scalar, it is assumed to be a filename. This file will be opened for writing and the compressed data will be written to it.

### A filehandle

If the `$output` parameter is a filehandle, the compressed data will be written to it. The string `'-'` can be used as an alias for standard output.

### A scalar reference

If `$output` is a scalar reference, the compressed data will be stored in `$$output`.

### An Array Reference

If `$output` is an array reference, the compressed data will be pushed onto the array.

### An Output FileGlob

If `$output` is a string that is delimited by the characters `"<"` and `">"` `zip` will assume that it is an *output fileglob string*. The output is the list of files that match the fileglob.

When `$output` is an fileglob string, `$input` must also be a fileglob string. Anything else is an error.

If the `$output` parameter is any other type, `undef` will be returned.

## Notes

When `$input` maps to multiple files/buffers and `$output` is a single file/buffer the input files/buffers will each be stored in `$output` as a distinct entry.

## Optional Parameters

Unless specified below, the optional parameters for `zip`, `OPTS`, are the same as those used with the OO interface defined in the *Constructor Options* section below.

`AutoClose => 0|1`

This option applies to any input or output data streams to `zip` that are filehandles.

If `AutoClose` is specified, and the value is true, it will result in all input and/or output filehandles being closed once `zip` has completed.

This parameter defaults to 0.

`BinModeIn => 0|1`

When reading from a file or filehandle, set `binmode` before reading.

Defaults to 0.

`Append => 0|1`

TODO

## Examples

To read the contents of the file `file1.txt` and write the compressed data to the file `file1.txt.zip`.

```
use strict ;
use warnings ;
use IO::Compress::Zip qw(zip $ZipError) ;
```

```
my $input = "file1.txt";
zip $input => "$input.zip"
    or die "zip failed: $ZipError\n";
```

To read from an existing Perl filehandle, `$input`, and write the compressed data to a buffer, `$buffer`.

```
use strict ;
use warnings ;
use IO::Compress::Zip qw(zip $ZipError) ;
use IO::File ;

my $input = new IO::File "<file1.txt"
    or die "Cannot open 'file1.txt': $!\n" ;
my $buffer ;
zip $input => \$buffer
    or die "zip failed: $ZipError\n";
```

To compress all files in the directory `/my/home` that match `*.txt` and store the compressed data in the same directory

```
use strict ;
use warnings ;
use IO::Compress::Zip qw(zip $ZipError) ;

zip '</my/home/*.txt>' => '<*.zip>'
    or die "zip failed: $ZipError\n";
```

and if you want to compress each file one at a time, this will do the trick

```
use strict ;
use warnings ;
use IO::Compress::Zip qw(zip $ZipError) ;

for my $input ( glob "/my/home/*.txt" )
{
    my $output = "$input.zip" ;
    zip $input => $output
        or die "Error compressing '$input': $ZipError\n";
}
```

## OO Interface

### Constructor

The format of the constructor for `IO::Compress::Zip` is shown below

```
my $z = new IO::Compress::Zip $output [,OPTS]
    or die "IO::Compress::Zip failed: $ZipError\n";
```

It returns an `IO::Compress::Zip` object on success and undef on failure. The variable `$ZipError` will contain an error message on failure.

If you are running Perl 5.005 or better the object, `$z`, returned from `IO::Compress::Zip` can be used exactly like an `IO::File` filehandle. This means that all normal output file operations can be carried out with `$z`. For example, to write to a compressed file/buffer you can use either of these forms

```
$z->print("hello world\n");  
print $z "hello world\n";
```

The mandatory parameter `$output` is used to control the destination of the compressed data. This parameter can take one of these forms.

**A filename**

If the `$output` parameter is a simple scalar, it is assumed to be a filename. This file will be opened for writing and the compressed data will be written to it.

**A filehandle**

If the `$output` parameter is a filehandle, the compressed data will be written to it. The string `'-'` can be used as an alias for standard output.

