

Package ‘qs2’

November 21, 2025

Type Package

Title Efficient Serialization of R Objects

Version 0.1.6

Date 2025-11-20

Maintainer Travers Ching <traversc@gmail.com>

Description Streamlines and accelerates the process of saving and loading R objects, improving speed and compression compared to other methods. The package provides two compression formats: the 'qs2' format, which uses R serialization via the C API while optimizing compression and disk I/O, and the 'qdata' format, featuring custom serialization for slightly faster performance and better compression. Additionally, the 'qs2' format can be directly converted to the standard 'RDS' format, ensuring long-term compatibility with future versions of R.

License GPL-3

LazyData true

Biarch true

Depends R (>= 3.5.0)

Imports Rcpp, stringfish (>= 0.15.1)

LinkingTo Rcpp, stringfish, RcppParallel

Suggests knitr, rmarkdown, dplyr, data.table, stringi

SystemRequirements GNU make

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

Copyright This package includes code from the 'zstd' library owned by Facebook, Inc. and created by Yann Collet; and code derived from the 'Blosc' library created and owned by Francesc Alted.

URL <https://github.com/qsbase/qs2>

BugReports <https://github.com/qsbase/qs2/issues>

NeedsCompilation yes

Author Travers Ching [aut, cre, cph],
 Yann Collet [ctb, cph] (Yann Collet is the author of the bundled zstd),
 Facebook, Inc. [cph] (Facebook is the copyright holder of the bundled
 zstd code),
 Reichardt Tino [ctb, cph] (Contributor/copyright holder of zstd bundled
 code),
 Skibinski Przemyslaw [ctb, cph] (Contributor/copyright holder of zstd
 bundled code),
 Mori Yuta [ctb, cph] (Contributor/copyright holder of zstd bundled
 code),
 Francisc Altred [ctb, cph] (Shuffling routines derived from Blosc
 library)

Repository CRAN

Date/Publication 2025-11-21 06:40:02 UTC

Contents

base85_decode	3
base85_encode	3
base91_decode	4
base91_encode	4
blosc_shuffle_raw	5
blosc_unshuffle_raw	6
catquo	6
decode_source	7
encode_source	7
qd_deserialize	8
qd_read	9
qd_save	10
qd_serialize	11
qopt	12
qs_deserialize	13
qs_read	14
qs_readm	15
qs_save	16
qs_savem	17
qs_serialize	17
qs_to_rds	18
qx_dump	19
rds_to_qs	20
starnames	21
xxhash_raw	21
zstd_compress_bound	22
zstd_compress_file	22
zstd_compress_raw	23
zstd_decompress_file	24
zstd_decompress_raw	24

base85_decode	<i>Z85 Decoding</i>
---------------	---------------------

Description

Decodes a Z85 encoded string back to binary

Usage

```
base85_decode(encoded_string)
```

Arguments

`encoded_string` A string.

Value

The original raw vector.

base85_encode	<i>Z85 Encoding</i>
---------------	---------------------

Description

Encodes binary data (a raw vector) as ASCII text using **Z85 encoding format**.

Usage

```
base85_encode(rawdata)
```

Arguments

`rawdata` A raw vector.

Details

Z85 is a binary to ASCII encoding format created by Pieter Hintjens in 2010 and is part of the ZeroMQ RFC. The encoding has a dictionary using 85 out of 94 printable ASCII characters. There are other base 85 encoding schemes, including Ascii85, which is popularized and used by Adobe. Z85 is distinguished by its choice of dictionary, which is suitable for easier inclusion into source code for many programming languages. The dictionary excludes all quote marks and other control characters, and requires no special treatment in R and most other languages. Note: although the official specification restricts input length to multiples of four bytes, the implementation here works with any input length. The overhead (extra bytes used relative to binary) is 25%. In comparison, base 64 encoding has an overhead of 33.33%.

Value

A string representation of the raw vector.

References

<https://rfc.zeromq.org/spec/32/>

base91_decode	<i>baseE91 Decoding</i>
---------------	-------------------------

Description

Decodes a baseE91 encoded string back to binary

Usage

```
base91_decode(encoded_string)
```

Arguments

encoded_string A string.

