

# Package ‘Rarr’

May 26, 2026

**Title** Read Zarr Files in R

**Version** 2.1.14

**Description** The Zarr specification defines a format for chunked, compressed, N-dimensional arrays. It's 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 (>= 4.1.0)

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

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

**VignetteBuilder** knitr

**biocViews** DataImport

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**SystemRequirements** GNU make

**Config/testthat/edition** 3

**Config/roxygen2/version** 8.0.0

**git\_url** <https://git.bioconductor.org/packages/Rarr>

**git\_branch** devel

**git\_last\_commit** e58bdc6

**git\_last\_commit\_date** 2026-05-25

**Repository** Bioconductor 3.24

**Date/Publication** 2026-05-25

**Author** Mike Smith [aut, ccp] (ORCID: <<https://orcid.org/0000-0002-7800-3848>>, Maintainer from 2022 to 2025.),  
 Hugo Gruson [aut, cre] (ORCID: <<https://orcid.org/0000-0002-4094-1476>>),  
 Artür Manukyan [ctb],  
 Sharla Gelfand [ctb],  
 German Network for Bioinformatics Infrastructure - de.NBI [fnd] (ROR: <<https://ror.org/01vmpm840>>)

**Maintainer** Hugo Gruson <[hugo.gruson@embl.de](mailto:hugo.gruson@embl.de)>

## Contents

Rarr-package . . . . .	2
.chunk_positions_by_chunk . . . . .	3
.compress_and_write_chunk . . . . .	4
.configure_codecs . . . . .	4
.extract_chunk . . . . .	5
.get_credentials . . . . .	5
.normalize_array_path . . . . .	6
.normalize_v3_metadata . . . . .	6
.prepare_write_data . . . . .	7
.read_array_metadata . . . . .	7
.read_consolidated_metadata . . . . .	8
.read_json_file . . . . .	8
compressors . . . . .	9
create_empty_zarr_array . . . . .	10
read_chunk . . . . .	12
read_zarr_array . . . . .	12
read_zarr_attributes . . . . .	13
update_zarr_array . . . . .	14
write_zarr_array . . . . .	15
write_zarr_attributes . . . . .	16
ZarrArray . . . . .	17
zarr_consolidate_metadata . . . . .	18
zarr_overview . . . . .	19
<b>Index</b>	<b>21</b>

---

Rarr-package

*Rarr: Read Zarr Files in R*

---

## Description

The Zarr specification defines a format for chunked, compressed, N-dimensional arrays. It's 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.

**Author(s)**

**Maintainer:** Hugo Gruson <hugo.gruson@embl.de> ([ORCID](#))

Authors:

- Hugo Gruson <hugo.gruson@embl.de> ([ORCID](#))
- Mike Smith ([ORCID](#)) (Maintainer from 2022 to 2025.) [conceptor]

Other contributors:

- Artür Manukyan [contributor]
- Sharla Gelfand [contributor]
- German Network for Bioinformatics Infrastructure - de.NBI ([ROR](#)) [funder]

**See Also**

Useful links:

- <https://huber-group-embl.github.io/Rarr/>
- <https://github.com/Huber-group-EMBL/Rarr>
- Report bugs at <https://github.com/Huber-group-EMBL/Rarr/issues>

---

`.chunk_positions_by_chunk`

*Precompute index positions grouped by chunk*

---

**Description**

For each chunk touched by `index`, returns the positions (1-based) within each dimension of `index` that fall inside that chunk, together with the within-chunk indices needed to extract values from the chunk array.

**Usage**

```
.chunk_positions_by_chunk(index, metadata)
```

**Arguments**

<code>index</code>	A list of integer vectors, one per dimension, giving the requested array indices (1-based).
<code>metadata</code>	List of array metadata as returned by <code>.read_array_metadata()</code> . Used to derive chunk name keys via <code>.create_chunk_names()</code> .

