

# Package ‘alabaster.matrix’

November 27, 2024

**Title** Load and Save Artifacts from File

**Version** 1.7.2

**Date** 2024-11-20

**License** MIT + file LICENSE

**Description**

Save matrices, arrays and similar objects into file artifacts, and load them back into memory. This is a more portable alternative to serialization of such objects into RDS files. Each artifact is associated with metadata for further interpretation; downstream applications can enrich this metadata with context-specific properties.

**Depends** alabaster.base

**Imports** methods, BiocGenerics, S4Vectors, DelayedArray (>= 0.31.8), S4Arrays, SparseArray (>= 1.5.22), rhdf5 (>= 2.47.1), HDF5Array, Matrix, Rcpp

**Suggests** testthat, knitr, BiocStyle, chihaya, BiocSingular, ResidualMatrix

**LinkingTo** Rcpp

**VignetteBuilder** knitr

**RoxygenNote** 7.3.2

**Encoding** UTF-8

**biocViews** DataImport, DataRepresentation

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

**git\_branch** devel

**git\_last\_commit** ed049db

**git\_last\_commit\_date** 2024-11-20

**Repository** Bioconductor 3.21

**Date/Publication** 2024-11-27

**Author** Aaron Lun [aut, cre]

**Maintainer** Aaron Lun <[infinite.monkeys.with.keyboards@gmail.com](mailto:infinite.monkeys.with.keyboards@gmail.com)>

## Contents

AmalgamatedArray . . . . .	2
createRawArraySeed . . . . .	3
DelayedMask . . . . .	4
preserveDelayedOperations . . . . .	5
readArray . . . . .	6
readDelayedArray . . . . .	7
readSparseMatrix . . . . .	8
recycleHdf5Files . . . . .	9
ReloadedArraySeed . . . . .	10
saveArray . . . . .	11
saveDelayedArray . . . . .	12
saveSparseMatrix . . . . .	14
storeDelayedObject . . . . .	15
WrapperArraySeed . . . . .	18
writeSparseMatrix . . . . .	19

<b>Index</b>	<b>22</b>
--------------	-----------

---

AmalgamatedArray	<i>Amalgamated array class</i>
------------------	--------------------------------

---

### Description

Implements an amalgamated array, equivalent to a delayed combination of DelayedArray objects. It allows `stageObject` to save a combination of multiple matrices without actually aggregating their data into a single file.

### Constructors

`AmalgamatedArraySeed(..., along=1)` accepts any number of named array-like objects and returns a `AmalgamatedArraySeed`. Each object corresponds to a block and should be named accordingly; names should be unique and non-empty. The `along` argument specifies the dimension in which matrices should be combined - for matrices, this is 1 for rows, 2 for columns.

`AmalgamatedArray(..., along=1)` accepts any number of named array-like objects and returns a `AmalgamatedArray`. Alternatively, a single `AmalgamatedArraySeed` may be provided in ...

### Functions

`componentNames(x)` will return a character vector of names of component arrays in a `AmalgamatedArray(Seed)` object `x`.

`extractComponents(x)` will return a named list of array-like objects, corresponding to the component arrays used to construct the `AmalgamatedArray(Seed)` object `x`.

`stageObject(x, dir, path, child = FALSE)` will save the `AmalgamatedArray` `x` and its components into the `path` inside `dir`. Each component array is staged into its own subdirectory inside `path`.

**Comments on usage**

The AmalgamatedArraySeed is closely related to (and in fact, is a subclass of) the [DelayedAbind](#) class. This means that we can leverage many of the **DelayedArray** methods for handling the delayed bind. In theory, we could just use a [DelayedAbind](#) directly and save it with **chihaya** in [stageObject](#) (via [preserveDelayedOperations\(TRUE\)](#)). However, this provides fewer opportunities for tracking and manipulating the samples. It also saves the per-sample matrices into a single file, which eliminates possibilities for per-file deduplication and linking, e.g., with [recycleHdf5Files\(TRUE\)](#).

**Author(s)**

Aaron Lun

**Examples**

```
first <- Matrix::rsparsematrix(10, 10, 0.1)
second <- Matrix::rsparsematrix(10, 20, 0.1)
mat <- AmalgamatedArray(list(foo = first, bar = second), along=2)
mat

componentNames(mat)
out <- extractComponents(mat)
lapply(out, dim)
```

---

createRawArraySeed     *Array loading utilities*

---

**Description**

Utilities for loading an array saved by [stageObject](#).

**Usage**

```
.createRawArraySeed(info, project, names = TRUE)

.extractArrayDimnames(path, group, ndim)
```

**Arguments**

info	A named list of metadata for this array.
project	Any argument accepted by the acquisition functions, see <a href="#">?acquireFile</a> . By default, this should be a string containing the path to a staging directory.
names	Logical scalar indicating whether the seed should be annotated with dimnames (if available).
path	String containing the path to the file containing said array.
group	String containing the name of the group with the dimnames.
ndim	Integer scalar specifying the number of dimensions.

