# Package: Rarr (via r-universe)

November 2, 2026

**Title** A Simple and Performant Native R Reader & Writer for Zarr Arrays

**Version** 2.1.53

**Description** The Zarr specification defines a format for chunked, compressed, N-dimensional arrays. Its design allows efficient access to subsets of the stored array, and supports both local and cloud storage systems. Rarr aims to implement this specification in R with minimal reliance on an external tools or libraries.

**License** MIT + file LICENSE

**URL** https://huber-group-embl.github.io/Rarr/, https://github.com/Huber-group-EMBL/Rarr

**BugReports** https://github.com/Huber-group-EMBL/Rarr/issues

**Depends** R (>= 4.2.0)

**Imports** curl, grumpy, jsonlite, paws.storage, R.utils, rlang, utils

**Suggests** BiocStyle, knitr, progressr, rmarkdown, testthat (>= 3.0.0), withr

**VignetteBuilder** knitr

**biocViews** DataImport

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**SystemRequirements** GNU make

**Config/testthat/edition** 3

**Config/roxygen2/version** 8.1.1

**Config/pak/sysreqs** make libxml2-dev libssl-dev

**Repository** https://grimbough.r-universe.dev

**Date/Publication** 2026-11-02 10:01:32 UTC

**RemoteUrl** https://github.com/grimbough/Rarr

**RemoteRef** HEAD

**RemoteSha** bfe7c0fe0be4614b57591b0cb1e0c5155a586a55

# Contents

compressors . . . . . 2
create_empty_zarr_array . . . . . 4
read_zarr_array . . . . . 5
read_zarr_attributes . . . . . 7
read_zarr_consolidated_metadata . . . . . 8
update_zarr_array . . . . . 9
write_zarr_array . . . . . 10
write_zarr_attributes . . . . . 12
write_zarr_group . . . . . 13
zarr_consolidate_metadata . . . . . 13
zarr_overview . . . . . 15

**Index** **17**

---

`compressors` *Define compression tool and settings*

---

## Description

These functions select a compression tool and its setting when writing a Zarr file

## Usage

```
use_blosc(
  cname = c("lz4", "lz4hc", "blosclz", "zstd", "zlib", "snappy"),
  clevel = 5L,
  shuffle = c("shuffle", "noshuffle", "bitshuffle"),
  typesize = NULL,
  blocksize = 0L
)

use_zlib(level = 6L)

use_gzip(level = 6L)

use_bz2(level = 9L)

use_lzma(level = 9L)

use_lz4()

use_zstd(level = 0L)
```

## Arguments

| | |
|---|---|
| cname | Blosc is a 'meta-compressor' providing access to several compression algorithms. This argument defines which compression tool should be used. Valid options are: "lz4", "lz4hc", "blosclz", "zstd", "zlib", "snappy". |
| clevel | An integer from 0 to 9 which controls the speed and level of compression. A level of 1 is the fastest compression method and produces the least compressions, while 9 is slowest and produces the most compression. Compression is turned off completely when level is 0. Defaults to 5. |
| shuffle | Specifies the type of shuffling to perform, if any, prior to compression. Must be one of "noshuffle", to indicate no shuffling; "shuffle" (default), to indicate byte-wise shuffling; "bitshuffle", to indicate bit-wise shuffling. |
| typesize | The data type size in bytes used by Blosc shuffling. If NULL (default), this will be inferred from the array datatype. Ignored if shuffle = "noshuffle". |
| blocksize | The requested size of the compressed blocks in bytes. Use 0 (default) to let Blosc choose automatically. |
| level | Specify the compression level to use. The range of possible values is dependant on the compression tool being used. For example, for use_zlib() this argument can be between 1 & 9, while for use_zstd()the valid range is 0 (default; level chosen automatically) to 22. |

## Value

A list containing the details of the selected compression tool. This will be written to the .zarray metadata when the Zarr array is created.