**Value**

A named list keyed by chunk names (same format as `.create_chunk_names()`, e.g. "c/0/1/0" for Zarr V3). Each element is a list with two components:

- `positions`: a per-dimension list of integer vectors of positions into the corresponding `index` vector that map to that chunk.
- `index_in_chunk`: a per-dimension list of 1-based integer vectors giving the within-chunk coordinates corresponding to `positions`.

---

```
.compress_and_write_chunk
```

*Compress and write a single chunk*

---

### Description

Compress and write a single chunk

### Usage

```
.compress_and_write_chunk(input_chunk, chunk_path, metadata)
```

### Arguments

input_chunk	Array containing the chunk data to be compressed. Will be converted to a raw vector before compression.
chunk_path	Character string giving the path to the chunk that should be written.

### Value

Returns TRUE if writing is successful. Mostly called for the side-effect of writing the compressed chunk to disk.

---

```
.configure_codecs
```

*Convert the metadata codecs elements into functions*

---

### Description

Convert the metadata codecs elements into functions

### Usage

```
.configure_codecs(codecs, operation = c("encode", "decode"))
```

### Arguments

codecs	A list containing the Zarr v3 codecs
operation	One of "encode" or "decode"

### Value

An list of 3 environments containing functions:

- bytes\_bytes\_codecs: functions to encode/decode raw bytes
- array\_array\_codecs: functions to encode/decode R arrays
- array\_bytes\_codecs: functions to encode/decode between R arrays and raw bytes

---

.extract\_chunk                    *Subset extraction for an array with a variable number of dimensions.*

---

### Description

Subset extraction for an array with a variable number of dimensions.

### Usage

```
.extract_chunk(x, idx)
```

### Arguments

x                    Array to extract from.  
idx                  List of index vectors, one per dimension.

### Value

The extracted sub-array (with drop = FALSE).

---

.get\_credentials                *This is a modified version of paws.storage:::get\_credentials(). It is included to prevent using the ::: operator. Look at that function if things stop working.*

---

### Description

This is a modified version of paws.storage:::get\_credentials(). It is included to prevent using the ::: operator. Look at that function if things stop working.

### Usage

```
.get_credentials(credentials)
```

### Arguments

credentials            Content stored at .internal\$config\$credentials in an object created by paws.storage::s3().

### Value

A credentials list to be reinserted into a paws.storage s3 object. If no valid credentials are found this function will error, which is expected and is caught by .check\_credentials.

---

`.normalize_array_path` *Normalize a Zarr array path*

---

### Description

Taken from <https://zarr.readthedocs.io/en/stable/spec/v2.html#logical-storage-paths>

### Usage

```
.normalize_array_path(path)
```

### Arguments

`path` Character vector of length 1 giving the path to be normalised.

### Value

A character vector of length 1 containing the normalised path.

---

`.normalize_v3_metadata`

*Apply Zarr v3-specific fixups to raw array metadata*

---

### Description

Called after reading a `zarr.json` file for an array node. Performs four normalisation steps that are needed before the metadata can be used by the rest of the package:

### Usage

```
.normalize_v3_metadata(metadata)
```

### Arguments

`metadata` A list as returned by `.read_json_file()` for a Zarr v3 array node (i.e. `metadata$zarr_format == 3L` and `metadata$node_type == "array"`).

### Details

1. Parse the `data_type` string into an R-friendly datatype list.
2. Name codecs by their name field for O(1) lookup.
3. Normalise scalar arrays (empty shape) to shape 1.
4. Inject a default transpose codec when absent (R uses F-order so we always need to know the intended order).
5. Replicate the endian field across all fields of struct datatypes so that downstream chunk-reading code can treat it uniformly.

### Value

The modified metadata list.

---

.prepare\_write\_data *Truncate overflow values and warn about bool NA values before writing*

---

### Description

Applies the two pre-write data checks that are needed identically in both `write_zarr_array()` and `update_zarr_array()`:

### Usage

```
.prepare_write_data(x, metadata)
```

### Arguments

`x` The R array to be written.  
`metadata` Array metadata list as returned by `.read_array_metadata()`.