Value

The original raw vector.

base91_encode	<i>baseE91 Encoding</i>
---------------	-------------------------

Description

Encodes binary data (a raw vector) as ASCII text using **baseE91 encoding format**.

Usage

```
base91_encode(rawdata, quote_character = "\"")
```

Arguments

rawdata A raw vector.

quote_character

The character to use in the encoding, replacing the double quote character. Must be either a single quote ("'"), a double quote ("\"") or a dash ("-").

Details

base91 (capital E for stylization) is a binary to ASCII encoding format created by Joachim Henke in 2005. The overhead (extra bytes used relative to binary) is 22.97% on average. In comparison, base 64 encoding has an overhead of 33.33%. The original encoding uses a dictionary of 91 out of 94 printable ASCII characters excluding - (dash), \ (backslash) and ' (single quote). The original encoding does include double quote characters, which are less than ideal for strings in R. Therefore, you can use the `quote_character` parameter to substitute dash or single quote.

Value

A string representation of the raw vector.

References

<https://base91.sourceforge.net/>

blosc_shuffle_raw	<i>Shuffle a raw vector</i>
-------------------	-----------------------------

Description

Shuffles a raw vector using BLOSC shuffle routines.

Usage

```
blosc_shuffle_raw(data, bytesofsize)
```

Arguments

<code>data</code>	A raw vector to be shuffled.
<code>bytesofsize</code>	Either 4 or 8.

Value

The shuffled vector.

Examples

```
x <- serialize(1L:1000L, NULL)
xshuf <- blosc_shuffle_raw(x, 4)
xunshuf <- blosc_unshuffle_raw(xshuf, 4)
```

blosc_unshuffle_raw	<i>Un-shuffle a raw vector</i>
---------------------	--------------------------------

Description

Un-shuffles a raw vector using BLOSC un-shuffle routines.

Usage

```
blosc_unshuffle_raw(data, bytesofsize)
```

Arguments

data	A raw vector to be unshuffled.
bytesofsize	Either 4 or 8.

Value

The unshuffled vector.

Examples

```
x <- serialize(1L:1000L, NULL)
xshuf <- blosc_shuffle_raw(x, 4)
xunshuf <- blosc_unshuffle_raw(xshuf, 4)
```

catquo	<i>catquo</i>
--------	---------------

Description

Prints a string with single quotes on a new line.

Usage

```
catquo(...)
```

Arguments

...	Arguments passed on to <code>cat()</code> .
-----	---

decode_source	<i>Decode a compressed string</i>
---------------	-----------------------------------

Description

A helper function for encoding and compressing a file or string to ASCII using [base91_encode\(\)](#) and [qs_serialize\(\)](#) with the highest compression level.

Usage

```
decode_source(string)
```

Arguments

string	A string to decode.
--------	---------------------

Value

The original (decoded) object.

See Also

[encode_source\(\)](#) for more details.

encode_source	<i>Encode and compress a file or string</i>
---------------	---

Description

A helper function for encoding and compressing a file or string to ASCII using [base91_encode\(\)](#) and [qs_serialize\(\)](#) with the highest compression level.

Usage

```
encode_source(x = NULL, file = NULL, width = 120)
```

Arguments

x	The object to encode (if file is not NULL)
file	The file to encode (if x is not NULL)
width	The output will be broken up into individual strings, with width being the longest allowable string.

Details

The `encode_source()` and `decode_source()` functions are useful for storing small amounts of data or text inline to a .R or .Rmd file.

Value

A character vector in base91 representing the compressed original file or object.

Examples

```
set.seed(1); data <- sample(500)
result <- encode_source(data)
# Note: the result string is not guaranteed to be consistent between qs or zstd versions
#       but will always properly decode regardless
print(result)
result <- decode_source(result) # [1] 1 2 3 4 5 6 7 8 9 10
```

qd_deserialize

qd_deserialize

Description

Deserializes a raw vector to an object using the qdata format.