## Examples

```
## define 2 compression filters for blosc (using snappy) and bzip2 (level 5)
blosc_with_snappy_compression <- use_blosc(cname = "snappy")
bzip2_compression <- use_bz2(level = 5)

## create an example array to write to a file
x <- array(runif(n = 1000, min = -10, max = 10), dim = c(10, 20, 5))

## write the array to two files using each compression filter
blosc_path <- tempfile()
bzip2_path <- tempfile()
write_zarr_array(
  x = x, zarr_array_path = blosc_path, chunk_dim = c(2, 5, 1),
  compressor = blosc_with_snappy_compression
)
write_zarr_array(
  x = x, zarr_array_path = bzip2_path, chunk_dim = c(2, 5, 1),
  compressor = bzip2_compression
)

## the contents of the two arrays should be the same
identical(read_zarr_array(blosc_path), read_zarr_array(bzip2_path))
```

```
## the size of the files on disk are not the same
sum(file.size(list.files(blosc_path, full.names = TRUE)))
sum(file.size(list.files(bzip2_path, full.names = TRUE)))
```

---

`create_empty_zarr_array` *Create an (empty) Zarr array*

---

## Description

Create an (empty) Zarr array

## Usage

```
create_empty_zarr_array(
  zarr_array_path,
  dim,
  chunk_dim,
  data_type,
  order = c("F", "C"),
  compressor = use_zstd(),
  fill_value = NULL,
  nchar = NULL,
  dimension_separator = if (zarr_version == 2L) "." else "/",
  dimension_names = NULL,
  zarr_version = 3L
)
```

## Arguments

| | |
|---|---|
| `zarr_array_path` | Character vector of length 1 giving the path to the new Zarr array. |
| `dim` | Dimensions of the new array. Should be a numeric vector with the same length as the number of dimensions. |
| `chunk_dim` | Dimensions of the array chunks. Should be a numeric vector with the same length as the `dim` argument. |
| `data_type` | Character vector giving the data type of the new array. Valid options are: "integer", "double", "character", "logical", which are based on standard R data types. You can also use the analogous Numpy formats: "\|i1", "<i2", "<i4", "<f4", "<f8", "\|S", "\|b1". If this argument isn't provided the `fill_value` will be used to determine the datatype. |
| `order` | Define the layout of the bytes within each chunk. Valid options are 'column', 'row', 'F' & 'C'. 'column' or 'F' will specify "column-major" ordering, which is how R arrays are arranged in memory. 'row' or 'C' will specify "row-major" order. |

| | |
|---|---|
| compressor | What (if any) compression tool should be applied to the array chunks. The default is to use zstd compression. Supplying NULL will disable chunk compression. See compressors for more details. |
| fill_value | The default value for uninitialized portions of the array. Does not have to be provided, in which case the default for the specified data type will be used. |
| nchar | For datatype = "character" this parameter gives the maximum length of the stored strings. It is an error not to specify this for a character array, but it is ignored for other data types. |
| dimension_separator | The character used to to separate the dimensions in the names of the chunk files. Valid options are limited to "." and "/". |
| dimension_names | Optional character vector with the same length as dim. |
| zarr_version | The version of the Zarr specification to use. Currently, either 2 or 3. The default is 3. |

## Value

This function is primarily called for the side effect of initialising a Zarr array location and creating the .zarray or zarr.json metadata file. Returns (invisibly) the normalized path it wrote the metadata to.

## See Also

write_zarr_array(), update_zarr_array()

## Examples

```
new_zarr_array <- file.path(tempdir(), "temp.zarr")
create_empty_zarr_array(new_zarr_array,
  dim = c(10, 20), chunk_dim = c(2, 5),
  data_type = "integer"
)
```

---

# read_zarr_array *Read a Zarr array*

---

## Description

Read a Zarr array

## Usage

```
read_zarr_array(zarr_array_path, index, s3_client = NULL)
```

## Arguments

| | |
|---|---|
| zarr_array_path | Path to a Zarr array. A character vector of length 1. This can either be a location on a local file system or the URI to an array in S3 storage. |
| index | A list of the same length as the number of dimensions in the Zarr array. Each entry in the list provides the indices in that dimension that should be read from the array. Setting a list entry to NULL will read everything in the associated dimension. If this argument is missing the entirety of the the Zarr array will be read. |
| s3_client | Object created by paws.storage::s3(). Only required for a file on S3. Leave as NULL for a file on local storage. |

## Details

This function makes use of the **progressr** package to signal progress. For large / complex arrays, users may want to enable the progress bar with progressr::handlers().

