NAME

IO::Uncompress::Unzip - Read zip files/buffers

SYNOPSIS

```
use IO::Uncompress::Unzip qw(unzip $UnzipError) ;

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

my $z = new IO::Uncompress::Unzip $input [OPTS]
    or die "unzip failed: $UnzipError\n";

$status = $z->read($buffer)
$status = $z->read($buffer, $length)
$status = $z->read($buffer, $length, $offset)
$line = $z->getline()
$char = $z->getc()
$char = $z->ungetc()
$char = $z->opened()

$status = $z->inflateSync()

$data = $z->trailingData()
$status = $z->nextStream()
$data = $z->getHeaderInfo()
$z->tell()
$z->seek($position, $whence)
$z->binmode()
$z->fileno()
$z->eof()
$z->close()

$UnzipError ;

# IO::File mode

<$z>
read($z, $buffer);
read($z, $buffer, $length);
read($z, $buffer, $length, $offset);
tell($z)
seek($z, $position, $whence)
binmode($z)
fileno($z)
eof($z)
close($z)
```

DESCRIPTION

This module provides a Perl interface that allows the reading of zlib files/buffers.

For writing zip files/buffers, see the companion module IO::Compress::Zip.

Functional Interface

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

```
use IO::Uncompress::Unzip qw(unzip $UnzipError) ;

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

The functional interface needs Perl5.005 or better.

unzip $input => $output [, OPTS]

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

The $input parameter

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

An Input FileGlob string

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

See *File::GlobMapper* for more details.

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

The $output parameter

The parameter `$output` is used to control the destination of the uncompressed 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 uncompressed data will be written to it.

A filehandle

If the `$output` parameter is a filehandle, the uncompressed 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 uncompressed data will be stored in `$$output`.

An Array Reference

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

An Output FileGlob

If `$output` is a string that is delimited by the characters "<" and ">" `unzip` 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.

See *File::GlobMapper* for more details.

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

Notes

When `$input` maps to multiple compressed files/buffers and `$output` is a single file/buffer, after uncompression `$output` will contain a concatenation of all the uncompressed data from each of the input files/buffers.

Optional Parameters

Unless specified below, the optional parameters for `unzip`, `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 `unzip` 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 `unzip` has completed.

This parameter defaults to 0.

`BinModeOut => 0|1`

When writing to a file or filehandle, set `binmode` before writing to the file.

Defaults to 0.

`Append => 0|1`

The behaviour of this option is dependent on the type of output data stream.

* A Buffer

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

* A Filename

  If `Append` is enabled, the file will be opened in append mode. Otherwise the contents of the file, if any, will be truncated before any uncompressed data is written to it.

* A Filehandle

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

When `Append` is specified, and set to true, it will *append* all uncompressed data to the output data stream.

So when the output is a filehandle it will carry out a seek to the eof before writing any

uncompressed data. If the output is a filename, it will be opened for appending. If the output is a buffer, all uncompressed data will be appended to the existing buffer.

Conversely when `Append` is not specified, or it is present and is set to false, it will operate as follows.

When the output is a filename, it will truncate the contents of the file before writing any uncompressed data. If the output is a filehandle its position will not be changed. If the output is a buffer, it will be wiped before any uncompressed data is output.

Defaults to 0.

`MultiStream => 0|1`

If the input file/buffer contains multiple compressed data streams, this option will uncompress the whole lot as a single data stream.

Defaults to 0.

`TrailingData => $scalar`

Returns the data, if any, that is present immediately after the compressed data stream once uncompression is complete.

This option can be used when there is useful information immediately following the compressed data stream, and you don't know the length of the compressed data stream.

If the input is a buffer, `trailingData` will return everything from the end of the compressed data stream to the end of the buffer.

If the input is a filehandle, `trailingData` will return the data that is left in the filehandle input buffer once the end of the compressed data stream has been reached. You can then use the filehandle to read the rest of the input file.

Don't bother using `trailingData` if the input is a filename.

If you know the length of the compressed data stream before you start uncompressing, you can avoid having to use `trailingData` by setting the `InputLength` option.

Examples

Say you have a zip file, `file1.zip`, that only contains a single member, you can read it and write the uncompressed data to the file `file1.txt` like this.