**A scalar reference**

If `$output` is a scalar reference, the compressed data will be stored in `$$output`.

If the `$output` parameter is any other type, `IO::Compress::Zip::new` will return `undef`.

## Constructor Options

`OPTS` is any combination of the following options:

`AutoClose => 0|1`

This option is only valid when the `$output` parameter is a filehandle. If specified, and the value is true, it will result in the `$output` being closed once either the `close` method is called or the `IO::Compress::Zip` object is destroyed.

This parameter defaults to 0.

`Append => 0|1`

Opens `$output` in append mode.

The behaviour of this option is dependent on the type of `$output`.

**\* A Buffer**

If `$output` is a buffer and `Append` is enabled, all compressed data will be append to the end of `$output`. Otherwise `$output` will be cleared before any data is written to it.

**\* A Filename**

If `$output` is a filename and `Append` is enabled, the file will be opened in append mode. Otherwise the contents of the file, if any, will be truncated before any compressed data is written to it.

**\* A Filehandle**

If `$output` is a filehandle, the file pointer will be positioned to the end of the file via a call to `seek` before any compressed data is written to it. Otherwise the file pointer will not be moved.

This parameter defaults to 0.

`Name => $string`

Stores the contents of `$string` in the zip filename header field. If `Name` is not specified, no zip filename field will be created.

`Time => $number`

Sets the last modified time field in the zip header to `$number`.

This field defaults to the time the `IO::Compress::Zip` object was created if this option is

not specified.

```
ExtAttr => $attr
```

This option controls the "external file attributes" field in the central header of the zip file. This is a 4 byte field.

If you are running a Unix derivative this value defaults to

```
0666 << 16
```

This should allow read/write access to any files that are extracted from the zip file/buffer.

For all other systems it defaults to 0.

```
exTime => [$atime, $mtime, $ctime]
```

This option expects an array reference with exactly three elements: `$atime`, `$mtime` and `$ctime`. These correspond to the last access time, last modification time and creation time respectively.

It uses these values to set the extended timestamp field (ID is "UT") in the local zip header using the three values, `$atime`, `$mtime`, `$ctime`. In addition it sets the extended timestamp field in the central zip header using `$mtime`.

If any of the three values is `undef` that time value will not be used. So, for example, to set only the `$mtime` you would use this

```
exTime => [undef, $mtime, undef]
```

If the `Minimal` option is set to true, this option will be ignored.

By default no extended time field is created.

```
exUnix2 => [$uid, $gid]
```

This option expects an array reference with exactly two elements: `$uid` and `$gid`. These values correspond to the numeric user ID and group ID of the owner of the files respectively.

When the `exUnix2` option is present it will trigger the creation of a Unix2 extra field (ID is "Ux") in the local zip. This will be populated with `$uid` and `$gid`. In addition an empty Unix2 extra field will also be created in the central zip header

If the `Minimal` option is set to true, this option will be ignored.

By default no Unix2 extra field is created.

```
Comment => $comment
```

Stores the contents of `$comment` in the Central File Header of the zip file.

By default, no comment field is written to the zip file.

```
ZipComment => $comment
```

Stores the contents of `$comment` in the End of Central Directory record of the zip file.

By default, no comment field is written to the zip file.

```
Method => $method
```

Controls which compression method is used. At present three compression methods are supported, namely Store (no compression at all), Deflate and Bzip2.

The symbols, `ZIP_CM_STORE`, `ZIP_CM_DEFLATE` and `ZIP_CM_BZIP2` are used to select the compression method.

These constants are not imported by `IO::Compress::Zip` by default.

```
use IO::Compress::Zip qw(:zip_method);
use IO::Compress::Zip qw(:constants);
use IO::Compress::Zip qw(:all);
```

Note that to create Bzip2 content, the module `IO::Compress::Bzip2` must be installed. A fatal error will be thrown if you attempt to create Bzip2 content when `IO::Compress::Bzip2` is not available.

