Package 'zlib'

January 20, 2025

Version 1.0.3

Type Package

Title Compression and Decompression

Author Semjon Geist [aut, cre]

License MIT + file LICENSE

URL https://github.com/sgeist-ionos/R-zlib

BugReports https://github.com/sgeist-ionos/R-zlib/issues

Description The 'zlib' package for R aims to offer an R-based equivalent of 'Python's' built-in 'zlib' module for data compression and decompression. This package provides a suite of functions for working with 'zlib' compression, including utilities for compressing and decompressing data streams, manipulating compressed files, and working with 'gzip', 'zlib', and 'deflate' formats.

Depends R (>= 3.6.0)

Imports Rcpp

Suggests testthat (>= 3.0.0)

LinkingTo Rcpp

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

Config/testthat/edition 3

NeedsCompilation yes

Maintainer Semjon Geist <mail@semjon-geist.de>

Repository CRAN

Date/Publication 2023-10-18 20:50:02 UTC

Contents

`compress` *Single-step compression of raw data*

Description

Compresses the provided raw data in a single step.

Usage

```
compress(
  data,
  level = -1,
  method = zlib$DEFLATED,
  wbits = zlib$MAX_WBITS,
  memLevel = zlib$DEF_MEM_LEVEL,
  strategy = zlib$Z_DEFAULT_STRATEGY,
  zdict = NULL
)
```

Arguments

`data`	Raw data to be compressed.
`level`	Compression level, default is -1.
`method`	Compression method, default is `zlib$DEFLATED`.
`wbits`	Window bits, default is `zlib$MAX_WBITS`.
`memLevel`	Memory level, default is `zlib$DEF_MEM_LEVEL`.
`strategy`	Compression strategy, default is `zlib$Z_DEFAULT_STRATEGY`.
`zdict`	Optional predefined compression dictionary as a raw vector.

Details

The compress function simplifies the compression process by encapsulating the creation of a compression object, compressing the data, and flushing the buffer all within a single call. This is particularly useful for scenarios where the user wants to quickly compress data without dealing with the intricacies of compression objects and buffer management. The function leverages the compressobj function to handle the underlying compression mechanics.

Value

A raw vector containing the compressed data.

Examples

```
compressed_data <- compress(charToRaw("some data"))
```

---

compressobj *Create a Compression Object*

Description

compressobj initializes a new compression object with specified parameters and methods. The function makes use of publicEval to manage scope and encapsulation.

Usage

```
compressobj(
             level = -1,
             method = zlib$DEFLATED,
             wbits = zlib$MAX_WBITS,
             memLevel = zlib$DEF_MEM_LEVEL,
             strategy = zlib$Z_DEFAULT_STRATEGY,
             zdict = NULL
         )
```

Arguments

level	Compression level, default is -1.
method	Compression method, default is zlib$DEFLATED.
wbits	Window bits, default is zlib$MAX_WBITS.
memLevel	Memory level, default is zlib$DEF_MEM_LEVEL.
strategy	Compression strategy, default is zlib$Z_DEFAULT_STRATEGY.
zdict	Optional predefined compression dictionary as a raw vector.

Value

Returns an environment containing the public methods compress and flush.

Methods

- `compress(data)`: Compresses a chunk of data.
- `flush()`: Flushes the compression buffer.

Examples

```
compressor <- compressobj(level = 6)
compressed_data <- compressor$compress(charToRaw("some data"))
compressed_data <- c(compressed_data, compressor$flush())
```

---

compress_chunk *Compress a Chunk of Data*

Description

Compresses a given chunk of raw binary data using a pre-existing compressor object.

Usage

```
compress_chunk(compressorPtr, input_chunk)
```

Arguments

`compressorPtr`	An external pointer to an existing compressor object. This object is usually initialized by calling a different function like `create_compressor()`.
`input_chunk`	A raw vector containing the uncompressed data that needs to be compressed.

Details

This function is primarily designed for use with a compressor object created by `create_compressor()`. It takes a chunk of raw data and compresses it, returning a raw vector of the compressed data.

Value

A raw vector containing the compressed data.

Examples

```
# Create a new compressor object for zlib -> wbts = 15
zlib_compressor <- create_compressor(wbits=31)
compressed_data <- compress_chunk(zlib_compressor, charToRaw("Hello, World"))
compressed_data <- c(compressed_data, flush_compressor_buffer(zlib_compressor))
decompressed_data <- memDecompress(compressed_data, type = "gzip")
cat(rawToChar(decompressed_data))
```

create_compressor *Create a new compressor object*

Description

Initialize a new compressor object for zlib-based compression with specified settings.

Usage

```
create_compressor(
  level = -1L,
  method = 8L,
  wbits = 15L,
  memLevel = 8L,
  strategy = 0L,
  zdict = NULL
)
```

Arguments

level	Compression level, integer between 0 and 9, or -1 for default.
method	Compression method.
wbits	Window size bits.
memLevel	Memory level for internal compression state.
strategy	Compression strategy.
zdict	Optional predefined compression dictionary as a raw vector.