## Value

An array with the same number of dimensions as the input array. The extent of each dimension will correspond to the length of the values provided to the index argument.

## Examples

```
## Using a local file provided with the package
## This array has 3 dimensions
z1 <- system.file("extdata", "zarr_examples", "row-first", "int32.zarr", package = "Rarr")

## read the entire array
read_zarr_array(zarr_array_path = z1)

## extract values for first 10 rows, all columns, first slice
read_zarr_array(zarr_array_path = z1, index = list(1:10, NULL, 1))


## using a Zarr file hosted on Amazon S3
## This array has a single dimension with length 2729077
z2 <- "https://noaa-nwm-retro-v2-zarr-pds.s3.amazonaws.com/feature_id/"

## read the entire array
read_zarr_array(zarr_array_path = z2)

## read alternating elements
read_zarr_array(zarr_array_path = z2, index = list(seq(1, 576, 2)))
```

`read_zarr_attributes` *Read the attributes associated with a Zarr array or group*

## Description

Read the attributes associated with a Zarr array or group

## Usage

```
read_zarr_attributes(
  zarr_path,
  s3_client = NULL,
  missing = c("ignore", "warning", "error")
)
```

## Arguments

| | |
|---|---|
| `zarr_path` | A character vector of length 1. This provides the path to a Zarr array or group of arrays. This can either be on a local file system or on S3 storage. |
| `s3_client` | A list representing an S3 client. This should be produced by `paws.storage::s3()`. |
| `missing` | A character vector of length 1. This determines the behaviour when no file containing attributes is found. This can be one of:<br>• "ignore" (the default): `NULL` is returned silently<br>• "warning": a warning is issued and `NULL` is returned.<br>• "error": an error is raised. |

## Value

A list containing the attributes. If the file containing attributes (`.zattrs` for Zarr v2 or `zarr.json` for Zarr v3) exists but no attributes are provided, `NULL` is returned.

## Examples

```
read_zarr_attributes(
  "https://uk1s3.embassy.ebi.ac.uk/idr/zarr/v0.4/idr0048A/9846152.zarr"
)
```

`read_zarr_consolidated_metadata` *Read consolidated metadata*

## Description

Read consolidated metadata

## Usage

```
read_zarr_consolidated_metadata(
  zarr_path,
  consolidate = c("never", "missing", "always"),
  s3_client = NULL
)
```

## Arguments

| | |
|---|---|
| `zarr_path` | A character vector of length 1. This provides the path to a Zarr array or group of arrays. This can either be on a local file system or on S3 storage. |
| `consolidate` | One of `"never"` (default), `"missing"` or `"always"`. This controls whether the consolidated metadata needs to be (re)generated on the fly, in memory. If `"never"` then the consolidated metadata is only read if it already exists, and `NULL` is returned otherwise. If `"missing"` then the consolidated metadata is generated if it does not already exist. If `"always"` then the consolidated metadata is always (re)generated, even if it already exists. |
| `s3_client` | A list representing an S3 client. This should be produced by `paws.storage::s3()`. |

## Details

This is stored in the `.zmetadata` (Zarr v2) or `zarr.json` (Zarr v3) file at the root of a Zarr store.

Note that it is not documented in the official Zarr specification, because it is not (yet?) part of the standard.

In particular, it lists the location of all the metadata files for arrays in the current group, so it is not necessary to crawl to discover them. It leads to much better performance, in particular when reading arrays from S3 storage.

