# Package 'qs2'

August 21, 2026

**Type** Package

**Title** Efficient Serialization of R Objects

**Version** 0.3.1

**Date** 2026-08-19

**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 (>= 3.6.0)

**Imports** Rcpp, RcppParallel (>= 6.1.1), stringfish (>= 0.18.0)

**LinkingTo** Rcpp, RcppParallel (>= 6.1.1)

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

**SystemRequirements** GNU make, C++17

**Encoding** UTF-8

**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

**Config/roxygen2/version** 8.0.0

**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),
Francesc Alted [ctb, cph] (Shuffling routines derived from Blosc library)

**Repository** CRAN

**Date/Publication** 2026-08-21 14:00:07 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
generate_test_data . . . 8
qd_deserialize . . . 9
qd_read . . . 10
qd_save . . . 11
qd_serialize . . . 12
qopt . . . 13
qs_cache . . . 14
qs_deserialize . . . 15
qs_read . . . 16
qs_readm . . . 17
qs_save . . . 18
qs_savem . . . 19
qs_serialize . . . 20
qs_to_rds . . . 21
qx_dump . . . 21
rds_to_qs . . . 22
starnames . . . 23
xxhash_raw . . . 24
zstd_compress_bound . . . 24
zstd_compress_raw . . . 25
zstd_decompress_raw . . . 25

zstd_file_functions . . . . . 26
zstd_in . . . . . 27

**Index** **29**

---

`base85_decode` *Z85 Decoding*

---

## 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` *Z85 Encoding*

---

## 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` *basE91 Decoding*

---

## Description

Decodes a basE91 encoded string back to binary

## Usage

```
base91_decode(encoded_string)
```

## Arguments

| | |
|---|---|
| `encoded_string` | A string. |

## Value

The original raw vector.

---

`base91_encode` *basE91 Encoding*

---

## Description

Encodes binary data (a raw vector) as ASCII text using basE91 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` *Shuffle a raw vector*

---

### Description

Shuffles a raw vector using BLOSC shuffle routines.

### Usage

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

### Arguments

| | |
|---|---|
| `data` | A raw vector to be shuffled. |
| `bytesofsize` | 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 *Un-shuffle a raw vector*

### 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 *catquo*

### Description

Prints a string with single quotes on a new line.

### Usage

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

### Arguments

| | |
|---|---|
| `...` | Arguments passed on to `cat()`. |

## decode_source *Decode a compressed string*

### 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 *Encode and compress a file or string*

### 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
```

---

# generate_test_data *Generate deterministic mixed-column test data*

---

## Description

Creates a small deterministic data frame with string, numeric, and integer columns. The numeric and integer columns combine seeded random noise with a mild linear trend so downstream smoke tests exercise both repetition and variation.

## Usage

```
generate_test_data(nrows, seed = 1L)
```

## Arguments

| | |
|---|---|
| nrows | Number of rows to generate. |
| seed | Integer seed used to make the output reproducible. |

## Value

A data frame with columns string, numeric, and int.

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

| | |
|---|---|
| input | The raw vector to deserialize. |
| use_alt_rep | Request ALTREP when reading qdata string data. This option is temporarily disabled; if TRUE, qs2 warns and falls back to ordinary character vectors (the initial value is FALSE). |
| validate_checksum | If TRUE, validate checksum before deserialization and error on mismatch (or missing checksum). If FALSE, checksum is computed during read and mismatches (or missing checksum) produce a warning after reading (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 <- 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

| | |
|---|---|
| file | The file name/path. |
| use_alt_rep | Request ALTREP when reading qdata string data. This option is temporarily disabled; if TRUE, qs2 warns and falls back to ordinary character vectors (the initial value is FALSE). |
| validate_checksum | If TRUE, validate checksum before deserialization and error on mismatch (or missing checksum). If FALSE, checksum is computed during read and mismatches (or missing checksum) produce a warning after reading (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()
qd_save(x, myfile)
x2 <- qd_read(myfile)
identical(x, x2) # returns TRUE
```

qd_save *qd_save*

## 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 1.5.7 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 *qd_serialize*

## 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).<br>The maximum and minimum possible values depend on the version of the ZSTD library used. As of ZSTD 1.5.7 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 *qs2 Option Getter/Setter*

## 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 requested 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 (accepted by qd_read and qd_deserialize, but temporarily disabled)

When parameter = "use_alt_rep" is set to TRUE, qdata reads currently warn and fall back to ordinary character vectors.

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:
qopt("use_alt_rep")
```

---

qs_cache *qcache*

---

## Description

Helper function for caching objects for long running tasks

## Usage

```
qs_cache(
  expr,
  name,
  envir = parent.frame(),
  cache_dir = ".cache",
  clear = FALSE,
  prompt = TRUE,
  qs_save_params = list(),
  qs_read_params = list(),
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| expr | The expression to evaluate. |
| name | The cached expression name (see details). |
| envir | The environment to evaluate expr in. |
| cache_dir | The directory to store cached files in. |
| clear | Set to TRUE to clear the cache (see details). |
| prompt | Whether to prompt before clearing. |

| | |
|---|---|
| qs_save_params | List of parameters passed on to qs_save. |
| qs_read_params | List of parameters passed on to qs_read. |
| verbose | Boolean. If TRUE, the function prints an informative message regarding the status of cached objects. |

## Details

This is a (very) simple helper function to cache results of long running calculations. There are other packages specializing in caching data that are more feature complete.

The evaluated expression is saved with qs_save() in <cache_dir>/<name>.qs2. If the file already exists instead, the expression is not evaluated and the cached result is read using qs_read() and returned.

To clear a cached result, you can manually delete the associated .qs2 file, or you can call qs_cache() with clear = TRUE. If prompt is also TRUE a prompt will be given asking you to confirm deletion. If name is not specified, all cached results in cache_dir will be removed.