**Details**

For `.createArraySeed`, the array should be one of:

- `hdf5_dense_array`
- `hdf5_sparse_matrix`
- `hdf5_delayed_array`
- `amalgamated_array`

For delayed arrays, the file may contain a seed array with the "custom alabaster local array" type. This should have a path dataset containing a relative path to another array in the same project, which is loaded and used as the seed for this delayed array. Callers can overwrite this behavior by setting "custom alabaster local array" in the `knownArrays` from **chihaya** before calling `.createRawArraySeed`.

For `.extractArrayDimnames`, `path` is expected to be a HDF5 file with a group specified by `group`. Each child of this group is a string dataset named after a (0-indexed) dimension, containing the names for that dimension.

**Value**

`.createRawArraySeed` returns a seed that can be used in the [DelayedArray](#) constructor.

`.extractArrayDimnames` returns a list of character vectors or NULL, containing the `dimnames`.

**Author(s)**

Aaron Lun

**Examples**

```
# Staging an array as an example:
dir <- tempfile()
dir.create(dir)
mat <- array(rpois(10000, 10), c(50, 20, 10))
meta <- stageObject(mat, dir, "whee")

# Loading it back as a DelayedArray seed:
.createRawArraySeed(meta, project=dir)
```

---

DelayedMask

*Delayed masking*

---

**Description**

Delayed masking of missing values, based on replacement of placeholder values with NA. This allows missingness to be encoded in frameworks without the same concept of NA as R.

## Usage

```
DelayedMask(x, placeholder)
```

## Arguments

`x` An existing **DelayedArray** seed.  
`placeholder` Placeholder value to replace with NA. This should be of the same type as `type(x)`.

## Details

If `is.na(placeholder)` is true for double-precision `x`, masking is performed for all values of `x` where `is.na` is true. This includes both NaNs and NAs; no attempt is made to distinguish between the NaN payloads.

Currently, an error is raised for any integer `x` that produces non-missing values of  $-2^{31}$  without a placeholder of `NA_integer_`. This is because R cannot distinguish the integer  $-2^{31}$  from an integer-type NA.

## Value

A `DelayedMask` object, to be wrapped in a `DelayedArray`.

## Author(s)

Aaron Lun

## Examples

```
original <- DelayedArray(matrix(rpois(40, lambda=2), ncol=5))
original
masked <- DelayedMask(original, 0)
DelayedArray(masked)
```

---

```
preserveDelayedOperations
```

*Preserve delayed operations during staging*

---

## Description

Preserve delayed operations via **chihaya** when staging a `DelayedArray` with `stageObject`.

## Usage

```
preserveDelayedOperations(preserve)
```

## Arguments

`preserve` Whether to preserve delayed operations using the **chihaya** specification.

**Details**

By default, any DelayedArray in `stageObject` will be saved as a new dense array or sparse matrix. However, if this option is enabled, DelayedArrays will instead be saved in the **chihaya** specification, where the delayed operations are themselves stored in the HDF5 file (see <https://artifacddb.github.io/chihaya/> for details).

The **chihaya** specification is more complicated to parse but can be helpful in reducing disk usage. One simple example is to avoid sparsity-breaking or integer-to-float operations by storing their delayed representations in the file. If the seed matrix is derived from some immutable reference location, advanced users can even store links to that location instead of duplicating the seed data.

**Value**

Logical scalar indicating whether delayed operations are to be preserved by the DelayedArray method. If `preserve` is supplied, it is used to set this scalar, and the *previous* value of the scalar is invisibly returned.

**Author(s)**

Aaron Lun

**Examples**

```
preserveDelayedOperations()
old <- preserveDelayedOperations(TRUE)
preserveDelayedOperations()
preserveDelayedOperations(old)
```

---

readArray	<i>Read a dense array from disk</i>
-----------	-------------------------------------

---

**Description**

Read a dense high-dimensional array from its on-disk representation. This is usually not directly called by users, but is instead called by dispatch in `readObject`.

**Usage**

```
readArray(path, metadata, ...)
```

**Arguments**

<code>path</code>	String containing a path to a directory, itself created by the <code>saveObject</code> method for a dense array.
<code>metadata</code>	Named list of metadata for this object, see <code>readObject</code> for more details.
<code>...</code>	Further arguments, ignored.

**Value**

A dense file-backed [ReloadedArray](#).

**Author(s)**

Aaron Lun

**See Also**

["saveObject,array-method"](#), to create the directory and its contents.

**Examples**

```
arr <- array(rpois(10000, 10), c(50, 20, 10))
dimnames(arr) <- list(
  paste0("GENE_", seq_len(nrow(arr))),
  letters[1:20],
  NULL
)

dir <- tempfile()
saveObject(arr, dir)
readObject(dir)
```

---

readDelayedArray	<i>Read a delayed array from disk</i>
------------------	---------------------------------------

---

**Description**

Read a delayed high-dimensional array from its on-disk representation. This is usually not directly called by users, but is instead called by dispatch in [readObject](#).