## Value

A list containing the consolidated metadata, formatted as Zarr v3 consolidated metadata. If `consolidate = "never"` and the consolidated metadata does not exist, `NULL` is returned.

## References

https://zarr.readthedocs.io/en/latest/user-guide/consolidated_metadata.html

## Examples

```
z <- system.file("extdata", "zarr_examples", "metadata", "consolidated.zarr", package = "Rarr")
m <- read_zarr_consolidated_metadata(z)
head(m)
```

---

`update_zarr_array` *Update (a subset of) an existing Zarr array*

---

## Description

Update (a subset of) an existing Zarr array

## Usage

```
update_zarr_array(zarr_array_path, x, index)
```

## Arguments

| | |
|---|---|
| `zarr_array_path` | Character vector of length 1 giving the path to the Zarr array that is to be modified. |
| `x` | The R array (or object that can be coerced to an array) that will be written to the Zarr array. |
| `index` | A list with the same length as the number of dimensions of the target array. This argument indicates which elements in the target array should be updated. |

## Details

This function makes use of the **progressr** package to signal progress. For large / complex arrays, users may want to enable the progress bar with `progressr::handlers()`.

## Value

The function is primarily called for the side effect of writing to disk. Returns (invisibly) `TRUE` if the array is successfully updated.

## Examples

```
## first create a new, empty, Zarr array
new_zarry_array <- file.path(tempdir(), "new_array.zarr")
create_empty_zarr_array(
  zarr_array_path = new_zarry_array, dim = c(20, 10),
  chunk_dim = c(10, 5), data_type = "double"
)

## create a matrix smaller than our Zarr array
small_matrix <- matrix(runif(6), nrow = 3)
```

```
## insert the matrix into the first 3 rows, 2 columns of the Zarr array
update_zarr_array(new_zarry_array, x = small_matrix, index = list(1:3, 1:2))

## reading back a slightly larger subset,
## we can see only the top left corner has been changed
read_zarr_array(new_zarry_array, index = list(1:5, 1:5))
```

---

`write_zarr_array` *Write an R array to Zarr*

---

## Description

Write an R array to Zarr

## Usage

```
write_zarr_array(
  x,
  zarr_array_path,
  chunk_dim,
  data_type = storage.mode(x),
  order = c("F", "C"),
  compressor = use_zstd(),
  fill_value = NULL,
  nchar,
  dimension_separator = if (zarr_version == 2L) "." else "/",
  zarr_version = 3L
)
```

## Arguments

| | |
|---|---|
| `x` | The R array that will be written to the Zarr array. |
| `zarr_array_path` | Character vector of length 1 giving the path to the new Zarr array. |
| `chunk_dim` | Dimensions of the array chunks. Should be a numeric vector with the same length as the `dim` argument. |
| `data_type` | Character vector giving the data type of the new array. Valid options are: "integer", "double", "character", "logical", which are based on standard R data types. You can also use the analogous Numpy formats: "\|i1", "<i2", "<i4", "<f4", "<f8", "\|S", "\|b1". If this argument isn't provided the `fill_value` will be used to determine the datatype. |
| `order` | Define the layout of the bytes within each chunk. Valid options are 'column', 'row', 'F' & 'C'. 'column' or 'F' will specify "column-major" ordering, which is how R arrays are arranged in memory. 'row' or 'C' will specify "row-major" order. |

`compressor` What (if any) compression tool should be applied to the array chunks. The default is to use `zstd` compression. Supplying `NULL` will disable chunk compression. See compressors for more details.

`fill_value` The default value for uninitialized portions of the array. Does not have to be provided, in which case the default for the specified data type will be used.

`nchar` For character arrays this parameter gives the maximum length of the stored strings. If this argument is not specified the array provided to `x` will be checked and the length of the longest string found will be used so no data are truncated. However this may be slow and providing a value to `nchar` can provide a modest performance improvement.

`dimension_separator` The character used to to separate the dimensions in the names of the chunk files. Valid options are limited to "." and "/".