### Details

1. If the target type is narrower than R's native representation (e.g. `int8`, `int16`, `float32`), clamp out-of-range values via `.truncate_overflow()`.
2. If the target type is `bool`, warn when `x` contains NA values (which the Zarr bool type cannot represent).

### Value

`x`, possibly with overflow values clamped.

---

.read\_array\_metadata *Read the .zarray or zarr.json metadata file associated with a Zarr array*

---

### Description

Read the `.zarray` or `zarr.json` metadata file associated with a Zarr array

### Usage

```
.read_array_metadata(zarr_path, 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.  
`s3_client` A list representing an S3 client. This should be produced by `paws.storage::s3()`.

### Value

A list containing the array metadata

---

`.read_consolidated_metadata`  
*Read consolidated metadata file*

---

**Description**

Read consolidated metadata file

**Usage**

```
.read_consolidated_metadata(
    zarr_path,
    nodes = c("group", "array"),
    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.

`s3_client`        A list representing an S3 client. This should be produced by `paws.storage::s3()`.

**Details**

This is stored in the `.zmetadata` 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.

It is implemented in `zarr-python` and discussed under the "consolidated metadata" phrase.

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.

**References**

[https://zarr.readthedocs.io/en/latest/user-guide/consolidated\\_metadata.html](https://zarr.readthedocs.io/en/latest/user-guide/consolidated_metadata.html)

---

`.read_json_file`        *Read a JSON file from local disk or S3*

---

**Description**

Read a JSON file from local disk or S3

**Usage**

```
.read_json_file(path, s3_client = NULL)
```

**Arguments**

`path`                Full path (local or S3) to a JSON file.

`s3_client`        An S3 client produced by `paws.storage::s3()`, or NULL for local files.

**Value**

A list as returned by `jsonlite::read_json()` / `jsonlite::fromJSON()`.

---

compressors	<i>Define compression tool and settings</i>
-------------	---

---

**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 = 6L)

use_lzma(level = 9L)

use_lz4()

use_zstd(level = 3L)
```

**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 1 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 = "F",
  compressor = use_zstd(),
  fill_value,
  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", "iS", "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 <a href="#">compressors</a> 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 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

If successful returns (invisibly) TRUE. However this function is primarily called for the side effect of initialising a Zarr array location and creating the .zarr metadata.

### 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_chunk	<i>Read a single Zarr chunk</i>
------------	---------------------------------

---

### Description

Read a single Zarr chunk

### Usage

```
read_chunk(chunk_path, metadata, s3_client = NULL)
```

### Arguments

chunk_path	A character vector of length 1, giving the path to the chunk to be read.
metadata	List produced by <code>.read_array_metadata()</code> holding the contents of the <code>.zarray</code> file. If missing this function will be called automatically, but it is probably preferable to pass the meta data rather than read it repeatedly for every chunk.
s3_client	Object created by <code>paws.storage::s3()</code> . Only required for a file on S3. Leave as NULL for a file on local storage.

### Value

An array containing the decompressed chunk values.

---

read_zarr_array	<i>Read a Zarr array</i>
-----------------	--------------------------

---

### 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 <code>paws.storage::s3()</code> . Only required for a file on S3. Leave as NULL for a file on local storage.

**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**

<code>zarr_path</code>	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.
<code>s3_client</code>	A list representing an S3 client. This should be produced by <code>paws.storage::s3()</code> .
<code>missing</code>	A character vector of length 1. This determines the behaviour when no file containing attributes is found. This can be one of: <ul style="list-style-type: none"> <li>"ignore" (the default): an empty list is returned silently</li> </ul>

- "warning": a warning is issued and an empty list is returned.
- "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, an empty list is returned.

### Examples

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

---

update_zarr_array	<i>Update (a subset of) an existing Zarr array</i>
-------------------	--

---

### 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.

### 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_zarray_array <- file.path(tempdir(), "new_array.zarr")
create_empty_zarr_array(
  zarr_array_path = new_zarray_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_zarray_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_zarray_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 = "F",
  compressor = use_zstd(),
  fill_value,
  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", "iS", "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 <a href="#">compressors</a> 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.

**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

<code>zarr_path</code>	A character vector of length 1. This provides the path to a Zarr array or group.
<code>new.zattrs</code>	a list inserted to <code>.zattrs</code> at the path.
<code>overwrite</code>	if TRUE (the default), existing <code>.zattrs</code> elements will be overwritten by <code>new.zattrs</code> .
<code>zarr_version</code>	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")
write_zarr_attributes(z1, list(date = "2025-01-01", author = "Jane Doe"))
```

