Package ‘qs’

October 13, 2022

Type Package
Title Quick Serialization of R Objects
Version 0.25.4
Date 2022-8-9
Maintainer Travers Ching <traversc@gmail.com>
Description Provides functions for quickly writing and reading any R object to and from disk.
License GPL-3
LazyData true
Biarch true
Depends R (>= 3.0.2)
SystemRequirements C++11
Imports Rcpp, RApiSerialize (>= 0.1.1), stringfish (>= 0.15.1)
LinkingTo Rcpp, RApiSerialize, stringfish
Encoding UTF-8
RoxygenNote 7.1.2
Suggests knitr, rmarkdown, testthat, dplyr, data.table
VignetteBuilder knitr
Copyright This package includes code from the 'zstd' library owned by
Facebook, Inc. and created by Yann Collet; the 'lz4' library
created and owned by Yann Collet; xxHash library created and
owned by Yann Collet; and code derived from the 'Blosc' library
created and owned by Francesc Alted.
URL https://github.com/traversc/qs
BugReports https://github.com/traversc/qs/issues
NeedsCompilation yes
Author Travers Ching [aut, cre, cph],
Yann Collet [ctb, cph] (Yann Collet is the author of the bundled zstd,
lz4 and xxHash code),
Facebook, Inc. [cph] (Facebook is the copyright holder of the bundled
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),
Romain Francois [ctb, cph] (Derived example/tutorials for ALTREP structures),
Francesc Alted [ctb, cph] (Shuffling routines derived from Blosc library),
Bryce Chamberlain [ctb] (qsavem and qload functions),
Salim Brüggemann [ctb] (<https://orcid.org/0000-0002-5329-5987>, documentation)

**Repository** CRAN

**Date/Publication** 2022-08-09 23:20:03 UTC

**R topics documented:**

- 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
- is_big_endian .......................................... 8
- lz4_compress_bound .................................... 9
- lz4_compress_raw ...................................... 9
- lz4_decompress_raw .................................. 10
- qdeserialize .......................................... 12
- qdump .................................................... 13
- qread ..................................................... 13
- qreadm ................................................... 14
- qread_fd ................................................ 16
- qread_handle .......................................... 16
- qread_ptr .............................................. 17
- qsave .................................................... 17
- qsavem .................................................. 19
- qsave_fd ............................................... 20
- qsave_handle .......................................... 22
- qserialize ............................................. 23
- starnames .............................................. 25
- zstd_compress_bound .................................. 26
**base85_decode**

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%.
**base91_encode**

**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
http://base91.sourceforge.net/

blosc_shuffle_raw

Shuffle a raw vector

Description
Shuffles a raw vector using BLOSC shuffle routines.

Usage
blosc_shuffle_raw(x, bytesofsize)

Arguments
x A raw vector.
bytesofsize Either 4 or 8.

Value
The shuffled vector

Examples
x <- serialize(1:1000L, 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**  
```r
blosc_unshuffle_raw(x, bytesofsize)
```

**Arguments**  
- `x`  
  A raw vector.
- `bytesofsize`  
  Either 4 or 8.

**Value**  
The unshuffled vector.

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

**catquo**

**Description**  
Prints a string with single quotes on a new line.

**Usage**  
```r
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 `qserialize()` with the highest compression level.

**Usage**

```r
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 `qserialize()` with the highest compression level.

**Usage**

```r
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

```r
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
```

Description

Tests system endianness. Intel and AMD based systems are little endian, and so this function will likely return `FALSE`. The `qs` package is not capable of transferring data between systems of different endianness. This should not matter for the large majority of use cases.

Usage

`is_big_endian()`

Value

`TRUE` if big endian, `FALSE` if little endian.

Examples

`is_big_endian() # returns FALSE on Intel/AMD systems`
**Description**

Exports the compress bound function from the lz4 library. Returns the maximum compressed size of an object of length `size`.

**Usage**

```r
lz4_compress_bound(size)
```

**Arguments**

- `size` An integer size.

**Value**

Maximum compressed size.

**Examples**

