Package 'ZarrArray'

May 1, 2026

Title Bring Zarr datasets in R as DelayedArray objects

Description The ZarrArray package leverages the Rarr package to bring Zarr datasets in R as DelayedArray objects. The main class in the package is the ZarrArray class. A ZarrArray object is an array-like object that represents a Zarr dataset in R. ZarrArray objects are DelayedArray derivatives and therefore support all operations (delayed or block-processed) supported by DelayedArray objects.

biocViews Infrastructure, DataRepresentation, DataImport

URL https://bioconductor.org/packages/ZarrArray

BugReports https://github.com/Bioconductor/ZarrArray/issues

Version 1.1.0

License Artistic-2.0

Encoding UTF-8

Depends R (>= 3.4), methods, SparseArray, DelayedArray

Imports stats, tools, BiocGenerics, S4Vectors, IRanges, S4Arrays, Rarr (>= 1.11.33)

Suggests paws.storage, HDF5Array, testthat, knitr, rmarkdown, BiocStyle

VignetteBuilder knitr

Collate utils.R options.R ZarrArraySeed-class.R ZarrArray-class.R writeZarrArray-auto-args.R writeZarrArray.R zzz.R

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

git_branch devel

git_last_commit 498be8c

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-04-30

Author Hervé Pagès [aut, cre] (ORCID: <https://orcid.org/0009-0002-8272-4522>), Mike Smith [aut] (ORCID: <https://orcid.org/0000-0002-7800-3848>), Hugo Gruson [aut] (ORCID: <https://orcid.org/0000-0002-4094-1476>), Artür Manukyan [aut] (ORCID: <https://orcid.org/0000-0002-0441-9517>), Levi Waldron [fnd] (ORCID: <https://orcid.org/0000-0003-2725-0694>)

Maintainer Hervé Pagès <hpages.on.github@gmail.com>

Contents

`writeZarrArray` *Write an array-like object to disk in Zarr format*

Description

A function for writing an array-like object to disk in Zarr format.

Usage

```
writeZarrArray(x, zarr_path=NULL, chunkdim=NULL,
               nchar=NULL, zarr_version=3, verbose=NA)
```

Arguments

`x`	The array-like object to write to disk in Zarr format. If `x` is a DelayedArray object or derivative, then `writeZarrArray()` *realizes* it on disk, that is, all the delayed operations carried by the object are executed on the fly while the object is written to disk. See "On-disk realization of a DelayedArray object as a Zarr dataset" section below for more information.
`zarr_path`	`NULL` or the path (as a single string) to the directory where to write the data in Zarr format. If `NULL`, then `writeZarrArray()` will obtain this path by calling `get_writeZarrArray_auto_path()` internally. See `?writeZarrArray_auto_args` for more information.
`chunkdim`	The dimensions of the physical chunks to use when writing the data to disk. See `?writeZarrArray_auto_args` for how this is automatically determined when `chunkdim` is set to `NULL`.
`nchar`	When `x` is of type `character`, this argument specifies the maximum length of the stored strings. By default `writeZarrArray()` will use `max(nchar(x)) + 1`. You can override this by supplying your own value as a single positive integer but only if you know what you are doing.
`zarr_version`	The version of the Zarr specification to use. Currently, either `2` or `3`. The default is `3`.
`verbose`	Whether block processing progress should be displayed or not. If set to `NA` (the default), verbosity is controlled by `DelayedArray:::get_verbose_block_processing()`. Setting `verbose` to `TRUE` or `FALSE` overrides this.

Details

`writeZarrArray()` leverages lower-level functionality implemented in the **Rarr** package like `create_empty_zarr_arr` and `update_zarr_array()`.

Please note that, depending on the size of the data to write to disk and the performance of the disk, `writeZarrArray()` can take a long time to complete. Use `verbose=TRUE` to see its progress.

Value

A ZarrArray object that points to the newly written Zarr dataset on disk.

IMPORTANT NOTE: The dimnames on `x` are NOT propagated to the returned ZarrArray object at the moment! This is a temporary situation that will be addressed in future versions of the **ZarrArray** package.

On-disk realization of a DelayedArray object as a Zarr dataset

When passed a DelayedArray object, `writeZarrArray()` *realizes* it on disk, that is, all the delayed operations carried by the object are executed on the fly while the object is written to disk. This uses a block-processing strategy so that the full object is not realized at once in memory. Instead the object is processed block by block i.e. the blocks are realized in memory and written to disk one at a time.