---

ZarrArray

*Deprecated DelayedArray backend functions*

---

## Description

### [Superseded]

The DelayedArray backend has moved to a dedicated package: the ZarrArray package (<https://github.com/Bioconductor/ZarrArray>).

## Usage

```
ZarrArray(...)
```

```
writeZarrArray(...)
```

## Arguments

... Passed to the new function in the ZarrArray package.

---

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 <code>paws.storage::s3()</code> .
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")
dir.create(zarr_v2)
jsonlite::write_json(
  list("zarr_format" = 2L),
  file.path(zarr_v2, ".zgroup")
)
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 its shape and size. `zarr_overview()` can be used to get a quick overview of the array shape and contents, based on the `.zarr` (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

<code>zarr_array_path</code>	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.
<code>s3_client</code>	A list representing an S3 client. This should be produced by <code>paws.storage::s3()</code> .
<code>as_data_frame</code>	Logical determining whether the Zarr array details should be printed to screen (FALSE) or returned as a <code>data.frame</code> (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

## \* internal

- .chunk\_positions\_by\_chunk, 3
- .compress\_and\_write\_chunk, 4
- .configure\_codecs, 4
- .extract\_chunk, 5
- .get\_credentials, 5
- .normalize\_array\_path, 6
- .normalize\_v3\_metadata, 6
- .prepare\_write\_data, 7
- .read\_array\_metadata, 7
- .read\_consolidated\_metadata, 8
- .read\_json\_file, 8
- Rarr-package, 2
- read\_chunk, 12
- .chunk\_positions\_by\_chunk, 3
- .compress\_and\_write\_chunk, 4
- .configure\_codecs, 4
- .extract\_chunk, 5
- .get\_credentials, 5
- .normalize\_array\_path, 6
- .normalize\_v3\_metadata, 6
- .prepare\_write\_data, 7
- .read\_array\_metadata, 7
- .read\_array\_metadata(), 7
- .read\_consolidated\_metadata, 8
- .read\_json\_file, 8
- .read\_json\_file(), 6

compressors, 9, 11, 15  
create\_empty\_zarr\_array, 10

jsonlite::fromJSON(), 9  
jsonlite::read\_json(), 9

paws.storage::s3(), 7, 8, 12, 13, 18, 19

Rarr (Rarr-package), 2  
Rarr-package, 2  
read\_chunk, 12  
read\_zarr\_array, 12  
read\_zarr\_array(), 19  
read\_zarr\_attributes, 13

update\_zarr\_array, 14  
update\_zarr\_array(), 7, 11

use\_blosc (compressors), 9  
use\_bz2 (compressors), 9  
use\_gzip (compressors), 9  
use\_lz4 (compressors), 9  
use\_lzma (compressors), 9  
use\_zlib (compressors), 9  
use\_zstd (compressors), 9

write\_zarr\_array, 15  
write\_zarr\_array(), 7, 11  
write\_zarr\_attributes, 16  
writeZarrArray (ZarrArray), 17

zarr\_consolidate\_metadata, 18  
zarr\_overview, 19  
ZarrArray, 17