```
use strict ;
use warnings ;
use IO::Uncompress::Unzip qw(unzip $UnzipError) ;

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

If you have a zip file that contains multiple members and want to read a specific member from the file, say `"data1"`, use the `Name` option

```
use strict ;
use warnings ;
use IO::Uncompress::Unzip qw(unzip $UnzipError) ;

my $input = "file1.zip";
my $output = "file1.txt";
unzip $input => $output, Name => "data1"
    or die "unzip failed: $UnzipError\n";
```

Alternatively, if you want to read the "data1" member into memory, use a scalar reference for the output partameter.

```
use strict ;
use warnings ;
use IO::Uncompress::Unzip qw(unzip $UnzipError) ;

my $input = "file1.zip";
my $output ;
unzip $input => \$output, Name => "data1"
    or die "unzip failed: $UnzipError\n";
# $output now contains the uncompressed data
```

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

```
use strict ;
use warnings ;
use IO::Uncompress::Unzip qw(unzip $UnzipError) ;
use IO::File ;

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

OO Interface

Constructor

The format of the constructor for IO::Uncompress::Unzip is shown below

```
my $z = new IO::Uncompress::Unzip $input [OPTS]
    or die "IO::Uncompress::Unzip failed: $UnzipError\n";
```

Returns an IO::Uncompress::Unzip object on success and undef on failure. The variable $UnzipError will contain an error message on failure.

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

```
$line = $z->getline();
$line = <$z>;
```

The mandatory parameter $input is used to determine the source of the compressed data. This parameter can take one of three forms.

A filename

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

A filehandle

> If the $input parameter is a filehandle, the compressed 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 compressed data will be read from `$$output`.

Constructor Options

The option names defined below are case insensitive and can be optionally prefixed by a '-'. So all of the following are valid

```
-AutoClose
-autoclose
AUTOCLOSE
autoclose
```

OPTS is a combination of the following options:

`Name => "membername"`

Open "membername" from the zip file for reading.

`AutoClose => 0|1`

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

This parameter defaults to 0.

`MultiStream => 0|1`

Treats the complete zip file/buffer as a single compressed data stream. When reading in multi-stream mode each member of the zip file/buffer will be uncompressed in turn until the end of the file/buffer is encountered.

This parameter defaults to 0.

`Prime => $string`

This option will uncompress the contents of `$string` before processing the input file/buffer.

This option can be useful when the compressed data is embedded in another file/data structure and it is not possible to work out where the compressed data begins without having to read the first few bytes. If this is the case, the uncompression can be *primed* with these bytes using this option.

`Transparent => 0|1`

If this option is set and the input file/buffer is not compressed data, the module will allow reading of it anyway.

In addition, if the input file/buffer does contain compressed data and there is non-compressed data immediately following it, setting this option will make this module treat the whole file/buffer as a single data stream.

This option defaults to 1.

`BlockSize => $num`

When reading the compressed input data, IO::Uncompress::Unzip will read it in blocks of `$num` bytes.

This option defaults to 4096.

`InputLength => $size`

When present this option will limit the number of compressed bytes read from the input file/buffer to `$size`. This option can be used in the situation where there is useful data directly after the compressed data stream and you know beforehand the exact length of the compressed data stream.

This option is mostly used when reading from a filehandle, in which case the file pointer will be left pointing to the first byte directly after the compressed data stream.

This option defaults to off.

`Append => 0|1`

This option controls what the `read` method does with uncompressed data.

If set to 1, all uncompressed data will be appended to the output parameter of the `read` method.

If set to 0, the contents of the output parameter of the `read` method will be overwritten by the uncompressed data.

Defaults to 0.

`Strict => 0|1`

This option controls whether the extra checks defined below are used when carrying out the decompression. When Strict is on, the extra tests are carried out, when Strict is off they are not.

The default for this option is off.

Examples

TODO

Methods

read

Usage is

```
$status = $z->read($buffer)
```

Reads a block of compressed data (the size the the compressed block is determined by the `Buffer` option in the constructor), uncompresses it and writes any uncompressed data into `$buffer`. If the `Append` parameter is set in the constructor, the uncompressed data will be appended to the `$buffer` parameter. Otherwise `$buffer` will be overwritten.

Returns the number of uncompressed bytes written to `$buffer`, zero if eof or a negative number on error.

read

Usage is

```
$status = $z->read($buffer, $length)
$status = $z->read($buffer, $length, $offset)