## Examples

```
cache_dir <- tempdir()

a <- 1
b <- 5

# not cached
result <- qs_cache({a + b},
                   name="aplusb",
                   cache_dir = cache_dir,
                   qs_save_params = list(compress_level = 5))

# cached
result <- qs_cache({a + b},
                   name="aplusb",
                   cache_dir = cache_dir,
                   qs_save_params = list(compress_level = 5))

# clear cached result
qs_cache(name="aplusb", clear=TRUE, prompt=FALSE, cache_dir = cache_dir)
```

---

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 | If TRUE, validate checksum before deserialization and error on mismatch (or missing checksum). If FALSE, checksum is computed during read and mismatches (or missing checksum) produce a warning after reading (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 | If TRUE, validate checksum before deserialization and error on mismatch (or missing checksum). If FALSE, checksum is computed during read and mismatches (or missing checksum) produce a warning after reading (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).<br>The maximum and minimum possible values depend on the version of the ZSTD library used. As of ZSTD 1.5.7 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

| | |
|---|---|
| object | The object to save. |
| compress_level | The compression level used (the initial value is 3L).<br>The maximum and minimum possible values depend on the version of the ZSTD library used. As of ZSTD 1.5.7 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

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 qs_save help file for more details on this parameter. |

### Details

rds_to_qs() currently supports only gzip-compressed RDS input files. Files that are uncompressed or use another compression format are rejected.

The shuffle parameter is currently not supported when converting from RDS to qs2.

### 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, compress = "gzip")
rds_to_qs(input_file = rds_tmp, output_file = qs_tmp)
x2 <- qs_read(qs_tmp, validate_checksum = TRUE)
stopifnot(identical(x, x2))
```

---

# starnames *Official list of IAU Star Names*

---

## 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 *XXH3_64 hash*

### 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 *Zstd compress bound*

### 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 | A single non-negative whole number. Values larger than `2^31 - 1` are allowed, up to `2^53 - 1`. Beyond that an R numeric no longer identifies a single integer, since `2^53` and `2^53 + 1` are the same double. |

### Value

A numeric scalar giving the maximum compressed size.

### Examples

```
zstd_compress_bound(100000)
zstd_compress_bound(2^31)
```

---

## zstd_compress_raw *Zstd compression*

---

### 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_raw *Zstd decompression*

---

### 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))
```

---

# zstd_file_functions *Zstd file helpers*

---

## Description

Helpers for compressing and decompressing zstd files.

A utility function to compresses a file with zstd.

A utility function to decompresses a zstd file to disk.

## Usage

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

zstd_decompress_file(input_file, output_file, max_output_bytes = NULL)
```

## Arguments

| | |
|---|---|
| `input_file` | Path to the input file. |
| `output_file` | Path to the output file. |
| `compress_level` | The compression level used. |
| `max_output_bytes` | Optional maximum number of decompressed output bytes. When supplied, decompression stops with an error before writing a chunk that would exceed this limit. |

## Value

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

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))
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_in *Zstd file substitution for input*

---

## Description

Substitutes a zstd compressed file for a regular input file. The zstd compressed file is decompressed to the input FUN.

Substitutes a zstd compressed file for a regular output file. The output of FUN is converted to a zstd compressed file at the target zstd file path.

## Usage

```
zstd_in(
  FUN,
  ...,
  envir = parent.frame(),
  tmpfile = tempfile(),
  max_output_bytes = NULL
)

zstd_out(FUN, ..., envir = parent.frame(), tmpfile = tempfile())
```

## Arguments

| | |
|---|---|
| FUN | Function to call. |
| ... | Arguments passed to FUN. The first named argument is treated as the file path. |
| envir | Environment for FUN evaluation. |
| tmpfile | Intermediate file path used during conversion. It is removed on exit, whether supplied or auto-generated. |
| max_output_bytes | Optional maximum number of decompressed output bytes passed through to zstd_decompress_file(). |

## Details

This is a generic wrapper that works with any function that reads from a file.

This is a generic wrapper that works with any function that writes to a file.

## Value

The value returned by FUN.

The value returned by FUN, with its visibility preserved.

## Examples

```
if (requireNamespace("data.table", quietly = TRUE)) {
  zfile <- tempfile(fileext = ".csv.zst")
  zstd_out(data.table::fwrite, mtcars, file = zfile)
  dt <- zstd_in(data.table::fread, file = zfile)
  print(nrow(dt))
}
```

# Index


∗ **datasets**
    starnames, 23

base85_decode, 3
base85_encode, 3
base91_decode, 4
base91_encode, 4
base91_encode(), *7*
base::load(), *17*
base::save(), *19*
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*

generate_test_data, 8

parent.frame(), *17*

qd_deserialize, 9
qd_read, 10
qd_save, 11
qd_serialize, 12
qopt, 13
qs_cache, 14
qs_cache(), *15*
qs_deserialize, 15
qs_read, 16, *17*
qs_read(), *15*
qs_readm, 17
qs_readm(), *19*
qs_save, 18
qs_save(), *15*, *19*
qs_savem, 19
qs_savem(), *17*
qs_serialize, 20
qs_serialize(), *7*
qs_to_rds, 21
qx_dump, 21

rds_to_qs, 22

starnames, 23

xxhash_raw, 24

zstd_compress_bound, 24
zstd_compress_file
        *(*zstd_file_functions*)*, 26
zstd_compress_raw, 25
zstd_decompress_file
        *(*zstd_file_functions*)*, 26
zstd_decompress_file(), *27*
zstd_decompress_raw, 25
zstd_file_functions, 26
zstd_in, 27
zstd_out *(*zstd_in*)*, 27