**Usage**

```
readDelayedArray(path, metadata, delayed_array.reload.args = list(), ...)
```

**Arguments**

path	String containing a path to a directory, itself created by the <a href="#">saveObject</a> method for a delayed array.
metadata	Named list of metadata for this object, see <a href="#">readObject</a> for more details.
delayed_array.reload.args	Named list of arguments to be passed to <a href="#">reloadDelayedObject</a> .
...	Further arguments, ignored.

**Value**

A multi-dimensional array-like object.

**Author(s)**

Aaron Lun

**See Also**

["saveObject,DelayedArray-method"](#), to create the directory and its contents.  
[reloadDelayedObject](#), for the methods to reload each delayed operation.

**Examples**

```
arr <- array(rpois(10000, 10), c(50, 20, 10))
dimnames(arr) <- list(
  paste0("GENE_", seq_len(nrow(arr))),
  letters[1:20],
  NULL
)

dir <- tempfile()
saveObject(arr, dir)
readObject(dir)
```

---

readSparseMatrix      *Read a sparse matrix from disk*

---

**Description**

Read a sparse matrix from its on-disk representation. This is usually not directly called by users, but is instead called by dispatch in [readObject](#).

**Usage**

```
readSparseMatrix(path, metadata, ...)
```

**Arguments**

path	String containing a path to a directory, itself created by the <a href="#">saveObject</a> method for a sparse matrix.
metadata	Named list of metadata for this object, see <a href="#">readObject</a> for more details.
...	Further arguments, ignored.

**Value**

A sparse [ReloadedMatrix](#) object.



**Author(s)**

Aaron Lun

**See Also**["saveObject,sparseMatrix-method"](#), to create the directory and its contents.**Examples**

```
mat <- Matrix::rsparsematrix(100, 200, density=0.2)
rownames(mat) <- paste0("GENE_", seq_len(nrow(mat)))
dir <- tempfile()
saveObject(mat, dir)
readObject(dir)
```

---

recycleHdf5Files	<i>Recycle existing HDF5 files</i>
------------------	------------------------------------

---

**Description**

Re-use existing files in HDF5-backed arrays rather than reserializing them in [stageObject](#).

**Usage**

```
recycleHdf5Files(recycle)
```

**Arguments**

recycle	Whether to recycle existing files for HDF5-backed DelayedArrays.
---------	--

**Details**

If this options is enabled, `stageObject` will attempt to link/copy existing files for any HDF5-backed DelayedArray instances - most specifically, [HDF5Array](#) objects and [H5SparseMatrix](#) objects using the 10X format. This avoids re-serialization of the data for faster staging. It also allows advanced users to add their own customizations into the HDF5 file during staging, as long as they do not interfere with [loadArray](#).

By default, this option is disabled as the properties of the existing file are not known in the general case. In particular, the file might contain other groups/datasets that are irrelevant, and use up extra disk space if copied; or confidential, and should not be stored in the staging directory. Users should only enable this option if they have full control over the generation and contents of the backing HDF5 files.

Also note that any dimnames on `x` will be ignored during recycling.

**Value**

Logical scalar indicating whether HDF5 files are to be reused. If `recycle` is supplied, it is used to set this scalar, and the *previous* value of the scalar is invisibly returned.

**Author(s)**

Aaron Lun

**Examples**

```
recycleHdf5Files()
old <- recycleHdf5Files(TRUE)
recycleHdf5Files()
recycleHdf5Files(old)
```

---

ReloadedArraySeed      *Reloaded **alabaster** array*

---

**Description**

An array that was reloaded from disk by the `readObject` function. This allows methods to refer to the existing on-disk representation by inspecting the path. For example, `saveObject` can just copy/link to the files instead of repeating the saving process.

**Usage**

```
ReloadedArraySeed(path, seed = NULL, ...)
```

```
ReloadedArray(path, seed = NULL, ...)
```

**Arguments**

<code>path</code>	String containing a path to the directory with the on-disk array representation. Alternatively an existing <code>ReloadedArraySeed</code> , which is returned without modification.
<code>seed</code>	Contents of the loaded array, e.g., as an ordinary R array, a <code>DelayedArray</code> or a sparse matrix. If <code>NULL</code> , this is obtained by calling <code>readObject</code> .
<code>...</code>	Further arguments to pass to <code>readObject</code> when <code>seed=NULL</code> .

**Details**

The `ReloadedArraySeed` is a subclass of the `WrapperArraySeed` and will just forward all operations to the underlying seed. Its main purpose is to track the path that was originally used to generate seed, which enables optimizations for methods that need to operate on the files.