```r
lz4_compress_bound(100000)
#` lz4_compress_bound(1e9)
```

---

**Description**

Compresses to a raw vector using the lz4 algorithm. Exports the main lz4 compression function.

**Usage**

```r
lz4_compress_raw(x, compress_level)
```

**Arguments**

- `x` The object to serialize.
- `compress_level` The compression level used. A number > 1 (higher is less compressed).

**Value**

The compressed data as a raw vector.
Examples

```r
x <- 1:1e6
xserialized <- serialize(x, connection=NULL)
xcompressed <- lz4_compress_raw(xserialized, compress_level = 1)
xrecovered <- unserialize(lz4_decompress_raw(xcompressed))
```

**Description**

Decompresses an lz4 compressed raw vector.

**Usage**

```r
lz4_decompress_raw(x)
```

**Arguments**

- `x` A raw vector.

**Value**

The de-serialized object.

**Examples**

```r
x <- 1:1e6
xserialized <- serialize(x, connection=NULL)
xcompressed <- lz4_compress_raw(xserialized, compress_level = 1)
xrecovered <- unserialize(lz4_decompress_raw(xcompressed))
```

**Description**

Helper function for caching objects for long running tasks
Usage

qcache(
  expr,
  name,
  envir = parent.frame(),
  cache_dir = ".cache",
  clear = FALSE,
  prompt = TRUE,
  qsave_params = list(),
  qread_params = list()
)

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.
- `qsave_params` Parameters passed on to `qsave`.
- `qread_params` Parameters passed on to `qread`.

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 `qsave` in `<cache_dir>/<name>.qs`. If the file already exists instead, the expression is not evaluated and the cached result is read using `qread()` and returned.

To clear a cached result, you can manually delete the associated .qs file, or you can call `qcache()` 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

```r
cache_dir <- tempdir()

a <- 1
b <- 5

# not cached
result <- qcache(a + b),
  name="aplusb",
  cache_dir = cache_dir,
  qsave_params = list(preset="fast"))
```
# cached
result <- qcache(a + b, name="aplusb", cache_dir = cache_dir, qsave_params = list(preset="fast"))

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

---

qdeserializew

### Description

Reads an object from a raw vector.

### Usage

```r
qdeserializew(x, use_alt_rep=FALSE, strict=FALSE)
```

### Arguments

- **x**: A raw vector.
- **use_alt_rep**: Use ALTREP when reading in string data (default FALSE). On R versions prior to 3.5.0, this parameter does nothing.
- **strict**: Whether to throw an error or just report a warning (default: FALSE, i.e. report warning).

### Details

See `qserialize()` for additional details and examples.

### Value

The de-serialized object.
**Description**

Exports the uncompressed binary serialization to a list of raw vectors. For testing purposes and exploratory purposes mainly.

**Usage**

`qdump(file)`

**Arguments**

- **file**
  - A file name/path.

**Value**

The uncompressed serialization.

**Examples**

```r
x <- data.frame(int = sample(1e3, replace=TRUE),
                num = rnorm(1e3),
                char = sample(starnames$'IAU Name', 1e3, replace=TRUE),
                stringsAsFactors = FALSE)
myfile <- tempfile()
qsave(x, myfile)
x2 <- qdump(myfile)
```

**Description**

Reads an object in a file serialized to disk.

**Usage**

`qread(file, use_alt_rep=FALSE, strict=FALSE, nthreads=1)`
Arguments

- **file**: The file name/path.
- **use_alt_rep**: Use ALTREP when reading in string data (default FALSE). On R versions prior to 3.5.0, this parameter does nothing.
- **strict**: Whether to throw an error or just report a warning (default: FALSE, i.e. report warning).
- **nthreads**: Number of threads to use. Default 1.

Value

The de-serialized object.

Examples

```r
x <- data.frame(int = sample(1e3, replace=TRUE),
                num = rnorm(1e3),
                char = sample(starnames$`IAU Name`, 1e3, replace=TRUE),
                stringsAsFactors = FALSE)
myfile <- tempfile()
qsave(x, myfile)
x2 <- qread(myfile)
identical(x, x2) # returns true

