Package ‘zlib’

September 2, 2023

Type Package

Title R zlib Package

Version 0.1.0

Author Semjon Geist

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

Description The RZlibEquivalent package 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 (>= 1.0.9)

Remotes Rcpp

License file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Suggests testthat (>= 3.0.0), remotes (>= 2.4.2), roxygen2md (>= 1.0.0), rmarkdown (>= 2.21)

Config/testthat/edition 3

LinkingTo Rcpp

Roxygen list(markdown = TRUE)

NeedsCompilation yes

R topics documented:

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

```
## Not run:
comp_obj <- compressobj(level = 6)
compressed_data <- comp_obj$compress("some data")
flushed_data <- comp_obj$flush()

## End(Not run)
```

`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

```
## Not run:
compressor <- create_compressor()
compressed_data <- compress_chunk(compressor, charToRaw("Hello, World"))
compressed_data <- c(compressed_data, flush_compressor_buffer(compressor))
decompressed_data <- memDecompress(compressed_data, type = "gzip")
cat(rawToChar(decompressed_data))

## End(Not run)
```

`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()
```

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.

Examples

```
# Create a decompressor with default window size
decompressor <- decompressobj()

# Create a decompressor with a specific window size
decompressor <- decompressobj(wbits = zlib$MAX_WBITS)
```

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. 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)
```

`publicEval` *Evaluate Expression with Public and Private Environments*

Description

`publicEval` creates an environment hierarchy consisting of public, self, and private environments. The expression `expr` is evaluated within these nested environments, allowing for controlled variable scope and encapsulation.

Usage

```
publicEval(expr, parentEnv = parent.frame(), name = NULL)
```

Arguments

`expr`	An expression to evaluate within the constructed environment hierarchy.
`parentEnv`	The parent environment for the new 'public' environment. Default is the parent frame.
`name`	Optional name attribute to set for the public environment.

Value

Returns an invisible reference to the public environment.

Environments

- Public: Variables in this environment are externally accessible.
- Self: Inherits from Public and also contains Private and Public as children.
- Private: Variables are encapsulated and are not externally accessible.

Examples

```
## Not run:
publicEnv <- publicEval({
  private$hidden_var <- "I am hidden"
  public_var <- "I am public"
}, parentEnv = parent.frame(), name = "MyEnvironment")

print(exists("public_var", envir = publicEnv))  # Should return TRUE
print(exists("hidden_var", envir = publicEnv))  # Should return FALSE

## End(Not run)
```

`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

```
## Not run:
validate_gzip_file("path/to/your/file.gz")

## End(Not run)
```

zlib *.onLoad function for the package*

Description

This function is automatically called when the package is loaded using `library()` or `require()`. It initializes the package environment, including defining a variety of constants related to the zlib compression library.

Usage

```
.onLoad(libname, pkgname)
```

Details

Specifically, the function assigns a new environment named "zlib" containing constants such as `DEFLATED`, `DEF_BUF_SIZE`, `MAX_WBITS`, and various flush and compression strategies like `Z_FINISH`, `Z_BEST_COMPRESSION`, etc.

See Also

`publicEval()` for the method used to set up the public environment.

`zlib_constants()` for the method used to set up the constants in the environment.

zlib_constants *Retrieve zlib Constants*

Description

This function returns a list of constants from the zlib C library.

Usage

```
constants <- zlib_constants()
```

Details

The constants are defined as follows:

- `DEFLATED`: The compression method, set to 8.
- `DEF_BUF_SIZE`: The default buffer size, set to 16384.
- `DEF_MEM_LEVEL`: Default memory level, set to 8.
- `MAX_WBITS`: Maximum size of the history buffer, set to 15.
- `Z_BEST_COMPRESSION`: Best compression level, set to 9.
- `Z_BEST_SPEED`: Best speed for compression, set to 1.
- `Z_BLOCK`: Block compression mode, set to 5.
- `Z_DEFAULT_COMPRESSION`: Default compression level, set to -1.

- `Z_DEFAULT_STRATEGY`: Default compression strategy, set to 0.
- `Z_FILTERED`: Filtered compression mode, set to 1.
- `Z_FINISH`: Finish compression mode, set to 4.
- `Z_FULL_FLUSH`: Full flush mode, set to 3.
- `Z_HUFFMAN_ONLY`: Huffman-only compression mode, set to 2.
- `Z_NO_COMPRESSION`: No compression, set to 0.
- `Z_NO_FLUSH`: No flush mode, set to 0.
- `Z_PARTIAL_FLUSH`: Partial flush mode, set to 1.
- `Z_RLE`: Run-length encoding compression mode, set to 3.
- `Z_SYNC_FLUSH`: Synchronized flush mode, set to 2.
- `Z_TREES`: Tree block compression mode, set to 6.

Value

A named list of zlib constants.

Index