One obvious optimization is the specialization of `saveObject` on `ReloadedArray` instances. Instead of loading the array data back into the R session and saving it again, the `saveObject` method can just

link or copy the existing files. This behavior is controlled by the optional `ReloadedArray.reuse.files` option in the `saveObject` method, which can be one of:

- "copy": copy the files from the original directory (as stored in the `ReloadedArray` object) to the new path specified in `saveObject`.
- "link": create a hard link from the files in the original directory to the new path. If this fails, we silently fall back to a copy. This mode is the default approach.
- "symlink": create a symbolic link from the files in the original directory to the new path. Each symbolic link refers to an absolute path in the original directory, which is useful when the contents of path might be moved (but the original directory will not).
- "relymlink": create a symbolic link from the files in the original directory to the new path. Each symbolic link refers to a relative path to its corresponding file in the original directory, which is useful when both path and the original directory are moved together, e.g., as they are part of the same parent object like a `SummarizedExperiment`.
- "none": ignore existing files and just save the contents by calling "`saveObject,DelayedArray-method`".

### Value

For the constructors, an instance of the [ReloadedArraySeed](#) or [ReloadedArray](#).

### Examples

```
arr <- array(rpois(10000, 10), c(50, 20, 10))
dir <- tempfile()
saveObject(arr, dir)
obj <- readArray(dir)
obj
DelayedArray::showtree(obj)
```

---

saveArray

*Save a multi-dimensional array to disk*

---

### Description

Save a high-dimensional array to its on-disk representations.

### Usage

```
## S4 method for signature 'array'
saveObject(x, path, ...)

## S4 method for signature 'denseMatrix'
saveObject(x, path, ...)
```

**Arguments**

- x                    An integer, numeric, logical or character array. Alternatively, any of the [dense-Matrix](#) subclasses from the **Matrix** package.
- path                String containing the path to a directory in which to save x.
- ...                 Further arguments, currently ignored.

**Value**

x is saved to path and NULL is invisibly returned.

**Author(s)**

Aaron Lun

**See Also**

[readArray](#), to read the directory contents back into the R session.

**Examples**

```
mat <- array(rpois(10000, 10), c(50, 20, 10))
dimnames(mat) <- list(
  paste0("GENE_", seq_len(nrow(mat))),
  letters[1:20],
  NULL
)

dir <- tempfile()
saveObject(mat, dir)
list.files(dir)
```

---

saveDelayedArray            *Save DelayedArrays to disk*

---

**Description**

Save [DelayedArray](#) objects to their on-disk representation.

**Usage**

```
## S4 method for signature 'DelayedArray'
saveObject(
  x,
  path,
  DelayedArray.dispatch.pristine = TRUE,
  DelayedArray.preserve.ops = FALSE,
  DelayedArray.store.args = list(),
```

```
    ...  
  )
```

### Arguments

`x` A [DelayedArray](#) object.

`path` String containing a path to a directory in which to save `x`.

`DelayedArray.dispatch.pristine` Logical scalar indicating whether to call the [saveObject](#) methods of seeds of pristine arrays.

`DelayedArray.preserve.ops` Logical scalar indicating whether delayed operations should be preserved on-disk.

`DelayedArray.store.args` Named arguments to pass to [storeDelayedObject](#).

`...` Further arguments, ignored.

### Value

`x` is saved to `path` and `NULL` is invisibly returned.

### Author(s)

Aaron Lun

### See Also

[storeDelayedObject](#), for the methods to save each delayed operation.

### Examples

```
mat <- Matrix::rsparsematrix(100, 200, density=0.2)  
rownames(mat) <- paste0("GENE_", seq_len(nrow(mat)))  
dmat <- DelayedArray::DelayedArray(mat) * 1  
  
dir <- tempfile()  
saveObject(dmat, dir, delayed.preserve.ops=TRUE)  
list.files(dir)
```

---

saveSparseMatrix      *Save a sparse matrix to disk*

---

### Description

Save a sparse matrix to its on-disk representations.

### Usage

```
## S4 method for signature 'sparseMatrix'  
saveObject(x, path, ...)
```

```
## S4 method for signature 'SVT_SparseMatrix'  
saveObject(x, path, ...)
```

### Arguments

x	A sparse matrix of some kind, typically from either the <b>Matrix</b> or <b>SparseArray</b> packages.
path	String containing the path to a directory in which to save x.
...	Further arguments, currently ignored.

### Value

x is saved to path and NULL is invisibly returned.

### Author(s)

Aaron Lun

### See Also

[readSparseMatrix](#), to read the directory contents back into the R session.

### Examples

```
mat <- Matrix::rsparsematrix(100, 200, density=0.2)  
rownames(mat) <- paste0("GENE_", seq_len(nrow(mat)))  
  
dir <- tempfile()  
saveObject(mat, dir)  
list.files(dir)
```

---

storeDelayedObject	<i>Store/reload a DelayedArray</i>
--------------------	------------------------------------

---

**Description**

Store or reload the delayed operations or array-like seeds of a [DelayedArray](#) in an existing HDF5 file.

