Package ‘alabaster.matrix’

April 3, 2025

Title Load and Save Artifacts from File

Version 1.7.8

Date 2025-03-04

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 (>= 0.33.3),
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 0b8582e

git_last_commit_date 2025-03-05

Repository Bioconductor 3.21

Date/Publication 2025-04-03

Author Aaron Lun [aut, cre]

Maintainer Aaron Lun <`infinite.monkeys.with.keyboards@gmail.com`>

Contents

`AmalgamatedArray` *Amalgamated array class*

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 ?`acquireFile`. 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://artifactdb.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` *Read a dense array from disk*

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

`path`	String containing a path to a directory, itself created by the `saveObject` method for a dense array.
`metadata`	Named list of metadata for this object, see `readObject` for more details.
`...`	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` *Read a delayed array from disk*

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 `saveObject` method for a delayed array.
`metadata`	Named list of metadata for this object, see `readObject` for more details.
`delayed_array.reload.args`	Named list of arguments to be passed to `reloadDelayedObject`.
`...`	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 `saveObject` method for a spars matrix.
`metadata`	Named list of metadata for this object, see `readObject` 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` *Recycle existing HDF5 files*

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

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

Details

The ReloadedArraySeed is a DelayedUnaryIsoOp subclass that 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 `ReloadedArray.reuse.files=` option in the `saveObject` method, which can be any of the choices for `action=` in `cloneDirectory`. It may also be `"none"` to 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,
  array.dedup.session = NULL,
  array.dedup.action = "link",
  array.character.vls = FALSE,
  ...
)

## S4 method for signature 'denseMatrix'
saveObject(
  x,
  path,
  array.dedup.session = NULL,
  array.dedup.action = "link",
  ...
)
```

Arguments

`x`	An integer, numeric, logical or character array. Alternatively, any of the denseMatrix subclasses from the **Matrix** package.
`path`	String containing the path to a directory in which to save `x`.
`array.dedup.session`	A session object created by `createDedupSession`, specifying which objects should be deduplicated if the same `x` is encountered multiple times.
`array.dedup.action`	String specifying how deduplication should occur, see options for the `action=` argument in `cloneDirectory`.
`array.character.vls`	Logical scalar indicating whether to save character arrays in the custom variable length string (VLS) array format. If `NULL`, this is determined based on a comparison of the expected storage against a fixed length array.
`...`	Further arguments, currently ignored.

Details

When `saveObject` is called multiple times on the same `x`, this method can avoid re-saving the array if a user supplies a deduplication session in `array.dedup.session=`. Instead, the method will link or copy (depending on the choice of `array.dedup.action=`) the files produced by the first `saveObject` call to the new `path=`. This saves time and reduces disk usage, and is particularly useful when saving complex data structures like a SummarizedExperiment. For example, if we have a SummarizedExperiment that contains multiple copies of the same array, we can pass a `array.dedup.session=` to the `saveObject` call on the SummarizedExperiment. This will instruct the internal `saveObject` calls to only write the array to disk once and subsequently make copies or links for all duplicates of that array. The same approach can be applied to deduplicate seeds in a DelayedArray, see `?"storeDelayedObject"` for details.

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.force.external = FALSE,
  DelayedArray.store.args = list(),
  array.dedup.session = NULL,
  array.dedup.action = NULL,
  ...
)
```

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.force.external`	Logical scalar indicating to save the seeds of `x` as external arrays. This is passed directly to `storeDelayedObject` as the `save.external.array=` argument, see `?storeDelayedObject` for details.
`DelayedArray.store.args`	More named arguments to pass to `storeDelayedObject`.

`array.dedup.session`, `array.dedup.action`	Arguments controlling deduplication of `x`, see `?"saveObject,array-method"` for details. If `x` is not a duplicate of an existing object, these arguments will be passed to further methods as described for `...`.
`...`	Further arguments passed to `storeDelayedObject` as `external.save.args`, if the delayed operations are to be preserved; otherwise, they are passed to `saveObject,array-method` or `saveObject,sparseMatrix-method`.

Details

Supplying `array.dedup.session=` by itself is only guaranteed to deduplicate `x` itself and may not deduplicate its seeds. Users should combine this with `DelayedArray.force.external=TRUE` to force seeds to be saved via `saveObject`, which exposes the seeds to the deduplication machinery in their respective `saveObject` methods. Check out `?"storeDelayedObject"` for more details.

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,
  array.dedup.session = NULL,
  array.dedup.action = "link",
  ...
)

## S4 method for signature 'SVT_SparseMatrix'
saveObject(
  x,
  path,
  array.dedup.session = NULL,
  array.dedup.action = "link",
  ...
)
```

Arguments

`x`	A sparse matrix of some kind, typically from either the **Matrix** or **SparseArray** packages.
`path`	String containing the path to a directory in which to save `x`.
`array.dedup.session, array.dedup.action`	Arguments controlling deduplication of `x`, see `?"saveObject,array-method"` for details.
`...`	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` *Store/reload a DelayedArray*

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 = NULL,
  ...
)
```

```
altStoreDelayedObjectFunction(store)

altStoreDelayedObject(...)

altReloadDelayedObjectFunction(reload)

altReloadDelayedObject(...)
```

Arguments

`x`	Any of the delayed operation/array classes from **DelayedArray**.
`handle`	An **rhdf5** 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 **chihaya** 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 **chihaya** attributes.
`subtype`	String specifying the subtype of the delayed object, This corresponds to `delayed_array` or `delayed_operation` type (depending on `type`) in the **chihaya** 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 `altSaveObject` when saving an external seed.
`external.dedup.session`	Deprecated, set `external.save.args$array.dedup.session` instead.
`external.dedup.action`	Deprecated, set `external.save.args$array.dedup.action` instead.
`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/funcion 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 specifying `save.external.array=TRUE`. 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 both `save.external.array=TRUE` and supplying `array.dedup.session=` in `external.save.args=`. The `array.dedup.session` object is filled up with unique seeds as `storeDelayedObject` is called on various DelayedArrays (see `?"saveObject,array-method"` for details). 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.

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://artifactdb.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

Deprecated, use DelayedUnaryIsoOp instead.

`writeSparseMatrix` *Write a sparse matrix*

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