Package: zstdlite (via r-universe)

October 22, 2024

Type Package

Title Fast Compression and Serialization with 'Zstandard' Algorithm

Version 0.2.6

Maintainer Mike Cheng <mikefc@coolbutuseless.com>

Description Fast, compressed serialization of R objects using the 'Zstandard' algorithm. R objects can be compressed and decompressed quickly using the standard serialization mechanism in R. Raw byte vectors and strings are also handled directly for compatibility with compressed data created by other systems and programs supporting 'Zstandard' compression. Dictionaries are supported for more effective compression of small data, and functions are provided for training these dictionaries. This implementation is a wrapper around the 'Zstandard' 'C' library which is available from <https://github.com/facebook/zstd>.

URL https://github.com/coolbutuseless/zstdlite

BugReports https://github.com/coolbutuseless/zstdlite/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.1

Copyright This package includes code from the 'zstd' library owned by Meta Platforms, Inc. and affiliates. and created by Yann Collet. See file 'inst/COPYRIGHTS' for details.

Suggests knitr, rmarkdown, testthat, bench

Depends R (>= 3.4.0)

VignetteBuilder knitr

Repository https://r-multiverse-staging.r-universe.dev

RemoteUrl https://github.com/coolbutuseless/zstdlite

RemoteRef 585458ccbe36eaa179d8b30f04f1e3a91dc6b993

RemoteSha 585458ccbe36eaa179d8b30f04f1e3a91dc6b993

Contents

`zstd_cctx` *Initialise a ZSTD compression context*

Description

Compression contexts can be re-used, meaning that they don't have to be created each time a compression function is called. This can make things faster when performing multiple compression operations.

Usage

```
zstd_cctx(level = 3L, num_threads = 1L, include_checksum = FALSE, dict = NULL)
```

Arguments

`level`	Compression level. Default: 3. Valid range is [-5, 22] with -5 representing the mode with least compression and 22 representing the mode with most compression. Note `level = 0` corresponds to the *default* level and is equivalent to `level = 3`
`num_threads`	Number of compression threads. Default 1. Using more threads can result in faster compression, but the magnitude of this speed-up depends on lots of factors e.g. cpu, drive speed, type of data compression level etc.
`include_checksum`	Include a checksum with the compressed data? Default: FALSE. If `TRUE` then a 32-bit hash of the original uncompressed data will be appended to the compressed data and checked for validity during decompression. See matching option for decompression in `zstd_dctx()` argument `validate_checksum`.
`dict`	Dictionary. Default: NULL. Can either be a raw vector or a filename. This dictionary can be created with `zstd_train_dict_compress()` , `zstd_train_dict_seriazlie()` or any other tool supporting `zstd` dictionary creation. Note: compressed data created with a dictionary *must* be decompressed with the same dictionary.

Value

External pointer to a ZSTD Compression Context which can be passed to `zstd_serialize()` and `zstd_compress()`

Examples

```
cctx <- zstd_cctx(level = 4)
```

`zstd_cctx_settings` *Get the configuration settings of a compression context*

Description

Get the configuration settings of a compression context

Usage

```
zstd_cctx_settings(cctx)
```

Arguments

`cctx`	ZSTD compression context, as created by `zstd_cctx()`

Value

named list of configuration options

Examples

```
cctx <- zstd_cctx()
zstd_cctx_settings(cctx)
```

`zstd_compress` *Compress/Decompress raw vectors and character strings.*

Description

This function is appropriate when handling data from other systems e.g. data compressed with the `zstd` command-line, or other compression programs.

Usage

```
zstd_compress(src, ..., file = NULL, cctx = NULL, use_file_streaming = FALSE)

zstd_decompress(
  src,
  type = "raw",
  ...,
  dctx = NULL,
  use_file_streaming = FALSE
)
```

Arguments

`src`	Source data to be compressed. This may be a raw vector, or a character string
`...`	extra arguments passed to `zstd_cctx()` or `zstd_dctx()` context initializers. Note: These argument are only used when `cctx` or `dctx` is NULL
`file`	filename in which to serialize data. If NULL (the default), then serialize the results to a raw vector
`cctx`	ZSTD Compression Context created by `zstd_cctx()` or NULL. Default: NULL will create a default compression context on-the-fly
`use_file_streaming`	Use the streaming interface when reading or writing to a file? This may reduce memory allocations and make better use of mutlithreading. Default: FALSE
`type`	Should data be returned as a 'raw' vector or as a 'string'? Default: 'raw'
`dctx`	ZSTD Decompression Context created by `zstd_dctx()` or NULL. Default: NULL will create a default decompression context on-the-fly.