**Usage**

```
storeDelayedObject(x, handle, name, ...)

reloadDelayedObject(handle, name, version = package_version("1.1"), ...)

reloadDelayedObjectFunctionRegistry(type = c("operation", "array"))

registerReloadDelayedObjectFunction(
  type = c("operation", "array"),
  subtype,
  fun,
  existing = c("old", "new", "error")
)

## S4 method for signature 'array'
storeDelayedObject(
  x,
  handle,
  name,
  version = package_version("1.1"),
  save.external.array = FALSE,
  ...
)

## S4 method for signature 'ANY'
storeDelayedObject(
  x,
  handle,
  name,
  version = package_version("1.1"),
  external.save.args = list(),
  external.dedup.session = NULL,
  external.dedup.action = c("link", "copy", "symlink", "relymlink"),
  ...
)

altStoreDelayedObjectFunction(store)
```

```

altStoreDelayedObject(...)

altReloadDelayedObjectFunction(reload)

altReloadDelayedObject(...)

createExternalSeedDedupSession()

```

### Arguments

x	Any of the delayed operation/array classes from <b>DelayedArray</b> .
handle	An <b>rhdf5</b> handle of a HDF5 file to save into (for storeDelayedObject) or load from (for reloadDelayedObject).
name	String containing the name of the group in file to save into (for storeDelayedObject) or load from (for reloadDelayedObject).
...	For storeDelayedObject and reloadDelayedObject, additional arguments to be passed to specific methods. For altStoreDelayedObject and altReloadDelayedObject, arguments to be passed to the alternative functions.
version	Package version of the <b>chihaya</b> format to use when storing or reloading delayed objects. When reloading, the version should be retrieved from the attributes of the outermost group, typically by readDelayedArray.
type	String specifying the type of delayed object, i.e., operation or array. This corresponds to delayed_type type in the <b>chihaya</b> attributes.
subtype	String specifying the subtype of the delayed object, This corresponds to delayed_array or delayed_operation type (depending on type) in the <b>chihaya</b> attributes.
fun	Function to reload a delayed object. This should accept the same arguments as reloadDelayedObject and should return a delayed array (if type="array") or operation (otherwise). It may also be NULL to delete an existing entry in the registry.
existing	Logical scalar indicating the action to take if a function has already been registered for type and subtype - keep the old or new function, or throw an error.
save.external.array	Logical scalar indicating whether to save an array-like seed as an external seed, even if a dedicated storeDelayedObject method is available.
external.save.args	Named list of further arguments to pass to <a href="#">altSaveObject</a> when saving an external seed.
external.dedup.session	Session object created by createExternalSeedDedupSession.
external.dedup.action	String specifying the deduplication method to use.
store	Function (typically a generic) to store delayed objects to file. This should accept the same arguments as storeDelayedObject.
reload	Function to reload delayed objects from file. This should accept the same arguments as reloadDelayedObject.



## Value

For `storeDelayedObject` and `altStoreDelayedObject`, the contents of `x` are saved to file, and `NULL` is invisibly returned.

For `reloadDelayedObject` and `altReloadDelayedObject`, a delayed operation or [DelayedArray](#) is returned.

For `altStoreDelayedObjectFunction`, the current store function is returned if `store` is missing. Otherwise, `store` is set as the current store function and the previous store function is returned.

For `altReloadDelayedObjectFunction`, the current reload function is returned if `reload` is missing. Otherwise, `reload` is set as the current reload function and the previous reload function is returned.

## Customization

Developers can easily extend **alabaster.matrix** to new delayed objects by writing new methods for `storeDelayedObject`. Methods should save the contents of the delayed object to the HDF5 file in the **chihaya** format. Each new store method typically requires a corresponding reloading function to be registered via `registerReloadDelayedObjectFunction`, so that `reloadDelayedObject` knows how to reconstitute the object from file.

Application developers can customize the process of storing/reloading delayed objects by specifying alternative functions in `altReloadDelayedObjectFunction` and `altStoreDelayedObjectFunction`. For example, if we want to preserve all delayed objects except for [DelayedSubset](#), we could replace `storeDelayedObject` with an `altStoreDelayedObject` that realizes any `DelayedSubset` instance into an ordinary matrix. This is analogous to the overrides for [altReadObject](#) and [altSaveObject](#).

Extension developers (i.e., those who write new methods for `storeDelayedObject` or new functions for `reloadDelayedObject`) should generally use `altStoreDelayedObject` and `altReloadDelayedObject` in their method/function bodies. This ensures that any custom overrides specified by application developers are still respected in the extensions to **alabaster.matrix**.

## External seeds

Whenever `storeDelayedObject` encounters a delayed operation or array-like seed for which it has no methods, the ANY method will save the delayed object as an “external seed”. The array is saved via [altSaveObject](#) into a `seeds` directory next to the file associated with `handle`. A reference to this external location is then stored in the `name` group inside `handle`.