In other words, `writeZarrArray(x, ...)` is semantically equivalent to `writeZarrArray(as.array(x), ...)`, except that `as.array(x)` is not called because this would realize the full object at once in memory.

See `?DelayedArray` for general information about DelayedArray objects.

See Also

- writeZarrArray_auto_args to control `writeZarrArray`'s *automatic argument values* like `zarr_path` and `chunkdim`.
- ZarrArray objects.

Examples

```
## ---------------------------------------------------------------------
## Write an ordinary matrix to disk in Zarr format
## ---------------------------------------------------------------------
m0 <- matrix(runif(180, min=-1), ncol=9)

path <- tempfile(fileext=".zarr")
M1 <- writeZarrArray(m0, path)
M1  # ZarrMatrix object

path(M1)
chunkdim(M1)

as(m0, "ZarrArray")  # equivalent to writeZarrArray(m0)

## ---------------------------------------------------------------------
## Transform a Zarr dataset and write it back in Zarr format
## ---------------------------------------------------------------------
M2 <- log(t(M1) + 1)  # DelayedMatrix object

M3 <- writeZarrArray(M2)
M3  # ZarrMatrix object

as(M2, "ZarrArray")  # equivalent to writeZarrArray(M2)

## ---------------------------------------------------------------------
## Use writeZarrArray() to convert an HDF5 dataset to Zarr format
## ---------------------------------------------------------------------
```

```
## The HDF5Array package includes an HDF5 file with some toy HDF5
## datasets:
library(HDF5Array)
h5_path <- system.file(package="HDF5Array", "extdata", "toy.h5")
h5ls(h5_path)

## Let's convert the M1 dataset to Zarr:
M1 <- HDF5Array(h5_path, "M1")
zarr_path <- file.path(tempdir(), "M1.zarr")
writeZarrArray(M1, zarr_path)

## Note that writeZarrArray() uses a block-processing strategy so that
## the original HDF5 dataset is not loaded at once in memory. Instead
## the object is loaded block by block and the blocks are written to
## disk one at a time. In other words writeZarrArray() can operate with
## a limited amount of memory regardless of the size of the original
## dataset. This amount of memory depends on the size of the blocks which
## can be controlled with setAutoBlockSize(). See '?setAutoBlockSize' in
## the DelayedArray package for more information.
```

`writeZarrArray-auto-args` *Control writeZarrArray's automatic argument values*

Description

`get_writeZarrArray_auto_path()` and `get_writeZarrArray_auto_chunkdim()` are used internally by `writeZarrArray()` and `ZarrRealizationSink()` to obtain *automatic values* for their `zarr_path` and `chunkdim` arguments when those arguments are not supplied.

Usage

```
## Used internally by writeZarrArray() to obtain "automatic values" for
## arguments 'zarr_path' and 'chunkdim':
get_writeZarrArray_auto_path()
get_writeZarrArray_auto_chunkdim(dim)

## Control the value returned by get_writeZarrArray_auto_path():
set_writeZarrArray_dump_dir(dir)

## Control the value returned by get_writeZarrArray_auto_chunkdim():
set_writeZarrArray_chunk_maxlen(maxlen=1000000L)
set_writeZarrArray_chunk_shape(shape="scale")

## The "get" functions that correspond to the "set" functions above:
get_writeZarrArray_dump_dir()
get_writeZarrArray_chunk_maxlen()
get_writeZarrArray_chunk_shape()
```

Arguments

`dim`	The dimensions (as an integer vector) of the array-like object to be realized to disk in Zarr format.

dir	The path (as a single string) to the "realization dump", that is, to the directory where realization of array-like objects in Zarr format should happen by default. If dir is missing, then the "realization dump" is set back to its default which is some directory under tempdir().
maxlen	The "maximum chunk length", that is, the maximum number of array elements per physical chunk when realizing an array-like object to disk in Zarr format.
shape	A string describing the shape of the physical chunks to use by default when realizing an array-like object to disk in Zarr format. See makeCappedVolumeBox in the **DelayedArray** package for the supported shapes.