Value

Raw vector of compressed data, or `NULL` if file created with compressed data

Examples

```
dat <- sample(as.raw(1:10), 1000, replace = TRUE)
vec <- zstd_compress(dat)
zstd_decompress(vec)

tmp <- tempfile()
zstd_compress(dat, file = tmp)
zstd_decompress(tmp)
```

`zstd_dctx` *Initialise a ZSTD decompression context*

Description

Decompression contexts can be re-used, meaning that they don't have to be created each time a decompression function is called. This can make things faster when performing multiple decompression operations.

Usage

```
zstd_dctx(validate_checksum = TRUE, dict = NULL)
```

Arguments

`validate_checksum`	If a checksum is present on the comrpessed data, should the checksum be validated? Default: TRUE. Set to `FALSE` to ignore the checksum, which may lead to a minor speed improvement. If no checksum is present in the compressed data, then this option has no effect.
`dict`	Dictionary. Default: NULL. Can either be a raw vector or a filename. This dictionary can be created with `zstd_train_dict_compress()` , `zstd_train_dict_seriazlie()` or any other tool supporting `zstd` dictionary creation. Note: compressed data created with a dictionary *must* be decompressed with the same dictionary.

Value

External pointer to a ZSTD Decompression Context which can be passed to `zstd_unserialize()` and `zstd_decompress()`

Examples

```
dctx <- zstd_dctx(validate_checksum = FALSE)
```

`zstd_dctx_settings` *Get the configuration settings of a decompression context*

Description

Get the configuration settings of a decompression context

Usage

```
zstd_dctx_settings(dctx)
```

Arguments

`dctx`	ZSTD decompression context, as created by `zstd_dctx()`

Value

named list of configuration options

Examples

```
dctx <- zstd_dctx()
zstd_dctx_settings(dctx)
```

`zstd_dict_id` *Get the Dictionary ID of a dictionary or a vector compressed data.*

Description

Dictionary IDs are generated automatically when a dictionary is created. When using a dictionary for compression, the same dictionary must be used during decompression. ZSTD internally does this check for matching IDs when attempting to decompress. This function exposes the dictionary ID to aid in handling and tracking dictionaries in R.

Usage

```
zstd_dict_id(dict)
```

Arguments

`dict`	raw vector or filename. This object could contain either a zstd dictionary, or a compressed object. If it is a compressed object, then it will return the dictionary id which was used to compress it.

Value

Signed integer value representing the Dictionary ID. If data does not represent a dictionary, or data which was compressed with a dictionary, then a value of 0 is returned.

Examples

```
dict_file <- system.file("sample_dict.raw", package = "zstdlite", mustWork = TRUE)
dict <- readBin(dict_file, raw(), file.size(dict_file))
zstd_dict_id(dict)
compressed_mtcars <- zstd_serialize(mtcars, dict = dict)
zstd_dict_id(compressed_mtcars)
```

zstd_serialize — *Serialize/Unserialize arbitrary R objects to a compressed stream of bytes using Zstandard*

Description

Serialize/Unserialize arbitrary R objects to a compressed stream of bytes using Zstandard

Usage

```
zstd_serialize(robj, ..., file = NULL, cctx = NULL, use_file_streaming = FALSE)

zstd_unserialize(src, ..., dctx = NULL, use_file_streaming = FALSE)
```

Arguments

`robj`	Any R object understood by `base::serialize()`
`...`	extra arguments passed to `zstd_cctx()` or `zstd_dctx()` context initializers. Note: These argument are only used when `cctx` or `dctx` is NULL
`file`	filename in which to serialize data. If NULL (the default), then serialize the results to a raw vector
`cctx`	ZSTD Compression Context created by `zstd_cctx()` or NULL. Default: NULL will create a default compression context on-the-fly
`use_file_streaming`	Use the streaming interface when reading or writing to a file? This may reduce memory allocations and make better use of mutlithreading. Default: FALSE
`src`	Raw vector or filename containing a ZSTD compressed serialized representation of an R object
`dctx`	ZSTD Decompression Context created by `zstd_dctx()` or NULL. Default: NULL will create a default decompression context on-the-fly.