Users can force this behavior for all array-like seeds by passing `save.external.array=TRUE` in the `...` arguments of `storeDelayedObject`. This instructs `storeDelayedObject` to save everything as external seeds, including those arrays for which it has methods. Doing so can be beneficial to enable deduplication, e.g., when two delayed arrays perform different operations on the same underlying seed. By saving the seeds externally, file management systems can identify the redundancy to save storage space.

Advanced users can explicitly deduplicate external seeds by setting `save.external.array=TRUE` and passing `external.dedup.session=` to `storeDelayedObject`. The `external.dedup.session` object is filled up with unique seeds as `storeDelayedObject` is called on various `DelayedArrays`. Whenever a duplicate seed is encountered, it is not saved again, but is instead linked or copied from the file path associated with the identical external seed. For example, a new session can be created when saving a `SummarizedExperiment` to deduplicate seeds across its assays.

The exact deduplication action can be specified by specifying the `external.dedup.action=` parameter. By default, `storeDelayedObject` attempts to create hard links, falling back to copies when a link cannot be created. Users can instead create copies, symbolic links to absolute paths, or even symbolic links to relative paths (e.g., to link to a “neighboring” assay of the same `SummarizedExperiment`).

When external seeds are encountered by `reloadDelayedObject`, they are loaded as `ReloadedArrays` (or some variant thereof) by `altReadObject`. Users can forcibly realize the reloaded seed into memory by passing `custom.takane.reload=TRUE` in `...` for the `reloadDelayedObject` call. This is occasionally helpful for providing a more faithful roundtrip from file back into memory.

### Author(s)

Aaron Lun

### See Also

[saveObject, DelayedArray-method](#) and [readDelayedArray](#), where these methods are used.  
<https://artifacddb.github.io/chihaya/>, for the file format specification of delayed objects.

### Examples

```
library(DelayedArray)
X <- DelayedArray(matrix(runif(100), ncol=20))
Y <- cbind(X, DelayedArray::ConstantArray(value=50, c(5, 10)))

library(rhdf5)
temp <- tempfile()
dir.create(temp)

fpath <- file.path(temp, "foo.h5")
fhandle <- H5Fcreate(fpath)
storeDelayedObject(Y@seed, fhandle, "YAY")
rhdf5::h5ls(fhandle)
H5Fclose(fhandle)

fhandle <- H5Fopen(fpath, "H5F_ACC_RDONLY")
reloadDelayedObject(fhandle, "YAY")
H5Fclose(fhandle)
```

---

WrapperArraySeed

*DelayedArray wrapper seed*

---

### Description

The `WrapperArraySeed` is, as the name suggests, a virtual class for a `DelayedArray` wrapper seed. This forwards most of the `DelayedArray` generic operations onto an internal seed class, typically

a [H5SparseMatrixSeed](#) or [HDF5ArraySeed](#) objects from [readSparseMatrix](#) or [readArray](#). Similarly, the `WrapperArray` is a virtual `DelayedArray` class that contains a `WrapperArraySeed`.

If an **alabaster** application operates on large arrays, developers may consider defining concrete subclasses of the `WrapperArraySeed` (and `WrapperArray`). These subclasses can store application-specific provenance-tracking information that persist throughout the lifetime of the array. Such information is most useful for optimizing [saveObject](#) calls, which can instruct the application to link to the existing array rather than creating a new file. Check out the [ReloadedArraySeed](#) class for an example of this approach.

## Examples

```
# Mocking up a concrete wrapper array class, which contains an
# extra 'foo_id' slot to track the provenance of the data.
setClass("FooArraySeed", contains="WrapperArraySeed",
        slots=c(seed="ANY", foo_id="character"))

y <- Matrix::rsparsematrix(1000, 100, 0.01)
foo <- new("FooArraySeed", seed=y, foo_id="F00.0001")

dim(foo)
is_sparse(foo)
extract_array(foo, list(1:10, 1:10))
extract_sparse_array(foo, list(1:10, 1:10))
```

---

writeSparseMatrix	<i>Write a sparse matrix</i>
-------------------	------------------------------

---

## Description

Writes a sparse matrix to file in a compressed sparse format.

## Usage

```
writeSparseMatrix(
  x,
  file,
  name,
  chunk = 10000,
  column = TRUE,
  tenx = FALSE,
  guess.integer = TRUE
)
```

## Arguments

`x` A sparse matrix of some sort. This includes sparse [DelayedMatrix](#) objects.

file	String containing a path to the HDF5 file. The file is created if it is not already present.
name	String containing the name of the group to store x.
chunk	Integer scalar specifying the chunk size for the indices and values.
column	Logical scalar indicating whether to store as compressed sparse column format.
tenx	Logical scalar indicating whether to use the 10X compressed sparse column format.
guess.integer	Logical scalar specifying whether to guess an appropriate integer type from x.