Usage

```
qd_deserialize(input,
               use_alt_rep = qopt("use_alt_rep"),
               validate_checksum = qopt("validate_checksum"),
               nthreads = qopt("nthreads"))
```

Arguments

<code>input</code>	The raw vector to deserialize.
<code>use_alt_rep</code>	Use ALTREP when reading in string data (the initial value is FALSE).
<code>validate_checksum</code>	Whether to validate the stored checksum in the file (the initial value is FALSE).
<code>nthreads</code>	The number of threads to use when reading data (the initial value is 1L). When TBB is not available, values greater than 1 emit a warning and fall back to 1.

Value

The deserialized object.

Examples

```
x <- data.frame(int = sample(1e3, replace=TRUE),
               num = rnorm(1e3),
               char = sample(state.name, 1e3, replace=TRUE),
               stringsAsFactors = FALSE)
xserialized <- qd_serialize(x)
x2 <- qd_deserialize(xserialized)
identical(x, x2) # returns TRUE
```

qd_read

*qd_read***Description**

Reads an object that was saved to disk in the qdata format.

Usage

```
qd_read(file,
        use_alt_rep = qopt("use_alt_rep"),
        validate_checksum = qopt("validate_checksum"),
        nthreads = qopt("nthreads"))
```

Arguments

<code>file</code>	The file name/path.
<code>use_alt_rep</code>	Use ALTREP when reading in string data (the initial value is FALSE).
<code>validate_checksum</code>	Whether to validate the stored checksum in the file (the initial value is FALSE).
<code>nthreads</code>	The number of threads to use when reading data (the initial value is 1L). When TBB is not available, values greater than 1 emit a warning and fall back to 1.

Value

The object stored in file.

Examples

```
x <- data.frame(int = sample(1e3, replace=TRUE),
               num = rnorm(1e3),
               char = sample(state.name, 1e3, replace=TRUE),
               stringsAsFactors = FALSE)
myfile <- tempfile()
qd_save(x, myfile)
x2 <- qd_read(myfile)
identical(x, x2) # returns TRUE
```

qd_save	<i>qd_save</i>
---------	----------------

Description

Saves an object to disk using the qdata format.

Usage

```
qd_save(object, file,
        compress_level = qopt("compress_level"),
        shuffle = qopt("shuffle"),
        warn_unsupported_types = qopt("warn_unsupported_types"),
        nthreads = qopt("nthreads"))
```

Arguments

object	The object to save.
file	The file name/path.
compress_level	The compression level used (the initial value is 3L). The maximum and minimum possible values depend on the version of the ZSTD library used. As of ZSTD 1.5.6 the maximum compression level is 22, and the minimum is -131072. Usually, values in the low positive range offer very good performance in terms of speed and compression.
shuffle	Whether to allow byte shuffling when compressing data (the initial value is TRUE).
warn_unsupported_types	Whether to warn when saving an object with an unsupported type (the initial value is TRUE).
nthreads	The number of threads to use when compressing data (the initial value is 1L). When TBB is not available, values greater than 1 emit a warning and fall back to 1.

Value

No value is returned. The file is written to disk.

Examples

```
x <- data.frame(int = sample(1e3, replace=TRUE),
               num = rnorm(1e3),
               char = sample(state.name, 1e3, replace=TRUE),
               stringsAsFactors = FALSE)
myfile <- tempfile()
qd_save(x, myfile)
x2 <- qd_read(myfile)
identical(x, x2) # returns TRUE
```

qd_serialize	<i>qd_serialize</i>
--------------	---------------------

Description

Serializes an object to a raw vector using the qdata format.

Usage

```
qd_serialize(object,
             compress_level = qopt("compress_level"),
             shuffle = qopt("shuffle"),
             warn_unsupported_types = qopt("warn_unsupported_types"),
             nthreads = qopt("nthreads"))
```

Arguments

object	The object to save.
compress_level	The compression level used (the initial value is 3L). The maximum and minimum possible values depend on the version of the ZSTD library used. As of ZSTD 1.5.6 the maximum compression level is 22, and the minimum is -131072. Usually, values in the low positive range offer very good performance in terms of speed and compression.
shuffle	Whether to allow byte shuffling when compressing data (the initial value is TRUE).
warn_unsupported_types	Whether to warn when saving an object with an unsupported type (the initial value is TRUE).
nthreads	The number of threads to use when compressing data (the initial value is 1L). When TBB is not available, values greater than 1 emit a warning and fall back to 1.