Details

Here's how writeZarrArray() obtains its *automatic argument values*:

- The automatic value for zarr_path is obtained with get_writeZarrArray_auto_path().
- The automatic value for chunkdim is obtained with chunkdim(x) where x is the array-like object passed to writeZarrArray. If chunkdim(x) returns NULL (which can happen if x is an in-memory object or if the dimensions of its physical chunks cannot be determined), then chunkdim is obtained with get_writeZarrArray_auto_chunkdim(dim(x)).

The **ZarrArray** package provides a set of utility functions to control the values returned by get_writeZarrArray_auto_ and get_writeZarrArray_auto_chunkdim():

- The value returned by get_writeZarrArray_auto_path() is controlled by set_writeZarrArray_dump_dir().
- The value returned by get_writeZarrArray_auto_chunkdim() is controlled by set_writeZarrArray_chunk_ma and set_writeZarrArray_chunk_shape().

In other words, the set_writeZarrArray_*() utility functions provide some control over the behavior of writeZarrArray() and ZarrRealizationSink() when only their first argument is specified, like in:

```
a <- array(101:160, dim=5:3)
A <- writeZarrArray(a)
```

or in:

```
ZarrRealizationSink(dim(a))
```

Consequently, they also provide some control over the behavior of coercion of an arbitrary array-like object to ZarrArray (i.e. on as(a, "ZarrArray")), since this coercion simply calls writeZarrArray() on the supplied object.

Value

get_writeZarrArray_auto_path() returns a single string containing the automatic path used by writeZarrArray() when its zarr_path argument is not specified. Note that the function is used internally by writeZarrArray() and is not meant to be used directly by the user.

get_writeZarrArray_auto_chunkdim() returns an integer vector containing the automatic chunk dimensions used by writeZarrArray() when its chunkdim argument is not specified. Note that the function is used internally by writeZarrArray() and is not meant to be used directly by the user.

get_writeZarrArray_dump_dir() returns a single string containing the path to the "realization dump". set_writeZarrArray_dump_dir() returns an invisible single string containing the previous path to the "realization dump". In other words,

```
prev_dir <- set_writeZarrArray_dump_dir(dir)
```

is equivalent to

```
prev_dir <- get_writeZarrArray_dump_dir()
set_writeZarrArray_dump_dir(dir)
```

`get_writeZarrArray_chunk_maxlen()` returns the "maximum chunk length" (i.e. maximum number of array elements) of the physical chunks to use by default when realizing an array-like object to disk in Zarr format. `set_writeZarrArray_chunk_maxlen()` returns an invisible number that is the previous "maximum chunk length". In other words,

```
prev_maxlen <- set_writeZarrArray_chunk_maxlen(maxlen)
```

is equivalent to

```
prev_maxlen <- get_writeZarrArray_chunk_maxlen()
set_writeZarrArray_chunk_maxlen(maxlen)
```

`get_writeZarrArray_chunk_shape()` returns a single string describing the "chunk shape", that is, the shape of the physical chunks to use by default when realizing an array-like object to disk in Zarr format. `set_writeZarrArray_chunk_shape()` returns an invisible string describing the previous "chunk shape". In other words,

```
prev_shape <- set_writeZarrArray_chunk_shape(shape)
```

is equivalent to

```
prev_shape <- get_writeZarrArray_chunk_shape()
set_writeZarrArray_chunk_shape(shape)
```

See Also

- `writeZarrArray` for writing an array-like object to disk in Zarr format.
- ZarrArray objects.
- `makeCappedVolumeBox` in the **DelayedArray** package.

Examples

```
a <- array(101:160, dim=5:3)

get_writeZarrArray_dump_dir()  # default "Zarr realization dump"
A1 <- writeZarrArray(a)
path(A1)

## Take control of where writeZarrArray() should write Zarr datasets
## by default:
my_zarr_dump <- file.path(tempdir(), "my_zarr_dump")
set_writeZarrArray_dump_dir(my_zarr_dump)
A2 <- writeZarrArray(a)
path(A2)

m <- matrix(101:140, ncol=8)
M <- as(m, "ZarrArray")  # equivalent to writeZarrArray(m)
path(M)

## Set "Zarr realization dump" to the default:
set_writeZarrArray_dump_dir()
```

ZarrArray-class *Zarr datasets as DelayedArray objects*

Description

The ZarrArray class is a DelayedArray extension for representing and operating on a Zarr dataset.

All the operations available for DelayedArray objects work on ZarrArray objects.