# qrs support multithreading
qsave(x, myfile, nthreads=2)
x2 <- qread(myfile, nthreads=2)
identical(x, x2) # returns true

# Other examples
z <- 1:1e7
myfile <- tempfile()
qsave(z, myfile)
z2 <- qread(myfile)
identical(z, z2) # returns true

w <- as.list(rnorm(1e6))
myfile <- tempfile()
qsave(w, myfile)
w2 <- qread(myfile)
identical(w, w2) # returns true
```

Description

Reads an object in a file serialized to disk using `qsavem()`.
qreadm

Usage

qreadm(file, env = parent.frame(), ...)  
qload(file, env = parent.frame(), ...)

Arguments

- **file**: The file name/path.
- **env**: The environment where the data should be loaded.
- **...**: additional arguments will be passed to qread.

Details

This function extends qread to replicate the functionality of `base::load()` to load multiple saved objects into your workspace. `qload` and `qreadm` are alias of the same function.

Value

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

Examples

```r
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()
qsavem(x1, x2, file=myfile)
rm(x1, x2)
qload(myfile)
exists('x1') && exists('x2') # returns true

# qs support multithreading
qsavem(x1, x2, file=myfile, nthreads=2)
rllx1, x2)
qload(myfile, nthreads=2)
exists('x1') && exists('x2') # returns true
```
qread_fd

Description
Reads an object from a file descriptor.

Usage
qread_fd(fd, use_alt_rep=FALSE, strict=FALSE)

Arguments
- `fd`: A file descriptor.
- `use_alt_rep`: Use ALTREP when reading in string data (default FALSE). On R versions prior to 3.5.0, this parameter does nothing.
- `strict`: Whether to throw an error or just report a warning (default: FALSE, i.e. report warning).

Details
See `qsave_fd()` for additional details and examples.

Value
The de-serialized object.

qread_handle

Description
Reads an object from a windows handle.

Usage
qread_handle(handle, use_alt_rep=FALSE, strict=FALSE)

Arguments
- `handle`: A windows handle external pointer.
- `use_alt_rep`: Use ALTREP when reading in string data (default FALSE). On R versions prior to 3.5.0, this parameter does nothing.
- `strict`: Whether to throw an error or just report a warning (default: FALSE, i.e. report warning).
Details

See `qsave_handle()` for additional details and examples.

Value

The de-serialized object.

qread_ptr

Description

Reads an object from an external pointer.

Usage

`qread_ptr(pointer, length, use_alt_rep=FALSE, strict=FALSE)`

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointer</td>
<td>An external pointer to memory.</td>
</tr>
<tr>
<td>length</td>
<td>The length of the object in memory.</td>
</tr>
<tr>
<td>use_alt_rep</td>
<td>Use ALTREP when reading in string data (default FALSE). On R versions prior to 3.5.0, this parameter does nothing.</td>
</tr>
<tr>
<td>strict</td>
<td>Whether to throw an error or just report a warning (default: FALSE, i.e. report warning).</td>
</tr>
</tbody>
</table>

Value

The de-serialized object.

qsave

Description

Saves (serializes) an object to disk.

Usage

`qsave(x, file, preset = "high", algorithm = "zstd", compress_level = 4L, shuffle_control = 15L, check_hash=TRUE, nthreads = 1)`
Arguments

x
file
preset
algorithm
compress_level
shuffle_control
check_hash
nthreads

Details

This function serializes and compresses R objects using block compression with the option of byte shuffling.

Value

The total number of bytes written to the file (returned invisibly).

Presets

There are lots of possible parameters. To simplify usage, there are four main presets that are performant over a large variety of data:

- "fast" is a shortcut for algorithm = "lz4", compress_level = 100 and shuffle_control = 0.
- "balanced" is a shortcut for algorithm = "lz4", compress_level = 1 and shuffle_control = 15.
- "high" is a shortcut for algorithm = "zstd", compress_level = 4 and shuffle_control = 15.
- "archive" is a shortcut for algorithm = "zstd_stream", compress_level = 14 and shuffle_control = 15. (zstd_stream is currently single-threaded only)

To gain more control over compression level and byte shuffling, set preset = "custom", in which case the individual parameters algorithm, compress_level and shuffle_control are actually regarded.
Byte shuffling

The parameter `shuffle_control` defines which numerical R object types are subject to **byte shuffling**. Generally speaking, the more ordered/sequential an object is (e.g., `1:1e7`), the larger the potential benefit of byte shuffling. It is not uncommon to improve compression ratio or compression speed by several orders of magnitude. The more random an object is (e.g., `rnorm(1e7)`), the less potential benefit there is, even negative benefit is possible. Integer vectors almost always benefit from byte shuffling, whereas the results for numeric vectors are mixed. To control block shuffling, add +1 to the parameter for logical vectors, +2 for integer vectors, +4 for numeric vectors and/or +8 for complex vectors.

Examples