Value

Raw vector of compressed serialized data, or `NULL` if file created with compressed data

Examples

```
vec <- zstd_serialize(mtcars)
zstd_unserialize(vec)

tmp <- tempfile()
zstd_serialize(mtcars, file = tmp)
zstd_unserialize(tmp)
```

zstd_train_dict_compress *Train a dictionary for use with* `zstd_compress()` *and* `zstd_decompress()`

Description

This function requires multiple samples representative of the expected data to train a dictionary for use during compression.

Usage

```
zstd_train_dict_compress(
  samples,
  size = 1e+05,
  optim = FALSE,
  optim_shrink_allow = 0
)
```

Arguments

`samples`	list of raw vectors, or length-1 character vectors. Each raw vector or string, should be a complete example of something to be compressed with `zstd_compress()`
`size`	Maximum size of dictionary in bytes. Default: 112640 (110 kB) matches the default size set by the command line version of `zstd`. Actual dictionary created may be smaller than this if (1) there was not enough training data to make use of this size (2) `optim_shrink_allow` was set and a smaller dictionary was found to be almost as useful.
`optim`	optimize the dictionary. Default FALSE. If TRUE, then ZSTD will spend time optimizing the dictionary. This can be a very length operation.
`optim_shrink_allow`	integer value representing a percentage. If non-zero, then a search will be carried out for dictionaries of a smaller size which are up to `optim_shrink_allow` percent worse than the maximum sized dictionary. Default: 0 means that no shrinking will be done.

Value

raw vector containing a ZSTD dictionary

Examples

```
# This example shows the mechanics of creating and training a dictionary but
# may not be a great example of when a dictionary might be useful
cars <- rownames(mtcars)
samples <- lapply(seq_len(1000), \(x) serialize(sample(cars), NULL))
zstd_train_dict_compress(samples, size = 5000)
```

zstd_train_dict_serialize *Train a dictionary for use with* `zstd_serialize()` *and* `zstd_unserialize()`

Description

Train a dictionary for use with `zstd_serialize()` and `zstd_unserialize()`

Usage

```
zstd_train_dict_serialize(
  samples,
  size = 1e+05,
  optim = FALSE,
  optim_shrink_allow = 0
)
```

Arguments

`samples`	list of example R objects to train a dictionary to be used with `zstd_serialize()`
`size`	Maximum size of dictionary in bytes. Default: 112640 (110 kB) matches the default size set by the command line version of `zstd`. Actual dictionary created may be smaller than this if (1) there was not enough training data to make use of this size (2) `optim_shrink_allow` was set and a smaller dictionary was found to be almost as useful.
`optim`	optimize the dictionary. Default FALSE. If TRUE, then ZSTD will spend time optimizing the dictionary. This can be a very length operation.
`optim_shrink_allow`	integer value representing a percentage. If non-zero, then a search will be carried out for dictionaries of a smaller size which are up to `optim_shrink_allow` percent worse than the maximum sized dictionary. Default: 0 means that no shrinking will be done.

Value

raw vector containing a ZSTD dictionary

Examples

```
# This example shows the mechanics of creating and training a dictionary but
# may not be a great example of when a dictionary might be useful
cars <- rownames(mtcars)
samples <- lapply(seq_len(1000), \(x) sample(cars))
zstd_train_dict_serialize(samples, size = 5000)
```

zstd_version *Get version string of zstd C library*

Description

Get version string of zstd C library

Usage

```
zstd_version()
```

Value

String containing version number of zstd C library

Examples

```
zstd_version()
```

Index