Usage

```
## Constructor function:
ZarrArray(zarr_path, s3_client=NULL)
```

Arguments

zarr_path	The path (as a single string) to the Zarr dataset.
s3_client	Object created by paws.storage::s3(). Only required for a Zarr dataset on a non-public S3 bucket. Leave as NULL for a Zarr dataset on local storage or on a public S3 bucket.

Value

A ZarrArray (or ZarrMatrix) object. (Note that ZarrMatrix extends ZarrArray.)

See Also

- DelayedArray objects in the **DelayedArray** package.
- s3 in the **paws.storage** package for how to create a client for the S3 service.
- writeZarrArray for writing an array-like object to disk in Zarr format.
- The ZarrArraySeed helper class.

Examples

```
zarr_path <- system.file(package="Rarr", "extdata",
                         "zarr_examples", "column-first", "int32.zarr")
A <- ZarrArray(zarr_path)
A  # 3D ZarrArray object

path(A)
dim(A)
type(A)
chunkdim(A)

aperm(A)  # multidimensional transposition
chunkdim(aperm(A))

A[ , , 1]
log1p(t(A[ , , 1]))
rowSums(log1p(t(A[ , , 1])))

## Sanity check:
stopifnot(
```

```
  identical(dim(aperm(A)), rev(dim(A))),
  identical(chunkdim(aperm(A)), rev(chunkdim(A))),
  identical(rowSums(log1p(t(A[ , , 1]))),
            rowSums(log1p(t(as.array(A)[ , , 1]))))
)
```

ZarrArraySeed-class *ZarrArraySeed objects*

Description

ZarrArraySeed is a low-level helper class for representing a pointer to a Zarr dataset.

Note that a ZarrArraySeed object is not intended to be used directly. Most end users will typically create and manipulate a higher-level ZarrArray object instead. See ?ZarrArray for more information.

Usage

```
## --- Constructor function ---

ZarrArraySeed(zarr_path, s3_client=NULL)

## --- Accessors --------------

## S4 method for signature 'ZarrArraySeed'
path(object)

## S4 method for signature 'ZarrArraySeed'
dim(x)

## S4 method for signature 'ZarrArraySeed'
type(x)

## S4 method for signature 'ZarrArraySeed'
chunkdim(x)

## --- Data extraction --------

## S4 method for signature 'ZarrArraySeed'
extract_array(x, index)
```

Arguments

zarr_path, s3_client	See ?ZarrArray for a description of these arguments.
object, x	A ZarrArraySeed object.
index	See ?extract_array in the **S4Arrays** package.

Details

ZarrArraySeed objects only support a limited set of methods:

- `path()`: Returns the path to the Zarr dataset. Note that the `path()` generic is defined and documented in the **BiocGenerics** package.
- `dim()`, `type()`, `chunkdim()`. Note that the `type()` generic is defined and documented in the **BiocGenerics** package, and the `chunkdim()` generic is defined and documented in the **DelayedArray** package.
- `extract_array()`, `as.array()`, `is_sparse()`: Note that these generics are defined and documented in other packages e.g. in **S4Arrays** for `extract_array()` and `is_sparse()`, and in **base** for `as.array()`.

In order to access the full set of operations that are available for DelayedArray objects, one needs to wrap a ZarrArraySeed object in a DelayedArray object, typically by calling the `DelayedArray()` constructor on it.

Note that this is exactly what the `ZarrArray()` constructor function does.

The result of this wrapping is a ZarrArray object, a DelayedArray derivative that simply represents a ZarrArraySeed object wrapped in a DelayedArray object.

Value

`ZarrArraySeed()` returns a ZarrArraySeed object.

See Also

- ZarrArray objects.
- `type`, `extract_array`, and `is_sparse`, in the **S4Arrays** package.
- `chunkdim` in the **DelayedArray** package.

Examples

```
zarr_path <- system.file(package="Rarr", "extdata",
                         "zarr_examples", "column-first", "int32.zarr")
seed <- ZarrArraySeed(zarr_path)
seed  # ZarrArraySeed object

path(seed)
dim(seed)
type(seed)
chunkdim(seed)

DelayedArray(seed)  # ZarrArray object

## Sanity checks:
stopifnot(class(seed) == "ZarrArraySeed",
          class(DelayedArray(seed)) == "ZarrArray")
```

Index