Value

The serialized object as a raw vector.

Examples

```
x <- data.frame(int = sample(1e3, replace=TRUE),
               num = rnorm(1e3),
               char = sample(state.name, 1e3, replace=TRUE),
               stringsAsFactors = FALSE)
xserialized <- qd_serialize(x)
x2 <- qd_deserialize(xserialized)
identical(x, x2) # returns TRUE
```

qopt	<i>qs2 Option Getter/Setter</i>
------	---------------------------------

Description

Get or set a global qs2 option.

Usage

```
qopt(parameter, value = NULL)
```

Arguments

parameter	A character string specifying the option to access. Must be one of "compress_level", "shuffle", "nthreads", "validate_checksum", "warn_unsupported_types", or "use_alt_rep".
value	If NULL (the default), the current value is retrieved. Otherwise, the global option is set to value.

Details

This function provides an interface to retrieve or update internal qs2 options such as compression level, shuffle flag, number of threads, checksum validation, warning for unsupported types, and ALTREP usage. It directly calls the underlying C-level functions.

The default settings are:

- compress_level: 3L
- shuffle: TRUE
- nthreads: 1L
- validate_checksum: FALSE
- warn_unsupported_types: TRUE (used only in qd_save)
- use_alt_rep: FALSE (used only in qd_read)

When value is NULL, the current value of the specified option is returned. Otherwise, the option is set to value and the new value is returned invisibly.

Value

If value is NULL, returns the current value of the specified option. Otherwise, sets the option and returns the new value invisibly.

Examples

```
# Get the current compression level:
qopt("compress_level")

# Set the compression level to 5:
qopt("compress_level", value = 5)

# Get the current shuffle setting:
qopt("shuffle")

# Get the current setting for warn_unsupported_types (used in qd_save):
qopt("warn_unsupported_types")

# Get the current setting for use_alt_rep (used in qd_read):
qopt("use_alt_rep")
```

qs_deserialize

qs_deserialize

Description

Deserializes a raw vector to an object using the qs2 format.

Usage

```
qs_deserialize(input,
  validate_checksum = qopt("validate_checksum"),
  nthreads = qopt("nthreads"))
```

Arguments

input	The raw vector to deserialize.
validate_checksum	Whether to validate the stored checksum in the file (the initial value is FALSE).
nthreads	The number of threads to use when reading data (the initial value is 1L). When TBB is not available, values greater than 1 emit a warning and fall back to 1.

Value

The deserialized object.

Examples

```
x <- data.frame(int = sample(1e3, replace=TRUE),
  num = rnorm(1e3),
  char = sample(state.name, 1e3, replace=TRUE),
  stringsAsFactors = FALSE)
xserialized <- qs_serialize(x)
x2 <- qs_deserialize(xserialized)
identical(x, x2) # returns TRUE
```

qs_read

qs_read

Description

Reads an object that was saved to disk in the qs2 format.

Usage

```
qs_read(file,
  validate_checksum = qopt("validate_checksum"),
  nthreads = qopt("nthreads"))
```

Arguments

file	The file name/path.
validate_checksum	Whether to validate the stored checksum in the file (the initial value is FALSE).
nthreads	The number of threads to use when reading data (the initial value is 1L). When TBB is not available, values greater than 1 emit a warning and fall back to 1.

Value

The object stored in file.

Examples

```
x <- data.frame(int = sample(1e3, replace=TRUE),
  num = rnorm(1e3),
  char = sample(state.name, 1e3, replace=TRUE),
  stringsAsFactors = FALSE)
myfile <- tempfile()
qs_save(x, myfile)
x2 <- qs_read(myfile)
identical(x, x2) # returns TRUE
```

qs_readm

*qs_readm***Description**

Reads an object in a file serialized to disk using [qs_savem\(\)](#).