### Details

This writes a sparse matrix to file in various formats:

- column=TRUE and tenx=FALSE uses H5AD's `csr_matrix` format.
- column=FALSE and tenx=FALSE uses H5AD's `csc_matrix` format.
- tenx=TRUE uses 10X Genomics' HDF5 matrix format.

For the first two formats, the apparent transposition is deliberate, because columns in R are interpreted as rows in H5AD. This allows us to retain consistency the interpretation of samples (columns in R, rows in H5AD) and features (vice versa). Constructors for classes like [H5SparseMatrix](#) will automatically transpose so no extra work is required.

If `guess.integer=TRUE`, we attempt to save x's values into the smallest type that will accommodate all of its values. If x only contains unsigned integers, we will attempt to save either 8-, 16- or 32-bit unsigned integers. If x contains signed integers, we will fall back to 32-bit signed integers. For all other values, we will fall back to double-precision floating point values.

We attempt to save x's indices to unsigned 16-bit integers if the relevant dimension of x is small enough. Otherwise we will save it as an unsigned 32-bit integer.

### Value

A NULL invisibly. The contents of x are written to name in file.

### Author(s)

Aaron Lun

### Examples

```
library(Matrix)
x <- rsparsematrix(100, 20, 0.5)
tmp <- tempfile(fileext=".h5")
writeSparseMatrix(x, tmp, "csc_matrix")
writeSparseMatrix(x, tmp, "csr_matrix", column=FALSE)
writeSparseMatrix(x, tmp, "tenx_matrix", tenx = TRUE)

rhdf5::h5ls(tmp)
library(HDF5Array)
H5SparseMatrix(tmp, "csc_matrix")
```

```
H5SparseMatrix(tmp, "csr_matrix")  
H5SparseMatrix(tmp, "tenx_matrix")
```

# Index

- [.createRawArraySeed](#)  
([createRawArraySeed](#)), [3](#)
  - [.extractArrayDimnames](#)  
([createRawArraySeed](#)), [3](#)
- [acquireFile](#), [3](#)
- [altReadObject](#), [17](#), [18](#)
- [altReloadDelayedObject](#)  
([storeDelayedObject](#)), [15](#)
- [altReloadDelayedObjectFunction](#)  
([storeDelayedObject](#)), [15](#)
- [altSaveObject](#), [16](#), [17](#)
- [altStoreDelayedObject](#)  
([storeDelayedObject](#)), [15](#)
- [altStoreDelayedObjectFunction](#)  
([storeDelayedObject](#)), [15](#)
- [AmalgamatedArray](#), [2](#)
- [AmalgamatedArray-class](#)  
([AmalgamatedArray](#)), [2](#)
- [AmalgamatedArraySeed](#)  
([AmalgamatedArray](#)), [2](#)
- [AmalgamatedArraySeed-class](#)  
([AmalgamatedArray](#)), [2](#)
- [AmalgamatedMatrix-class](#)  
([AmalgamatedArray](#)), [2](#)
  
- [chunkdim](#), [DelayedMask](#)-method  
([DelayedMask](#)), [4](#)
- [chunkdim](#), [WrapperArraySeed](#)-method  
([WrapperArraySeed](#)), [18](#)
- [coerce](#), [AmalgamatedArray](#), [AmalgamatedMatrix](#)-method  
([AmalgamatedArray](#)), [2](#)
- [coerce](#), [AmalgamatedMatrix](#), [AmalgamatedArray](#)-method  
([AmalgamatedArray](#)), [2](#)
- [coerce](#), [ReloadedArray](#), [ReloadedMatrix](#)-method  
([ReloadedArraySeed](#)), [10](#)
- [coerce](#), [ReloadedMatrix](#), [ReloadedArray](#)-method  
([ReloadedArraySeed](#)), [10](#)
- [coerce](#), [WrapperArray](#), [dgCMatrix](#)-method  
([WrapperArraySeed](#)), [18](#)
  
- [coerce](#), [WrapperArraySeed](#), [dgCMatrix](#)-method  
([WrapperArraySeed](#)), [18](#)
- [componentNames](#) ([AmalgamatedArray](#)), [2](#)
- [createExternalSeedDedupSession](#)  
([storeDelayedObject](#)), [15](#)
- [createRawArraySeed](#), [3](#)
  
- [DelayedAbind](#), [3](#)
- [DelayedArray](#), [4](#), [5](#), [10](#), [12](#), [13](#), [15](#), [17](#)
- [DelayedArray](#), [AmalgamatedArraySeed](#)-method  
([AmalgamatedArray](#)), [2](#)
- [DelayedArray](#), [ReloadedArraySeed](#)-method  
([ReloadedArraySeed](#)), [10](#)
- [DelayedMask](#), [4](#)
- [DelayedMask-class](#) ([DelayedMask](#)), [4](#)
- [DelayedMatrix](#), [19](#)
- [DelayedSubset](#), [17](#)
- [denseMatrix](#), [12](#)
- [dim](#), [DelayedMask](#)-method ([DelayedMask](#)), [4](#)
- [dim](#), [WrapperArraySeed](#)-method  
([WrapperArraySeed](#)), [18](#)
- [dimnames](#), [DelayedMask](#)-method  
([DelayedMask](#)), [4](#)
- [dimnames](#), [WrapperArraySeed](#)-method  
([WrapperArraySeed](#)), [18](#)
  