`zarr_version` The version of the Zarr specification to use. Currently, either `2` or `3`. The default is `3`.

## Details

This function makes use of the **progressr** package to signal progress. For large / complex arrays, users may want to enable the progress bar with `progressr::handlers()`.

## Value

The function is primarily called for the side effect of writing to disk. Returns (invisibly) `TRUE` if the array is successfully written.

## Note

If `x` has `dimnames`, `names(dimnames(x))` will be stored as the `dimension_names` field in the Zarr metadata.

## Examples

```
new_zarr_array <- file.path(tempdir(), "integer.zarr")
x <- array(1:50, dim = c(10, 5))
write_zarr_array(
  x = x, zarr_array_path = new_zarr_array,
  chunk_dim = c(2, 5)
)
```

`write_zarr_attributes` *Read the .zattrs file associated with a Zarr array or group*

## Description

Read the .zattrs file associated with a Zarr array or group

## Usage

```
write_zarr_attributes(
  zarr_path,
  new.zattrs = list(),
  overwrite = TRUE,
  zarr_version = if (has_metadata_v2) 2L else 3L
)
```

## Arguments

| | |
|---|---|
| `zarr_path` | A character vector of length 1. This provides the path to a Zarr array or group. |
| `new.zattrs` | a list inserted to .zattrs at the `path`. |
| `overwrite` | if `TRUE` (the default), existing .zattrs elements will be overwritten by `new.zattrs`. |
| `zarr_version` | The version of the Zarr specification to use. If a metadata file already exists, the version will be inferred from the file. Otherwise, the default is `3`. |

## Value

Invisibly, the updated attributes as a named list. This is equivalent to (but faster than) using `read_zarr_attributes()` after writing. If no attributes were present before, this is identical to `new.zattrs`.

## Examples

```
z1 <- withr::local_tempdir(fileext = ".zarr")
create_empty_zarr_array(
  z1,
  fill_value = 0L,
  dim = c(10, 10),
  chunk_dim = c(2, 5),
  zarr_version = 3L
)
write_zarr_attributes(z1, list(date = "2025-01-01", author = "Jane Doe"))
```

## write_zarr_group *Initialize a Zarr group*

### Description

Initialize a Zarr group

### Usage

```
write_zarr_group(
  zarr_path,
  group,
  zarr_version = if (has_metadata_v2) 2L else 3L
)
```

### Arguments

| | |
|---|---|
| zarr_path | A character vector of length 1. This provides the path to a Zarr store. |
| group | A character vector of length 1. This provides the name of the group to create. If "", the root group will be created. |
| zarr_version | The version of the Zarr specification to use. If a metadata file already exists, the version will be inferred from the file. Otherwise, the default is 3. |

### Details

Nested groups are created recursively. For example, if group = "foo/bar", then the group foo will be created first, followed by the group bar inside of it.

### Examples

```
zarr_v2 <- withr::local_tempfile(fileext = ".zarr")
write_zarr_group(zarr_v2, "test/deep/nested/group", zarr_version = 2L)
```

## zarr_consolidate_metadata *Consolidate Zarr metadata files into a single file*

### Description

This function reads all the metadata files in a Zarr store and consolidates them into a single file. Thanks to this, a single request can be made to retrieve all the elements and their related metadata for a Zarr store, which is especially beneficial for remote stores like S3.

## Usage

```
zarr_consolidate_metadata(
  zarr_store_path,
  s3_client = NULL,
  action = c("write", "return"),
  overwrite = TRUE
)
```

## Arguments

| | |
|---|---|
| zarr_store_path | A character vector of length 1. This provides the path to a Zarr store. |
| s3_client | A list representing an S3 client. This should be produced by paws.storage::s3(). |
| action | A character string specifying the action to take with the consolidated metadata. If "write" (the default), the consolidated metadata will be written back to the Zarr store. If "return", the consolidated metadata will be returned as a list without writing it back to the store. The latter is particularly useful for non-writable stores. |
| overwrite | A logical value (default TRUE) indicating whether to overwrite existing consolidated metadata when action is "write". If FALSE and consolidated metadata already exists, an error will be raised. |

## Value

If action is "return", a list containing the consolidated metadata. Otherwise, the function is called for its side effect and NULL is returned invisibly.