Usage

```
qs_readm(file, env = parent.frame(), ...)
```

Arguments

file	The file name/path.
env	The environment where the data should be loaded. Default is the calling environment (parent.frame()).
...	additional arguments will be passed to qs_read .

Details

This function extends [qs_read](#) to replicate the functionality of [base::load\(\)](#) to load multiple saved objects into your workspace.

Value

Nothing is explicitly returned, but the function will load the saved objects into the workspace.

Examples

```
x1 <- data.frame(int = sample(1e3, replace=TRUE),
                 num = rnorm(1e3),
                 char = sample(starnames$`IAU Name`, 1e3, replace=TRUE),
                 stringsAsFactors = FALSE)
x2 <- data.frame(int = sample(1e3, replace=TRUE),
                 num = rnorm(1e3),
                 char = sample(starnames$`IAU Name`, 1e3, replace=TRUE),
                 stringsAsFactors = FALSE)
myfile <- tempfile()
qs_savem(x1, x2, file=myfile)
rm(x1, x2)
qs_readm(myfile)
exists('x1') && exists('x2') # returns true
```

 qs_save

qs_save

Description

Saves an object to disk using the qs2 format.

Usage

```
qs_save(object, file,
        compress_level = qopt("compress_level"),
        shuffle = qopt("shuffle"),
        nthreads = qopt("nthreads"))
```

Arguments

object	The object to save.
file	The file name/path.
compress_level	The compression level used (the initial value is 3L). The maximum and minimum possible values depend on the version of the ZSTD library used. As of ZSTD 1.5.6 the maximum compression level is 22, and the minimum is -131072. Usually, values in the low positive range offer very good performance in terms of speed and compression.
shuffle	Whether to allow byte shuffling when compressing data (the initial value is TRUE).
nthreads	The number of threads to use when compressing data (the initial value is 1L). When TBB is not available, values greater than 1 emit a warning and fall back to 1.

Value

No value is returned. The file is written to disk.

Examples

```
x <- data.frame(int = sample(1e3, replace=TRUE),
               num = rnorm(1e3),
               char = sample(state.name, 1e3, replace=TRUE),
               stringsAsFactors = FALSE)
myfile <- tempfile()
qs_save(x, myfile)
x2 <- qs_read(myfile)
identical(x, x2) # returns TRUE
```

 qs_savem

qs_savem

Description

Saves (serializes) multiple objects to disk.

Usage

```
qs_savem(...)
```

Arguments

... Objects to serialize. Named arguments will be passed to [qs_save\(\)](#) during saving. Un-named arguments will be saved. A named file argument is required.

Details

This function extends [qs_save\(\)](#) to replicate the functionality of [base::save\(\)](#) to save multiple objects. Read them back with [qs_readm\(\)](#).

Examples

```
x1 <- data.frame(int = sample(1e3, replace=TRUE),
                 num = rnorm(1e3),
                 char = sample(starnames$`IAU Name`, 1e3, replace=TRUE),
                 stringsAsFactors = FALSE)
x2 <- data.frame(int = sample(1e3, replace=TRUE),
                 num = rnorm(1e3),
                 char = sample(starnames$`IAU Name`, 1e3, replace=TRUE),
                 stringsAsFactors = FALSE)
myfile <- tempfile()
qs_savem(x1, x2, file=myfile)
rm(x1, x2)
qs_readm(myfile)
exists('x1') && exists('x2') # returns true
```

 qs_serialize

qs_serialize

Description

Serializes an object to a raw vector using the qs2 format.

Usage

```
qs_serialize(object,
             compress_level = qopt("compress_level"),
             shuffle = qopt("shuffle"),
             nthreads = qopt("nthreads"))
```

Arguments

<code>object</code>	The object to save.
<code>compress_level</code>	The compression level used (the initial value is 3L). The maximum and minimum possible values depend on the version of the ZSTD library used. As of ZSTD 1.5.6 the maximum compression level is 22, and the minimum is -131072. Usually, values in the low positive range offer very good performance in terms of speed and compression.
<code>shuffle</code>	Whether to allow byte shuffling when compressing data (the initial value is TRUE).
<code>nthreads</code>	The number of threads to use when compressing data (the initial value is 1L). When TBB is not available, values greater than 1 emit a warning and fall back to 1.