```r
x <- data.frame(int = sample(1e3, replace=TRUE),
                num = rnorm(1e3),
                char = sample(starnames$'IAU Name', 1e3, replace=TRUE),
                stringsAsFactors = FALSE)
myfile <- tempfile()
qsave(x, myfile)
x2 <- qread(myfile)
identical(x, x2) # returns true

# qs support multithreading
qsave(x, myfile, nthreads=2)
x2 <- qread(myfile, nthreads=2)
identical(x, x2) # returns true

# Other examples
z <- 1:1e7
myfile <- tempfile()
qsave(z, myfile)
z2 <- qread(myfile)
idential(z, z2) # returns true

w <- as.list(rnorm(1e6))
myfile <- tempfile()
qsave(w, myfile)
w2 <- qread(myfile)
identical(w, w2) # returns true
```

Description

Saves (serializes) multiple objects to disk.

Usage

`qsavem(...)`
Arguments

Objects to serialize. Named arguments will be passed to \texttt{qsave()} during saving. Un-named arguments will be saved. A named file argument is required.

Details

This function extends \texttt{qsave()} to replicate the functionality of \texttt{base::save()} to save multiple objects. Read them back with \texttt{qload()}.

Examples

\begin{verbatim}
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()
qsavem(x1, x2, file=myfile)
rm(x1, x2)
qload(myfile)
exists('x1') && exists('x2') # returns true

# qs support multithreading
qsavem(x1, x2, file=myfile, nthreads=2)
rm(x1, x2)
qload(myfile, nthreads=2)
exists('x1') && exists('x2') # returns true
\end{verbatim}

Description

Saves an object to a file descriptor.

Usage

\begin{verbatim}
qsave_fd(x, fd,
preset = "high", algorithm = "zstd", compress_level = 4L,
shuffle_control = 15L, check_hash=TRUE)
\end{verbatim}
Arguments

- **x**: The object to serialize.
- **fd**: A file descriptor.
- **preset**: One of "fast", "balanced", "high" (default), "archive", "uncompressed" or "custom". See section *Presets* for details.
- **algorithm**: *Ignored unless* preset = "custom". Compression algorithm used: "lz4", "zstd", "lz4hc", "zstd_stream" or "uncompressed".
- **compress_level**: *Ignored unless* preset = "custom". The compression level used. For lz4, this number must be > 1 (higher is less compressed). For zstd, a number between -50 to 22 (higher is more compressed). Due to the format of qs, there is very little benefit to compression levels > 5 or so.
- **shuffle_control**: *Ignored unless* preset = "custom". An integer setting the use of byte shuffle compression. A value between 0 and 15 (default 15). See section *Byte shuffling* for details.
- **check_hash**: Default TRUE, compute a hash which can be used to verify file integrity during serialization.

Details

This function serializes and compresses R objects using block compression with the option of byte shuffling.

Value

The total number of bytes written to the file (returned invisibly).

Presets

There are lots of possible parameters. To simplify usage, there are four main presets that are performant over a large variety of data:

- "fast" is a shortcut for algorithm = "lz4", compress_level = 100 and shuffle_control = 0.
- "balanced" is a shortcut for algorithm = "lz4", compress_level = 1 and shuffle_control = 15.
- "high" is a shortcut for algorithm = "zstd", compress_level = 4 and shuffle_control = 15.
- "archive" is a shortcut for algorithm = "zstd_stream", compress_level = 14 and shuffle_control = 15. (zstd_stream is currently single-threaded only)

To gain more control over compression level and byte shuffling, set preset = "custom", in which case the individual parameters algorithm, compress_level and shuffle_control are actually regarded.
Byte shuffling

