Package 'Rarr'

December 18, 2025

Title Read Zarr Files in R

Version 1.11.16

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 BiocGenerics, DelayedArray, R (>= 4.1.0)

Imports curl, jsonlite, methods, paws.storage, R.utils, utils

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

VignetteBuilder knitr

biocViews DataImport

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

SystemRequirements GNU make

Config/testthat/edition 3

git_url https://git.bioconductor.org/packages/Rarr

git_branch devel

git_last_commit ec950fd

git_last_commit_date 2025-12-12

Repository Bioconductor 3.23

Date/Publication 2025-12-17

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]

Maintainer Hugo Gruson <hugo.gruson@embl.de>

Contents

`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:

- Mike Smith (ORCID) (Maintainer from 2022 to 2025.) [conceptor]

Other contributors:

- Artür Manukyan [contributor]
- Sharla Gelfand [contributor]
- German Network for Bioinformatics Infrastructure - de.NBI [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

---

`.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, is_base64 = FALSE)
```

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.
`is_base64`	When dealing with Py_unicode strings we convert them to base64 strings for storage in our intermediate R arrays. This argument indicates if base64 is in use, because the conversion to raw in .as_raw should be done differently for base64 strings vs other types.

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

.create_replace_call *Create a string of the form* x[idx[[1]], idx[[2]]] <- y *for an array* x *where the number of dimensions is variable.*

Description

Create a string of the form x[idx[[1]], idx[[2]]] <- y for an array x where the number of dimensions is variable.

Usage

```
.create_replace_call(x_name, idx_name, idx_length, y_name)
```

Arguments

x_name	Name of the object to have items replaced
idx_name	Name of the list containing the indices
idx_length	Length of the list specified in idx_name
y_name	Name of the object containing the replacement items

Value

A character vector of length one containing the replacement commands. This is expected to be passed to `parse() |> eval()`.

.format_chunk *Format the decompressed chunk as an array of the correct type*

Description

When a chunk is decompressed it is returned as a vector of raw bytes. This function uses the array metadata to select how to convert the bytes into the final datatype and then converts the resulting output into an array of the appropriate dimensions, including re-ordering if the original data is in row-major order.

Usage

```
.format_chunk(decompressed_chunk, metadata, alt_chunk_dim)
```

Arguments

`decompressed_chunk`	Raw vector holding the decompressed bytes for this chunk.
`metadata`	List produced by `.read_array_metadata()` holding the contents of the `.zarray` file.
`alt_chunk_dim`	The dimensions of the array that should be created from this chunk. Normally this will be the same as the chunk shape in `metadata`, but when dealing with edge chunks, which may overlap the true extent of the array, the returned array should be smaller than the chunk shape.

Value

A list of length 2. The first element is the formatted chunk data. The second is an integer of length 1, indicating if warnings were encountered when converting types

If "chunk_data" is larger than the space remaining in destination array i.e. it contains the overflowing elements, these will be trimmed when the chunk is returned to `read_data()`

`.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.

.parse_datatype *Parse the data type encoding string*

Description

Parse the data type encoding string

Usage

```
.parse_datatype(typestr)
```

Arguments

typestr	The datatype encoding string. This is in the Numpy array typestr format.

Value

A list of length 4 containing the details of the data type.

.read_array_metadata *Read the .zarray metadata file associated with a Zarr array*

Description

Read the .zarray metadata file associated with a Zarr array

Usage

```
.read_array_metadata(zarr_path, metadata_file, 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.
metadata_file	One of ".zarray" (Zarr v2) or "zarr.json" (Zarr v3) specifying which metadata file to read.
s3_client	A list representing an S3 client. This should be produced by paws.storage::s3().

Value

A list containing the array metadata

.read_zmetadata *Read consolidated metadata file*

Description

Read consolidated metadata file

Usage

```
.read_zmetadata(zarr_path, s3_client)
```

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

.update_fill_value *Convert special fill values from strings to numbers*

Description

Special case fill values (NaN, Inf, -Inf) are encoded as strings in the Zarr metadata. R will create arrays of type character if these are defined and the chunk isn't present on disk. This function updates the fill value to be R's representation of these special values, so numeric arrays are created. A "null" fill value implies no missing values. We set this to NA as you can't create an array of type NULL in R. It should have no impact if there are really no missing values.