Value

The serialized object as a raw vector.

Examples

```
x <- data.frame(int = sample(1e3, replace=TRUE),
               num = rnorm(1e3),
               char = sample(state.name, 1e3, replace=TRUE),
               stringsAsFactors = FALSE)
xserialized <- qs_serialize(x)
x2 <- qs_deserialize(xserialized)
identical(x, x2) # returns TRUE
```

`qs_to_rds`
qs2 to RDS format

Description

Converts a file saved in the qs2 format to the RDS format.

Usage

```
qs_to_rds(input_file, output_file, compress_level = 6)
```

Arguments

`input_file` The qs2 file to convert.

`output_file` The RDS file to write.

`compress_level` The gzip compression level to use when writing the RDS file (a value between 0 and 9).

Value

No value is returned. The converted file is written to disk.

Examples

```
qs_tmp <- tempfile(fileext = ".qs2")
rds_tmp <- tempfile(fileext = ".RDS")

x <- runif(1e6)
qs_save(x, qs_tmp)
qs_to_rds(input_file = qs_tmp, output_file = rds_tmp)
x2 <- readRDS(rds_tmp)
stopifnot(identical(x, x2))
```

qx_dump

qx_dump

Description

Exports the uncompressed binary serialization to a list of raw vectors for both qs2 and qdata formats. For testing and exploratory purposes mainly.

Usage

```
qx_dump(file)
```

Arguments

`file` A file name/path.

Value

A list containing uncompressed binary serialization and metadata.

Examples

```
x <- data.frame(int = sample(1e3, replace=TRUE),
               num = rnorm(1e3),
               char = sample(state.name, 1e3, replace=TRUE),
               stringsAsFactors = FALSE)
myfile <- tempfile()
qs_save(x, myfile)
binary_data <- qx_dump(myfile)
```

rds_to_qs

RDS to qs2 format

Description

Converts a file saved in the RDS format to the qs2 format.

Usage

```
rds_to_qs(input_file, output_file, compress_level = 3)
```

Arguments

input_file	The RDS file to convert.
output_file	The qs2 file to write.
compress_level	The zstd compression level to use when writing the qs2 file. See the <code>qs_save</code> help file for more details on this parameter.

Details

The shuffle parameters is currently not supported when converting from RDS to qs2. When reading the resulting qs2 file, `validate_checksum` must be set to `FALSE`.

Value

No value is returned. The converted file is written to disk.

Examples

```
qs_tmp <- tempfile(fileext = ".qs2")
rds_tmp <- tempfile(fileext = ".RDS")

x <- runif(1e6)
saveRDS(x, rds_tmp)
rds_to_qs(input_file = rds_tmp, output_file = qs_tmp)
x2 <- qs_read(qs_tmp, validate_checksum = FALSE)
stopifnot(identical(x, x2))
```

starnames	<i>Official list of IAU Star Names</i>
-----------	--

Description

Data from the International Astronomical Union. An official list of the 336 internationally recognized named stars, updated as of June 1, 2018.

Usage

```
data(starnames)
```

Format

A `data.frame` with official IAU star names and several properties, such as coordinates.

Source

[Naming Stars | International Astronomical Union.](#)

References

E Mamajek et. al. (2018), *WG Triennial Report (2015-2018) - Star Names*, Reports on Astronomy, 22 Mar 2018.

Examples

```
data(starnames)
```

xxhash_raw	<i>XXH3_64 hash</i>
------------	---------------------

Description

Calculates a 64-bit XXH3 hash.

Usage

```
xxhash_raw(data)
```

Arguments

data	The data to hash.
------	-------------------

Value

The 64-bit hash.

Examples

```
x <- as.raw(c(1,2,3))
xxhash_raw(x)
```

zstd_compress_bound	<i>Zstd compress bound</i>
---------------------	----------------------------

Description

Exports the compress bound function from the zstd library. Returns the maximum potential compressed size of an object of length size.

Usage

```
zstd_compress_bound(size)
```

Arguments

size	An integer size
------	-----------------

Value

Maximum compressed size.