The parameter shuffle_control defines which numerical R object types are subject to byte shuffling. Generally speaking, the more ordered/sequential an object is (e.g., 1:1e7), the larger the potential benefit of byte shuffling. It is not uncommon to improve compression ratio or compression speed by several orders of magnitude. The more random an object is (e.g., rnorm(1e7)), the less potential benefit there is, even negative benefit is possible. Integer vectors almost always benefit from byte shuffling, whereas the results for numeric vectors are mixed. To control block shuffling, add +1 to the parameter for logical vectors, +2 for integer vectors, +4 for numeric vectors and/or +8 for complex vectors.

qsave_handle

Description

Saves an object to a windows handle.

Usage

qsave_handle(x, handle, preset = "high", algorithm = "zstd", compress_level = 4L, shuffle_control = 15L, check_hash=TRUE)

Arguments

x The object to serialize.
handle A windows handle external pointer.
preset One of "fast", "balanced", "high" (default), "archive", "uncompressed" or "custom". See section Presets for details.
algorithm Ignored unless preset = "custom". Compression algorithm used: "lz4", "zstd", "lz4hc", "zstd_stream" or "uncompressed".
compress_level Ignored unless preset = "custom". The compression level used.
  For lz4, this number must be > 1 (higher is less compressed).
  For zstd, a number between -50 to 22 (higher is more compressed). Due to the format of qs, there is very little benefit to compression levels > 5 or so.
shuffle_control Ignored unless preset = "custom". An integer setting the use of byte shuffle compression. A value between 0 and 15 (default 15). See section Byte shuffling for details.
check_hash Default TRUE, compute a hash which can be used to verify file integrity during serialization.

Details

This function serializes and compresses R objects using block compression with the option of byte shuffling.
qserialize

Value

The total number of bytes written to the file (returned invisibly).

Presets

There are lots of possible parameters. To simplify usage, there are four main presets that are performant over a large variety of data:

- "fast" is a shortcut for `algorithm = "lz4", compress_level = 100 and shuffle_control = 0`.
- "balanced" is a shortcut for `algorithm = "lz4", compress_level = 1 and shuffle_control = 15`.
- "high" is a shortcut for `algorithm = "zstd", compress_level = 4 and shuffle_control = 15`.
- "archive" is a shortcut for `algorithm = "zstd_stream", compress_level = 14 and shuffle_control = 15`. (`zstd_stream` is currently single-threaded only)

To gain more control over compression level and byte shuffling, set `preset = "custom"`, in which case the individual parameters `algorithm`, `compress_level` and `shuffle_control` are actually regarded.

Byte shuffling

The parameter `shuffle_control` defines which numerical R object types are subject to byte shuffling. Generally speaking, the more ordered/sequential an object is (e.g., `1:1e7`), the larger the potential benefit of byte shuffling. It is not uncommon to improve compression ratio or compression speed by several orders of magnitude. The more random an object is (e.g., `rnorm(1e7)`), the less potential benefit there is, even negative benefit is possible. Integer vectors almost always benefit from byte shuffling, whereas the results for numeric vectors are mixed. To control block shuffling, add +1 to the parameter for logical vectors, +2 for integer vectors, +4 for numeric vectors and/or +8 for complex vectors.

---

qserialize  qserialize

Description

Saves an object to a raw vector.

Usage