- [extract\\_array](#), [DelayedMask](#)-method  
([DelayedMask](#)), [4](#)
- [extract\\_array](#), [WrapperArraySeed](#)-method  
([WrapperArraySeed](#)), [18](#)
- [extract\\_sparse\\_array](#), [DelayedMask](#)-method  
([DelayedMask](#)), [4](#)
- [extract\\_sparse\\_array](#), [WrapperArraySeed](#)-method  
([WrapperArraySeed](#)), [18](#)
- [extractComponents](#) ([AmalgamatedArray](#)), [2](#)
  
- [H5SparseMatrix](#), [9](#), [20](#)
- [H5SparseMatrixSeed](#), [19](#)
- [HDF5Array](#), [9](#)
- [HDF5ArraySeed](#), [19](#)

- is.na, [5](#)
- is\_sparse, DelayedMask-method  
(DelayedMask), [4](#)
- is\_sparse, WrapperArraySeed-method  
(WrapperArraySeed), [18](#)
- loadArray, [9](#)
- loadArray (readArray), [6](#)
- loadWrapperArray (WrapperArraySeed), [18](#)
- matrixClass, AmalgamatedArray-method  
(AmalgamatedArray), [2](#)
- matrixClass, ReloadedArray-method  
(ReloadedArraySeed), [10](#)
- path, DelayedMask-method (DelayedMask), [4](#)
- path, ReloadedArraySeed-method  
(ReloadedArraySeed), [10](#)
- path, WrapperArraySeed-method  
(WrapperArraySeed), [18](#)
- preserveDelayedOperations, [3, 5](#)
- readArray, [6, 12, 19](#)
- readDelayedArray, [7, 18](#)
- readObject, [6–8, 10](#)
- readSparseMatrix, [8, 14, 19](#)
- recycleHdf5Files, [3, 9](#)
- registerReloadDelayedObjectFunction  
(storeDelayedObject), [15](#)
- reloadDelayedObject, [7, 8](#)
- reloadDelayedObject  
(storeDelayedObject), [15](#)
- reloadDelayedObjectFunctionRegistry  
(storeDelayedObject), [15](#)
- ReloadedArray, [7, 11, 18](#)
- ReloadedArray (ReloadedArraySeed), [10](#)
- ReloadedArray-class  
(ReloadedArraySeed), [10](#)
- ReloadedArraySeed, [10, 11, 19](#)
- ReloadedArraySeed-class  
(ReloadedArraySeed), [10](#)
- ReloadedMatrix, [8](#)
- ReloadedMatrix-class  
(ReloadedArraySeed), [10](#)
- saveArray, [11](#)
- saveDelayedArray, [12](#)
- saveObject, [6–8, 10, 13, 19](#)
- saveObject, array-method (saveArray), [11](#)
- saveObject, DelayedArray-method  
(saveDelayedArray), [12](#)
- saveObject, denseMatrix-method  
(saveArray), [11](#)
- saveObject, ReloadedArray-method  
(ReloadedArraySeed), [10](#)
- saveObject, sparseMatrix-method  
(saveSparseMatrix), [14](#)
- saveObject, SVT\_SparseMatrix-method  
(saveSparseMatrix), [14](#)
- saveSparseMatrix, [14](#)
- stageObject, [2, 3, 5, 6, 9](#)
- stageObject, AmalgamatedArray-method  
(AmalgamatedArray), [2](#)
- stageObject, array-method (saveArray), [11](#)
- stageObject, DelayedArray-method  
(saveDelayedArray), [12](#)
- stageObject, DelayedMatrix-method  
(saveDelayedArray), [12](#)
- stageObject, Matrix-method (saveArray),  
[11](#)
- storeDelayedObject, [13, 15](#)
- storeDelayedObject, ANY-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, array-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, ConstantArraySeed-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, DelayedAbind-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, DelayedAperm-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, DelayedNaryIsoOp-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, DelayedSetDimnames-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, DelayedSubassign-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, DelayedSubset-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, DelayedUnaryIsoOpStack-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, DelayedUnaryIsoOpWithArgs-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, denseMatrix-method  
(storeDelayedObject), [15](#)
- storeDelayedObject, sparseMatrix-method  
(storeDelayedObject), [15](#)

storeDelayedObject, SVT\_SparseMatrix-method  
(storeDelayedObject), [15](#)

type, [5](#)

WrapperArray-class (WrapperArraySeed),  
[18](#)

WrapperArraySeed, [10](#), [18](#)

WrapperArraySeed-class  
(WrapperArraySeed), [18](#)

writeSparseMatrix, [19](#)