The default method is `ZIP_CM_DEFLATE`.

`Stream => 0|1`

This option controls whether the zip file/buffer output is created in streaming mode.

Note that when outputting to a file with streaming mode disabled (`Stream` is 0), the output file must be seekable.

The default is 1.

`Zip64 => 0|1`

Create a Zip64 zip file/buffer. This option should only be used if you want to store files larger than 4 Gig.

If you intend to manipulate the Zip64 zip files created with this module using an external zip/unzip make sure that it supports streaming Zip64.

In particular, if you are using Info-Zip you need to have zip version 3.x or better to update a Zip64 archive and unzip version 6.x to read a zip64 archive. At the time of writing both are beta status.

When the `Zip64` option is enabled, the `Stream` option *must* be enabled as well.

The default is 0.

`TextFlag => 0|1`

This parameter controls the setting of a bit in the zip central header. It is used to signal that the data stored in the zip file/buffer is probably text.

The default is 0.

`ExtraFieldLocal => $data=`~~`item`~~ `ExtraFieldCentral => $data`

The `ExtraFieldLocal` option is used to store additional metadata in the local header for the zip file/buffer. The `ExtraFieldCentral` does the same for the matching central header.

An extra field consists of zero or more subfields. Each subfield consists of a two byte header followed by the subfield data.

The list of subfields can be supplied in any of the following formats

```
ExtraFieldLocal => [$id1, $data1,
                    $id2, $data2,
                    ...
                  ]

ExtraFieldLocal => [ [$id1 => $data1],
                    [$id2 => $data2],
                    ...
                  ]

ExtraFieldLocal => { $id1 => $data1,
                    $id2 => $data2,
                    ...
                  }
```

Where `$id1`, `$id2` are two byte subfield ID's.

If you use the hash syntax, you have no control over the order in which the `ExtraSubFields` are stored, plus you cannot have `SubFields` with duplicate ID.

Alternatively the list of subfields can be supplied as a scalar, thus

```
ExtraField => $rawdata
```

The Extended Time field (ID "UT"), set using the `exTime` option, and the Unix2 extra field (ID "Ux"), set using the `exUnix2` option, are examples of extra fields.

If the `Minimal` option is set to true, this option will be ignored.

The maximum size of an extra field is 65535 bytes.

```
Minimal => 1|0
```

If specified, this option will disable the creation of all extra fields in the zip local and central headers. So the `exTime`, `exUnix2`, `ExtraFieldLocal` and `ExtraFieldCentral` options will be ignored.

This parameter defaults to 0.

```
BlockSize100K => number
```

Specify the number of 100K blocks bzip2 uses during compression.

Valid values are from 1 to 9, where 9 is best compression.

This option is only valid if the `Method` is `ZIP_CM_BZIP2`. It is ignored otherwise.

The default is 1.

```
WorkFactor => number
```

Specifies how much effort bzip2 should take before resorting to a slower fallback compression algorithm.

Valid values range from 0 to 250, where 0 means use the default value 30.

This option is only valid if the `Method` is `ZIP_CM_BZIP2`. It is ignored otherwise.

The default is 0.

**-Level**

Defines the compression level used by zlib. The value should either be a number between 0 and 9 (0 means no compression and 9 is maximum compression), or one of the symbolic constants defined below.

```
Z_NO_COMPRESSION
Z_BEST_SPEED
Z_BEST_COMPRESSION
Z_DEFAULT_COMPRESSION
```

The default is `Z_DEFAULT_COMPRESSION`.

Note, these constants are not imported by `IO::Compress::Zip` by default.

```
use IO::Compress::Zip qw(:strategy);
use IO::Compress::Zip qw(:constants);
use IO::Compress::Zip qw(:all);
```

**-Strategy**

Defines the strategy used to tune the compression. Use one of the symbolic constants defined below.

```
Z_FILTERED
Z_HUFFMAN_ONLY
Z_RLE
Z_FIXED
Z_DEFAULT_STRATEGY
```



The default is Z\_DEFAULT\_STRATEGY.