Value

A SEXP pointer to the new compressor object.

Examples

```
compressor <- create_compressor(level = 6, memLevel = 8)
```

create_decompressor *Create a new decompressor object*

Description

Initialize a new decompressor object for zlib-based decompression.

Usage

```
create_decompressor(wbits = 0L)
```

Arguments

wbits	The window size bits parameter. Default is 0.

Value

A SEXP pointer to the new decompressor object.

Examples

```
decompressor <- create_decompressor()
```

---

decompress *Single-step decompression of raw data*

Description

Decompresses the provided compressed raw data in a single step.

Usage

```
decompress(data, wbits = 0)
```

Arguments

data	Compressed raw data to be decompressed.
wbits	The window size bits parameter. Default is 0.

Details

The `decompress` function offers a streamlined approach to decompressing raw data. By abstracting the creation of a decompression object, decompressing the data, and flushing the buffer into one function call, it provides a hassle-free way to retrieve original data from its compressed form. This function is designed to work seamlessly with data compressed using the `compress` function or any other zlib-based compression method.

Value

A raw vector containing the decompressed data.

Examples

```
original_data <- charToRaw("some data")
compressed_data <- compress(original_data)
decompressed_data <- decompress(compressed_data)
```

`decompressobj` *Create a new decompressor object*

Description

Initializes a new decompressor object for zlib-based decompression.

Usage

```
decompressobj(wbits = 0)
```

Arguments

`wbits`	The window size bits parameter. Default is 0.

Details

The returned decompressor object has methods for performing chunk-wise decompression on compressed data using the zlib library.

Value

A decompressor object with methods for decompression.

Methods

- `decompress(data)`: Compresses a chunk of data.
- `flush()`: Flushes the compression buffer.

Examples

```
compressor <- zlib$compressobj(zlib$Z_DEFAULT_COMPRESSION, zlib$DEFLATED, zlib$MAX_WBITS + 16)
compressed_data <- compressor$compress(charToRaw("some data"))
compressed_data <- c(compressed_data, compressor$flush())
decompressor <- decompressobj(zlib$MAX_WBITS + 16)
decompressed_data <- c(decompressor$decompress(compressed_data), decompressor$flush())
```

decompress_chunk *Decompress a chunk of data*

Description

Perform chunk-wise decompression on a given raw vector using a decompressor object.

Usage

```
decompress_chunk(decompressorPtr, input_chunk)
```

Arguments

decompressorPtr	An external pointer to an initialized decompressor object.
input_chunk	A raw vector containing the compressed data chunk.

Value

A raw vector containing the decompressed data.

Examples

```
rawToChar(decompress_chunk(create_decompressor(), memCompress(charToRaw("Hello, World"))))
```

flush_compressor_buffer *Flush the internal buffer of the compressor object.*

Description

This function flushes the internal buffer according to the specified mode.

Usage

```
flush_compressor_buffer(compressorPtr, mode = 4L)
```

Arguments

compressorPtr	A SEXP pointer to an existing compressor object.
mode	A compression flush mode. Default is Z_FINISH. Available modes are Z_NO_FLUSH, Z_PARTIAL_FLUSH, Z_SYNC_FLUSH, Z_FULL_FLUSH, Z_BLOCK, and Z_FINISH.

Value

A raw vector containing the flushed output.

Examples

```
compressor <- create_compressor()
# ... (some compression actions)
flushed_data <- flush_compressor_buffer(compressor)
```

`flush_decompressor_buffer` *Flush the internal buffer of the decompressor object.*

Description

This function processes all pending input and returns the remaining uncompressed output. The function uses the provided initial buffer size and dynamically expands it as necessary to ensure all remaining data is decompressed. After calling this function, the decompress_chunk() method cannot be called again on the same object.

Usage

```
flush_decompressor_buffer(decompressorPtr, length = 256L)
```

Arguments

`decompressorPtr`	A SEXP pointer to an existing decompressor object.
`length`	An optional parameter that sets the initial size of the output buffer. Default is 256.

Value

A raw vector containing the remaining uncompressed output.

Examples

```
decompressor <- create_decompressor()
# ... (some decompression actions)
flushed_data <- flush_decompressor_buffer(decompressor)
```

validate_gzip_file *Validate if a File is a Valid Gzip File*

Description

This function takes a file path as input and checks if it's a valid gzip-compressed file. It reads the file in chunks and tries to decompress it using the zlib library. If any step fails, the function returns FALSE. Otherwise, it returns TRUE.

Usage

```
validate_gzip_file(file_path)
```

Arguments

file_path	A string representing the path of the file to validate.

Value

A boolean value indicating whether the file is a valid gzip file. TRUE if the file is valid, FALSE otherwise.

Examples

```
validate_gzip_file("path/to/your/file.gz")
```

Index