## Examples

```
# v2
zarr_v2 <- withr::local_tempfile(fileext = ".zarr")
write_zarr_group(zarr_v2, "", zarr_version = 2L)
write_zarr_array(
  array(1:4, dim = c(2, 2)),
  file.path(zarr_v2, "array1"),
  chunk_dim = c(1, 2),
  zarr_version = 2L
)
write_zarr_array(
  array(c(3.14, 42.42, 12.96, 7.89), dim = c(2, 2)),
  file.path(zarr_v2, "array2"),
  chunk_dim = c(1, 2),
  zarr_version = 2L
)
write_zarr_attributes(
 file.path(zarr_v2, "array1"),
 list(description = "This is array 1")
)
zarr_consolidate_metadata(zarr_v2, action = "return")
```

```
zarr_consolidate_metadata(zarr_v2, action = "write")
zarr_overview(zarr_v2)
```

---

`zarr_overview` *Print a summary of a Zarr array or group*

---

## Description

When reading a Zarr array using `read_zarr_array()` it is necessary to know it's shape and size. `zarr_overview()` can be used to get a quick overview of the array shape and contents, based on the `.zarray` (Zarr v2) or `zarr.json` (Zarr v3) metadata file each array contains.

## Usage

```
zarr_overview(zarr_array_path, s3_client = NULL, as_data_frame = FALSE)
```

## Arguments

| | |
|---|---|
| `zarr_array_path` | A character vector of length 1. This provides the path to a Zarr array or group of arrays. This can either be on a local file system or on S3 storage. |
| `s3_client` | A list representing an S3 client. This should be produced by `paws.storage::s3()`. |
| `as_data_frame` | Logical determining whether the Zarr array details should be printed to screen (`FALSE`) or returned as a `data.frame` (`TRUE`) so they can be used computationally. |

## Details

The function currently prints the following information to the R console:

- array path
- array shape and size
- chunk and size
- the number of chunks
- the datatype of the array
- codec used for data compression (if any)

If given the path to a group of arrays the function will attempt to print the details of all sub-arrays in the group.

## Value

If `as_data_frame = FALSE` the function invisible returns `TRUE` if successful. However it is primarily called for the side effect of printing details of the Zarr array(s) to the screen. If `as_data_frame = TRUE` then a `data.frame` containing details of the array is returned.

## Examples

```
## Using a local file provided with the package
z1 <- system.file("extdata", "zarr_examples", "row-first",
  "int32.zarr",
  package = "Rarr"
)

## read the entire array
zarr_overview(zarr_array_path = z1)

## using a file on S3 storage

z2 <- "https://noaa-nwm-retro-v2-zarr-pds.s3.amazonaws.com/feature_id/"
zarr_overview(z2)
```

# Index

compressors, 2, *5*, *11*
create_empty_zarr_array, 4

paws.storage::s3(), *6–8*, *14*, *15*
progressr::handlers(), *6*, *9*, *11*

read_zarr_array, 5
read_zarr_array(), *15*
read_zarr_attributes, 7
read_zarr_consolidated_metadata, 8

update_zarr_array, 9
update_zarr_array(), *5*
use_blosc *(*compressors*)*, 2
use_bz2 *(*compressors*)*, 2
use_gzip *(*compressors*)*, 2
use_lz4 *(*compressors*)*, 2
use_lzma *(*compressors*)*, 2
use_zlib *(*compressors*)*, 2
use_zstd *(*compressors*)*, 2

write_zarr_array, 10
write_zarr_array(), *5*
write_zarr_attributes, 12
write_zarr_group, 13

zarr_consolidate_metadata, 13
zarr_overview, 15