Strict => 0|1

This is a placeholder option.

## Examples

TODO

## Methods

### print

Usage is

```
$z->print($data)
print $z $data
```

Compresses and outputs the contents of the `$data` parameter. This has the same behaviour as the `print` built-in.

Returns true if successful.

### printf

Usage is

```
$z->printf($format, $data)
printf $z $format, $data
```

Compresses and outputs the contents of the `$data` parameter.

Returns true if successful.

### syswrite

Usage is

```
$z->syswrite $data
$z->syswrite $data, $length
$z->syswrite $data, $length, $offset
```

Compresses and outputs the contents of the `$data` parameter.

Returns the number of uncompressed bytes written, or `undef` if unsuccessful.

### write

Usage is

```
$z->write $data
$z->write $data, $length
$z->write $data, $length, $offset
```

Compresses and outputs the contents of the `$data` parameter.

Returns the number of uncompressed bytes written, or `undef` if unsuccessful.

### flush

Usage is

```
$z->flush;
$z->flush($flush_type);
```

Flushes any pending compressed data to the output file/buffer.

This method takes an optional parameter, `$flush_type`, that controls how the flushing will be carried out. By default the `$flush_type` used is `Z_FINISH`. Other valid values for `$flush_type` are `Z_NO_FLUSH`, `Z_SYNC_FLUSH`, `Z_FULL_FLUSH` and `Z_BLOCK`. It is strongly recommended that you only set the `flush_type` parameter if you fully understand the implications of what it does - overuse of `flush` can seriously degrade the level of compression achieved. See the `zlib` documentation for details.

Returns true on success.

## **tell**

Usage is

```
$z->tell()  
tell $z
```

Returns the uncompressed file offset.

## **eof**

Usage is

```
$z->eof();  
eof($z);
```

Returns true if the `close` method has been called.

## **seek**

```
$z->seek($position, $whence);  
seek($z, $position, $whence);
```

Provides a sub-set of the `seek` functionality, with the restriction that it is only legal to seek forward in the output file/buffer. It is a fatal error to attempt to seek backward.

Empty parts of the file/buffer will have NULL (0x00) bytes written to them.

The `$whence` parameter takes one the usual values, namely `SEEK_SET`, `SEEK_CUR` or `SEEK_END`.

Returns 1 on success, 0 on failure.

## **binmode**

Usage is

```
$z->binmode  
binmode $z ;
```

This is a noop provided for completeness.

## **opened**

```
$z->opened()
```

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

## **autoflush**

```
my $prev = $z->autoflush()  
my $prev = $z->autoflush(EXPR)
```

If the `$z` object is associated with a file or a filehandle, this method returns the current autoflush setting for the underlying filehandle. If `EXPR` is present, and is non-zero, it will enable flushing after every write/print operation.

If `$z` is associated with a buffer, this method has no effect and always returns `undef`.

**Note** that the special variable `$|` **cannot** be used to set or retrieve the autoflush setting.

### input\_line\_number

```
$z->input_line_number()  
$z->input_line_number(EXPR)
```

This method always returns `undef` when compressing.

### fileno

```
$z->fileno()  
fileno($z)
```

If the `$z` object is associated with a file or a filehandle, `fileno` will return the underlying file descriptor. Once the `close` method is called `fileno` will return `undef`.

If the `$z` object is associated with a buffer, this method will return `undef`.

### close

```
$z->close() ;  
close $z ;
```

Flushes any pending compressed data and then closes the output file/buffer.

For most versions of Perl this method will be automatically invoked if the `IO::Compress::Zip` object is destroyed (either explicitly or by the variable with the reference to the object going out of scope). The exceptions are Perl versions 5.005 through 5.00504 and 5.8.0. In these cases, the `close` method will be called automatically, but not until global destruction of all live objects when the program is terminating.

