Package: ZarrArray (via r-universe)

July 21, 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 1.1.3

License Artistic-2.0

Encoding UTF-8

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

Imports stats, tools, BiocGenerics, S4Vectors, IRanges, S4Arrays, Rarr (>= 2.1.9)

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

VignetteBuilder knitr

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

Config/pak/sysreqs libblosc-dev libxml2-dev libzstd-dev libssl-dev zlib1g-dev

Repository https://huber-group-embl.r-universe.dev

Date/Publication 2026-07-21 17:05:11 UTC

RemoteUrl https://github.com/Bioconductor/ZarrArray

RemoteRef HEAD

RemoteSha 092c417853c7cd2a8a171142e92c46a49b3a58c7

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_array() 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_pat`
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_maxle`
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")
```

ZarrSparseMatrix-class

HDF5 sparse matrices as DelayedMatrix objects

Description

The ZarrSparseMatrix class is a DelayedMatrix subclass for representing and operating on a Zarr-based sparse matrix stored in CSR/CSC/Yale format.

All the operations available for DelayedMatrix objects work on ZarrSparseMatrix objects.

Usage

```
## Constructor function:
ZarrSparseMatrix(zarr_store, group)
```

Arguments

zarr_store	The path (as a single string) to the Zarr store where the sparse matrix is located.
group	The name of the group node in the Zarr store where the sparse matrix is located.

Value

A ZarrSparseMatrix object.

See Also

- ZarrArray objects for representing conventional (a.k.a. dense) Zarr datasets as DelayedArray objects.
- ZarrADMatrix objects for representing a matrix from an AnnData-style Zarr store as a DelayedMatrix object.
- DelayedMatrix objects in the **DelayedArray** package.
- The ZarrSparseMatrixSeed helper class.
- SparseArray objects in the **SparseArray** package.

Examples

```
## ---------------------------------------------------------------------
## BASIC EXAMPLE
## ---------------------------------------------------------------------

## The anndataR package contains 3 AnnData-style ondisk data structures
## that contain the same data stored in different formats:
##   1. example.h5ad:        HDF5-based AnnData-style data structure;
##   2. example_v2.zarr.zip: Zarr-v2-based AnnData-style data structure;
##   3. example_v3.zarr.zip: Zarr-v3-based AnnData-style data structure.
##
## In each data structure the following groups contain sparse matrices:
##   group                      dim  type
##   /X                     100 x 50  double
##   /layers/counts         100 x 50  double
##   /layers/csc_counts     100 x 50  double
##   /obsp/connectivities    50 x 50  double
##   /obsp/distances         50 x 50  double
##   /uns/Sparse1D            6 x  1  integer

library(anndataR)

## Unzip example_v3.zarr.zip:
zad_basename <- "example_v3.zarr"  # zad: Zarr-based AnnData
zad_zip <- paste0(zad_basename, ".zip")
zad_zip_path <- system.file(package="anndataR", "extdata", zad_zip)
exdir <- tempdir()
unzip(zad_zip_path, exdir=exdir)
zad_store <- file.path(exdir, zad_basename)

## Represent /obsp/connectivities group as ZarrSparseMatrix object:
```

```
ZSM <- ZarrSparseMatrix(zad_store, "/obsp/connectivities")
ZSM

class(ZSM)  # ZarrSparseMatrix
is(ZSM, "DelayedMatrix")  # TRUE

seed(ZSM)
class(seed(ZSM))  # CSC_H5SparseMatrixSeed

dim(ZSM)
path(ZSM)
is_sparse(ZSM)  # TRUE

## Use coercion to load the full dataset into memory:
as.matrix(ZSM)          # as ordinary array (usually not recommended)
as(ZSM, "dgCMatrix")    # as dgCMatrix
as(ZSM, "SparseArray")  # as SparseArray object (most efficient)
SparseArray(ZSM)        # equivalent to 'as(ZSM, "SparseArray")'

## ---------------------------------------------------------------------
## ZarrSparseMatrix objects vs H5SparseMatrix objects
## ---------------------------------------------------------------------

## ZarrSparseMatrix and H5SparseMatrix objects are equivalent in terms
## of functionality. As a sanity check, we're going to compare the
## ZarrSparseMatrix objects obtained from example_v3.zarr with the
## corresponding H5SparseMatrix objects obtained from example.h5ad:

library(HDF5Array)
h5ad_path <- system.file(package="anndataR", "extdata", "example.h5ad")