```r
qserialize(x, preset = "high",
algorithm = "zstd", compress_level = 4L,
shuffle_control = 15L, check_hash=TRUE)
```
Arguments

- **x**: The object to serialize.
- **preset**: One of "fast", "balanced", "high" (default), "archive", "uncompressed" or "custom". See section Presets for details.
- **algorithm**: `Ignored unless preset = "custom"`. Compression algorithm used: "lz4", "zstd", "lz4hc", "zstd_stream" or "uncompressed".
- **compress_level**: `Ignored unless preset = "custom"`. The compression level used.
  - For lz4, this number must be > 1 (higher is less compressed).
  - For zstd, a number between -50 to 22 (higher is more compressed). Due to the format of qs, there is very little benefit to compression levels > 5 or so.
- **shuffle_control**: `Ignored unless preset = "custom"`. An integer setting the use of byte shuffle compression. A value between 0 and 15 (default 15). See section Byte shuffling for details.
- **check_hash**: Default TRUE, compute a hash which can be used to verify file integrity during serialization.

Details

This function serializes and compresses R objects using block compression with the option of byte shuffling.

Value

A raw vector.

Presets

There are lots of possible parameters. To simplify usage, there are four main presets that are performant over a large variety of data:

- **"fast"** is a shortcut for `algorithm = "lz4", compress_level = 100` and `shuffle_control = 0`.
- **"balanced"** is a shortcut for `algorithm = "lz4", compress_level = 1` and `shuffle_control = 15`.
- **"high"** is a shortcut for `algorithm = "zstd", compress_level = 4` and `shuffle_control = 15`.
- **"archive"** is a shortcut for `algorithm = "zstd_stream", compress_level = 14` and `shuffle_control = 15` (zstd_stream is currently single-threaded only).

To gain more control over compression level and byte shuffling, set `preset = "custom"`, in which case the individual parameters `algorithm`, `compress_level` and `shuffle_control` are actually regarded.
Byte shuffling

The parameter `shuffle_control` defines which numerical R object types are subject to byte shuffling. Generally speaking, the more ordered/sequential an object is (e.g., `1:1e7`), the larger the potential benefit of byte shuffling. It is not uncommon to improve compression ratio or compression speed by several orders of magnitude. The more random an object is (e.g., `rnorm(1e7)`), the less potential benefit there is, even negative benefit is possible. Integer vectors almost always benefit from byte shuffling, whereas the results for numeric vectors are mixed. To control block shuffling, add +1 to the parameter for logical vectors, +2 for integer vectors, +4 for numeric vectors and/or +8 for complex vectors.

---

**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**

```r
data(starnames)
```

**Format**

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

**Source**

Naming Stars | International Astronomical Union.

**References**


**Examples**

```r
data(starnames)
```
### zstd_compress_bound

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

**Usage**
```r
zstd_compress_bound(size)
```

**Arguments**
- `size` An integer size

**Value**
maximum compressed size

**Examples**
```r
zstd_compress_bound(100000)
zstd_compress_bound(1e9)
```

### zstd_compress_raw

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

**Usage**
```r
zstd_compress_raw(x, compress_level)
```

**Arguments**
- `x` The object to serialize.
- `compress_level` The compression level used (default 4). A number between -50 to 22 (higher is more compressed). Due to the format of qs, there is very little benefit to compression levels > 5 or so.

**Value**
The compressed data as a raw vector.
Examples

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

---

**Description**

Decompresses a zstd compressed raw vector.

**Usage**

```r
zstd_decompress_raw(x)
```

**Arguments**

- **x**
  - A raw vector.

**Value**

The de-serialized object.

**Examples**

```r
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, 25

base85_decode, 3
base85_encode, 3
base91_decode, 4
base91_encode, 4
base91_encode(), 7
base::load(), 15
base::save(), 20
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

is_big_endian, 8

lz4_compress_bound, 9
lz4_compress_raw, 9
lz4_decompress_raw, 10

qcache, 10
qcache(), 11
qdeserialze, 12
qdump, 13
qload(qreadm), 14
qload(), 20
qread, 13
qread(), 11
qread_fd, 16
qread_handle, 16
qread_ptr, 17
qreadm, 14
qsave, 17

qsave(), 11, 20
qsave_fd, 20
qsave_fd(), 16
qsave_handle, 22
qsave_handle(), 17
qsavem, 19
qsavem(), 14
qserialize, 23
qserialize(), 7, 12

starnames, 25

zstd_compress_bound, 26
zstd_compress_raw, 26
zstd_decompress_raw, 27