Therefore, if you want your scripts to be able to run on all versions of Perl, you should call `close` explicitly and not rely on automatic closing.

Returns true on success, otherwise 0.

If the `AutoClose` option has been enabled when the `IO::Compress::Zip` object was created, and the object is associated with a file, the underlying file will also be closed.

### newStream([OPTS])

Usage is

```
$z->newStream( [OPTS] )
```

Closes the current compressed data stream and starts a new one.

`OPTS` consists of any of the the options that are available when creating the `$z` object.

See the *Constructor Options* section for more details.

### deflateParams

Usage is

```
$z->deflateParams
```

TODO

## Importing

A number of symbolic constants are required by some methods in `IO::Compress::Zip`. None are imported by default.

`:all`

Imports `zip`, `$ZipError` and all symbolic constants that can be used by `IO::Compress::Zip`. Same as doing this

```
use IO::Compress::Zip qw(zip $ZipError :constants) ;
```

`:constants`

Import all symbolic constants. Same as doing this

```
use IO::Compress::Zip qw(:flush :level :strategy :zip_method) ;
```

`:flush`

These symbolic constants are used by the `flush` method.

```
Z_NO_FLUSH
Z_PARTIAL_FLUSH
Z_SYNC_FLUSH
Z_FULL_FLUSH
Z_FINISH
Z_BLOCK
```

`:level`

These symbolic constants are used by the `Level` option in the constructor.

```
Z_NO_COMPRESSION
Z_BEST_SPEED
Z_BEST_COMPRESSION
Z_DEFAULT_COMPRESSION
```

`:strategy`

These symbolic constants are used by the `Strategy` option in the constructor.

```
Z_FILTERED
Z_HUFFMAN_ONLY
Z_RLE
Z_FIXED
Z_DEFAULT_STRATEGY
```

`:zip_method`

These symbolic constants are used by the `Method` option in the constructor.

```
ZIP_CM_STORE
ZIP_CM_DEFLATE
ZIP_CM_BZIP2
```

## EXAMPLES

## Apache::GZip Revisited

See *IO::Compress::FAQ*

## Working with Net::FTP

See *IO::Compress::FAQ*

## SEE ALSO

*Compress::Zlib*, *IO::Compress::Gzip*, *IO::Uncompress::Gunzip*, *IO::Compress::Deflate*, *IO::Uncompress::Inflate*, *IO::Compress::RawDeflate*, *IO::Uncompress::RawInflate*, *IO::Compress::Bzip2*, *IO::Uncompress::Bunzip2*, *IO::Compress::Lzop*, *IO::Uncompress::UnLzop*, *IO::Compress::Lzf*, *IO::Uncompress::UnLzf*, *IO::Uncompress::AnyInflate*, *IO::Uncompress::AnyUncompress*

*Compress::Zlib::FAQ*

*File::GlobMapper*, *Archive::Zip*, *Archive::Tar*, *IO::Zlib*

For RFC 1950, 1951 and 1952 see <http://www.faqs.org/rfcs/rfc1950.html>, <http://www.faqs.org/rfcs/rfc1951.html> and <http://www.faqs.org/rfcs/rfc1952.html>

The *zlib* compression library was written by Jean-loup Gailly [gzip@prep.ai.mit.edu](mailto:gzip@prep.ai.mit.edu) and Mark Adler [madler@alumni.caltech.edu](mailto:madler@alumni.caltech.edu).

The primary site for the *zlib* compression library is <http://www.zlib.org>.

The primary site for *gzip* is <http://www.gzip.org>.

## AUTHOR

This module was written by Paul Marquess, [pmqs@cpan.org](mailto:pmqs@cpan.org).

## MODIFICATION HISTORY

See the Changes file.

## COPYRIGHT AND LICENSE

Copyright (c) 2005-2009 Paul Marquess. All rights reserved.

This program is free software; you can redistribute it and/or modify it under the same terms as Perl itself.