groups <- c("/X", "/layers/counts", "/layers/csc_counts",
            "/obsp/connectivities", "/obsp/distances")
for (group in groups) {
    ZSM <- ZarrSparseMatrix(zad_store, group)
    HSM <- H5SparseMatrix(h5ad_path, group)
    stopifnot(identical(SparseArray(ZSM), SparseArray(HSM)))
}
```

`ZarrSparseMatrixSeed-class` *ZarrSparseMatrixSeed objects*

Description

ZarrSparseMatrixSeed is a low-level helper class for representing an on-disk sparse matrix stored in a Zarr store. The data in such matrix is typically represented as a group of three monodimensional Zarr datasets (data/indices/indptr) that altogether achieve the CSC or CSR layout commonly used to organize sparse matrix data.

ZarrSparseMatrixSeed is a virtual class with two concrete subclasses: CSC_ZarrSparseMatrixSeed for the *Compressed Sparse Column* layout, and CSR_ZarrSparseMatrixSeed for the *Compressed Sparse Row* layout.

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

Usage

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

ZarrSparseMatrixSeed(zarr_store, group, subdata=NULL,
                     dim=NULL, sparse.layout=NULL)

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

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

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

## S4 method for signature 'ZarrSparseMatrixSeed'
dimnames(x)

## S4 method for signature 'CSC_ZarrSparseMatrixSeed'
chunkdim(x)
## S4 method for signature 'CSR_ZarrSparseMatrixSeed'
chunkdim(x)

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

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

## S4 method for signature 'CSC_ZarrSparseMatrixSeed'
extract_sparse_array(x, index)
## S4 method for signature 'CSR_ZarrSparseMatrixSeed'
extract_sparse_array(x, index)

## --- Other methods ----------

## S4 method for signature 'ZarrSparseMatrixSeed'
is_sparse(x)

## S4 method for signature 'ZarrSparseMatrixSeed'
nzcount(x)
```

Arguments

`zarr_store`, `group`	See ?`ZarrSparseMatrix` for a description of these arguments.
`subdata`	Experimental. Don't use!
`dim`, `sparse.layout`	The `ZarrSparseMatrixSeed()` constructor should be able to automatically detect the dimensions and layout of the sparse matrix stored in the specified Zarr group, so the user shouldn't need to supply these arguments. See Details section below for some rare situations where the user might need to specify them.
`object`, `x`	A ZarrSparseMatrixSeed derivative.
`index`	See ?`extract_array` in the **S4Arrays** package.

Details

*** Layout in R vs physical layout ***

CSC_ZarrSparseMatrixSeed and CSR_ZarrSparseMatrixSeed objects transpose the matrix stored in the Zarr store when loading it into R. This means that a CSC_ZarrSparseMatrixSeed object represents a sparse matrix stored physically in the CSR layout (Compressed Sparse Row) at the Zarr level, and a CSR_ZarrSparseMatrixSeed object represents a sparse matrix stored physically in the CSC layout (Compressed Sparse Column) at the Zarr level.

*** Automatic detection of the dimensions and layout ***

The `ZarrSparseMatrixSeed()` constructor should be able to automatically detect the dimensions and layout of the sparse matrix stored in the specified Zarr group. However, in some rare situations, the user might want to bypass the detection mechanism, or they might be dealing with a sparse matrix stored in a Zarr group that doesn't provide this information (e.g. the group only contains the `data`, `indices`, and `indptr` components). In which case, they can supply the `dim` and `sparse.layout` arguments:

- `dim` must be an integer vector of length 2.
- `sparse.layout` must be `"CSC"` or `"CSR"`.

Note that both values must describe the dimensions and layout of the R object that will be returned, that is, *after* transposition from the physical layout used at the Zarr level. Also be aware that the supplied values will take precedence over whatever the Zarr group metadata says, which means that bad things will happen if they don't reflect the actual dimensions and layout of the sparse matrix. Use these arguments only if you know what you are doing!

*** ZarrSparseMatrixSeed object vs ZarrSparseMatrix object ***

Note that ZarrSparseMatrixSeed derivatives support a very limited set of methods:

- `path()`: Returns the path to the Zarr store that contains the sparse matrix.
- `dim()`, `dimnames()`.
- `extract_array()`, `is_sparse()`, `extract_sparse_array()`, `chunkdim()`: These generics are defined and documented in other packages e.g. in **S4Arrays** for `extract_array()` and `is_sparse()`, in **SparseArray** for `extract_sparse_array()`, and in **DelayedArray** for `chunkdim()`.

- `nzcount()`: Returns the number of nonzero values in the object.

In order to have access to the full set of operations that are available for DelayedMatrix objects, a ZarrSparseMatrixSeed derivative would first need to be wrapped in a DelayedMatrix object, typically by calling the `DelayedArray()` constructor on it.

Value

`ZarrSparseMatrixSeed()` returns a ZarrSparseMatrixSeed derivative (CSC_ZarrSparseMatrixSeed or CSR_ZarrSparseMatrixSeed object).

References

https://en.wikipedia.org/wiki/Sparse_matrix for a description of the CSR/CSC/Yale format (section "Compressed sparse row (CSR, CRS or Yale format)").

See Also

- ZarrSparseMatrix objects.

Examples

```
showClass("ZarrSparseMatrixSeed")
```

Index