Usage

```
.update_fill_value(metadata, datatype)
```

Arguments

metadata	A list containing the array metadata. This should normally be generated by running read_json() on the .zarray file.
datatype	A list of details for the array datatype. Expected to be produced by .parse_datatype().

Value

Returns a list with the same structure as the input. The returned list will be identical to the input, unless the fill_value entry was on of: NULL, "NaN", "Infinity" or "-Infinity".

---

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 = "lz4")

use_zlib(level = 6L)

use_gzip(level = 6L)

use_bz2(level = 6L)

use_lzma(level = 9L)

use_lz4()

use_zstd(level = 3)
```

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'.
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_zlib(),
  fill_value,
  nchar = NULL,
  dimension_separator = "."
)
```

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 `zlib` 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 "/".

Value

If successful returns (invisibly) `TRUE`. However this function is primarily called for the size effect of initialising a Zarr array location and creating the `.zarray` 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"
)
```

get_decompressed_chunk_size *Determine the size of chunk in bytes after decompression*

Description

Determine the size of chunk in bytes after decompression

Usage

```
get_decompressed_chunk_size(datatype, dimensions)
```

Arguments

datatype	A list of details for the array datatype. Expected to be produced by `.parse_datatype()`.
dimensions	A list containing the dimensions of the chunk. Expected to be found in a list produced by `.read_array_metadata()`.

Value

An integer giving the size of the chunk in bytes

read_chunk *Read a single Zarr chunk*

Description

Read a single Zarr chunk

Usage

```
read_chunk(
  zarr_array_path,
  chunk_name,
  metadata,
  s3_client = NULL,
  alt_chunk_dim = NULL,
  fill = FALSE
)
```

Arguments

zarr_array_path	A character vector of length 1, giving the path to the Zarr array
chunk_name	The name of the chunk to read.
metadata	List produced by `.read_array_metadata()` holding the contents of the `.zarray` 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 `paws.storage::s3()`. Only required for a file on S3. Leave as `NULL` for a file on local storage.
alt_chunk_dim	The dimensions of the array that should be created from this chunk. Normally this will be the same as the chunk shape in `metadata`, but when dealing with edge chunks, which may overlap the true extent of the array the returned array should be smaller than the chunk shape.
fill	Logical of length 1. If `TRUE`, missing chunks will be filled with the fill value from the array metadata. If `FALSE` (the default), missing chunks will return `NULL`.

Value

A list of length 2. The entries should be names "chunk_data" and "warning". The first is an array containing the decompressed chunk values, the second is an integer indicating whether there were any overflow warnings generated will reading the chunk into an R datatype.

---

read_zarr_array *Read a Zarr array*

Description

Read a Zarr array

Usage

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

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.

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)
```

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 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.

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.

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 = "F",
  compressor = use_zlib(),
  fill_value,
  nchar,
  dimension_separator = "."
)
```

Arguments

`x`	The R array (or object that can be coerced to an 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 `zlib` 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 "/".

Value

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

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)
```

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.

Examples

```
z1 <- withr::local_tempdir(fileext = ".zarr")
write_zarr_attributes(z1, list(date = "2025-01-01", author = "Jane Doe"))
```

ZarrArray-class *ZarrArray constructor*

Description

ZarrayArray: function to create a new object of class ZarrArray.

Usage

```
ZarrArray(zarr_array_path)
```

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.

Value

Object of class ZarrArray

Examples

```
zarr_example <- system.file(
  "extdata", "zarr_examples", "column-first", "int32.zarr",
  package = "Rarr"
)
zarr_array <- ZarrArray(zarr_example)
is(zarr_array)
dim(zarr_array)
chunkdim(zarr_array)
```

ZarrRealizationSink *Write arrays to Zarr*

Description

Write array data to a Zarr backend via **DelayedArray**'s RealizationSink machinery.

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

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, 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