Examples

```
zstd_compress_bound(100000)
zstd_compress_bound(1e9)
```

zstd_compress_file	<i>Zstd file compression</i>
--------------------	------------------------------

Description

A utility function to compresses a file with zstd.

Usage

```
zstd_compress_file(input_file, output_file, compress_level = qopt("compress_level"))
```

Arguments

input_file	Path to the file to compress.
output_file	Path for the compressed file.
compress_level	The compression level used.

Value

No value is returned. The file is written to disk.

Examples

```
infile <- tempfile()
writeBin(as.raw(1:5), infile)
outfile <- tempfile()
zstd_compress_file(infile, outfile, compress_level = 1)
stopifnot(file.exists(outfile))
```

zstd_compress_raw	<i>Zstd compression</i>
-------------------	-------------------------

Description

Compresses to a raw vector using the zstd algorithm. Exports the main zstd compression function.

Usage

```
zstd_compress_raw(data, compress_level = qopt("compress_level"))
```

Arguments

`data` Raw vector to be compressed.

`compress_level` The compression level used.

Value

The compressed data as a raw vector.

Examples

```
x <- 1:1e6
xserialized <- serialize(x, connection=NULL)
xcompressed <- zstd_compress_raw(xserialized, compress_level = 1)
xrecovered <- unserialize(zstd_decompress_raw(xcompressed))
```

zstd_decompress_file	<i>Zstd file decompression</i>
----------------------	--------------------------------

Description

A utility function to decompresses a zstd file to disk.

Usage

```
zstd_decompress_file(input_file, output_file)
```

Arguments

input_file	Path to the zstd file.
output_file	Path for the decompressed file.

Value

No value is returned. The file is written to disk.

Examples

```
infile <- tempfile()
writeBin(as.raw(1:5), infile)
zfile <- tempfile()
zstd_compress_file(infile, zfile, compress_level = 1)
outfile <- tempfile()
zstd_decompress_file(zfile, outfile)
stopifnot(identical(readBin(infile, what = "raw", n = 5), readBin(outfile, what = "raw", n = 5)))
```

zstd_decompress_raw	<i>Zstd decompression</i>
---------------------	---------------------------

Description

Decompresses a zstd compressed raw vector.

Usage

```
zstd_decompress_raw(data)
```

Arguments

data	A raw vector to be decompressed.
------	----------------------------------

Value

The decompressed data as a raw vector.

Examples

```
x <- 1:1e6
xserialized <- serialize(x, connection=NULL)
xcompressed <- zstd_compress_raw(xserialized, compress_level = 1)
xrecovered <- unserialize(zstd_decompress_raw(xcompressed))
```

Index

* **datasets**

starnames, [21](#)

base85_decode, [3](#)

base85_encode, [3](#)

base91_decode, [4](#)

base91_encode, [4](#)

base91_encode(), [7](#)

base::load(), [15](#)

base::save(), [17](#)

blosc_shuffle_raw, [5](#)

blosc_unshuffle_raw, [6](#)

cat(), [6](#)

catquo, [6](#)

decode_source, [7](#)

decode_source(), [8](#)

encode_source, [7](#)

encode_source(), [7](#), [8](#)

parent.frame(), [15](#)

qd_deserialize, [8](#)

qd_read, [9](#)

qd_save, [10](#)

qd_serialize, [11](#)

qopt, [12](#)

qs_deserialize, [13](#)

qs_read, [14](#), [15](#)

qs_readm, [15](#)

qs_readm(), [17](#)

qs_save, [16](#)

qs_save(), [17](#)

qs_savem, [17](#)

qs_savem(), [15](#)

qs_serialize, [17](#)

qs_serialize(), [7](#)

qs_to_rds, [18](#)

qx_dump, [19](#)

rds_to_qs, [20](#)

starnames, [21](#)

xxhash_raw, [21](#)

zstd_compress_bound, [22](#)

zstd_compress_file, [22](#)

zstd_compress_raw, [23](#)

zstd_decompress_file, [24](#)

zstd_decompress_raw, [24](#)