$status = read($z, $buffer, $length)
$status = read($z, $buffer, $length, $offset)
```

Attempt to read `$length` bytes of uncompressed data into `$buffer`.

The main difference between this form of the `read` method and the previous one, is that this one will attempt to return *exactly* `$length` bytes. The only circumstances that this function will not is if end-of-file or an IO error is encountered.

Returns the number of uncompressed bytes written to `$buffer`, zero if eof or a negative number on error.

getline

Usage is

```
$line = $z->getline()
$line = <$z>
```

Reads a single line.

This method fully supports the use of of the variable `$/` (or `$INPUT_RECORD_SEPARATOR` or `$RS` when `English` is in use) to determine what constitutes an end of line. Paragraph mode, record mode and file slurp mode are all supported.

getc

Usage is

```
$char = $z->getc()
```

Read a single character.

ungetc

Usage is

```
$char = $z->ungetc($string)
```

inflateSync

Usage is

```
$status = $z->inflateSync()
```

TODO

getHeaderInfo

Usage is

```
$hdr  = $z->getHeaderInfo();
@hdrs = $z->getHeaderInfo();
```

This method returns either a hash reference (in scalar context) or a list or hash references (in array context) that contains information about each of the header fields in the compressed data stream(s).

tell

Usage is

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

Returns the uncompressed file offset.

eof

Usage is

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

Returns true if the end of the compressed input stream has been reached.

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 input file/buffer. It is a fatal error to attempt to seek backward.

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)
```

Returns the current uncompressed line number. If EXPR is present it has the effect of setting the line number. Note that setting the line number does not change the current position within the file/buffer being read.

The contents of `$/` are used to to determine what constitutes a line terminator.

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 is associated with a buffer, this method will return `undef`.

close

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

Closes the output file/buffer.

For most versions of Perl this method will be automatically invoked if the IO::Uncompress::Unzip 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::Uncompress::Unzip object was created, and the object is associated with a file, the underlying file will also be closed.

nextStream

Usage is

```
my $status = $z->nextStream();
```

Skips to the next compressed data stream in the input file/buffer. If a new compressed data stream is found, the eof marker will be cleared and `$.` will be reset to 0.

Returns 1 if a new stream was found, 0 if none was found, and -1 if an error was encountered.

trailingData

Usage is

```
my $data = $z->trailingData();
```

Returns the data, if any, that is present immediately after the compressed data stream once uncompression is complete. It only makes sense to call this method once the end of the compressed data stream has been encountered.

This option can be used when there is useful information immediately following the compressed data stream, and you don't know the length of the compressed data stream.

If the input is a buffer, `trailingData` will return everything from the end of the compressed data stream to the end of the buffer.

If the input is a filehandle, `trailingData` will return the data that is left in the filehandle input buffer once the end of the compressed data stream has been reached. You can then use the filehandle to read the rest of the input file.

Don't bother using `trailingData` if the input is a filename.

If you know the length of the compressed data stream before you start uncompressing, you can avoid having to use `trailingData` by setting the `InputLength` option in the constructor.

Importing

No symbolic constants are required by this IO::Uncompress::Unzip at present.

:all

Imports `unzip` and `$UnzipError`. Same as doing this

```
use IO::Uncompress::Unzip qw(unzip $UnzipError) ;
```

EXAMPLES

Working with Net::FTP

See *IO::Uncompress::Unzip::FAQ*

Walking through a zip file

The code below can be used to traverse a zip file, one compressed data stream at a time.

```
use IO::Uncompress::Unzip qw($UnzipError);

my $zipfile = "somefile.zip";
my $u = new IO::Uncompress::Unzip $zipfile
    or die "Cannot open $zipfile: $UnzipError";

my $status;
for ($status = 1; ! $u->eof(); $status = $u->nextStream())
{

    my $name = $u->getHeaderInfo()->{Name};
    warn "Processing member $name\n" ;

    my $buff;
    while (($status = $u->read($buff)) > 0) {
        # Do something here
    }

    last if $status < 0;
}

die "Error processing $zipfile: $!\n"
    if $status < 0 ;
```

Each individual compressed data stream is read until the logical end-of-file is reached. Then `nextStream` is called. This will skip to the start of the next compressed data stream and clear the end-of-file flag.

It is also worth noting that `nextStream` can be called at any time -- you don't have to wait until you have exhausted a compressed data stream before skipping to the next one.

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::Lzma*, *IO::Uncompress::UnLzma*, *IO::Compress::Xz*, *IO::Uncompress::UnXz*, *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* and Mark Adler *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*.

MODIFICATION HISTORY

See the Changes file.

COPYRIGHT AND LICENSE