Package: SeuratDisk (via r-universe)

May 23, 2026

Type Package

Title Interfaces for HDF5-Based Single Cell File Formats

Version 0.0.0.9021

Date 2023-03-16

Description The h5Seurat file format is specifically designed for the storage and analysis of multi-modal single-cell and spatially-resolved expression experiments, for example, from CITE-seq or 10X Visium technologies. It holds all molecular information and associated metadata, including (for example) nearest-neighbor graphs, dimensional reduction information, spatial coordinates and image data, and cluster labels. We also support rapid and on-disk conversion between h5Seurat and AnnData objects, with the goal of enhancing interoperability between Seurat and Scanpy.

URL `https://mojaveazure.github.io/seurat-disk/`, `https://github.com/mojaveazure/seurat-disk`

BugReports `https://github.com/mojaveazure/seurat-disk/issues`

License GPL-3 | file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Remotes satijalab/seurat-data

Depends R (>= 3.6.0)

Imports cli (>= 2.0.1), crayon (>= 1.3.4), hdf5r (>= 1.3.0), Matrix (>= 1.2.18), methods (>= 3.6.0), R6 (>= 2.4.1), rlang (>= 0.4.4), Seurat (>= 3.2.0), SeuratObject (>= 4.0.0), stats, stringi (>= 1.4.6), tools, utils, withr (>= 2.1.2)

Collate 'zzz.R' 'GetObject.R' 'AssembleObject.R' 'AppendData.R' 'TestObject.R' 'TestH5.R' 'h5info.R' 'scdisk.R' 'h5Seurat.R' 'Connect.R' 'PadMatrix.R' 'sparse_matrix.R' 'Transpose.R' 'Convert.R' 'LoadH5Seurat.R' 'loom.R' 'loom_bindings.R'

'LoadLoom.R' 'ReadH5.R' 'WriteH5Group.R' 'SaveH5Seurat.R' 'SaveLoom.R' 'h5Seurat_bindings.R'

Suggests httr, knitr, rmarkdown, SeuratData

VignetteBuilder knitr

Config/pak/sysreqs cmake libglpk-dev make libhdf5-dev libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev python3 zlib1g-dev

Repository https://blaserlab.r-universe.dev

Date/Publication 2023-11-04 15:33:03 UTC

RemoteUrl https://github.com/mojaveazure/seurat-disk

RemoteRef HEAD

RemoteSha 877d4e18ab38c686f5db54f8cd290274ccdbe295

Contents

`SeuratDisk-package` *SeuratDisk: Interfaces for HDF5-Based Single Cell File Formats*

Description

The h5Seurat file format is specifically designed for the storage and analysis of multi-modal single-cell and spatially-resolved expression experiments, for example, from CITE-seq or 10X Visium technologies. It holds all molecular information and associated metadata, including (for example) nearest-neighbor graphs, dimensional reduction information, spatial coordinates and image data, and cluster labels. We also support rapid and on-disk conversion between h5Seurat and AnnData objects, with the goal of enhancing interoperability between Seurat and Scanpy.

Package options

SeuratDisk uses the following options to control behavior, users can configure these with `options`:

`SeuratDisk.dtypes.logical_to_int` When writing logical vectors, coerce to integer types to ensure compatibility across languages (see `BoolToInt` for more details)

`SeuratDisk.dtypes.dataframe_as_group` When writing data.frames, always write out as a group regardless of factor presence

`SeuratDisk.chunking.MARGIN` Default direction for chunking datasets; choose from:

- **largest** Chunk along the largest dimension of a dataset
- **smallest** Chunk along the smallest dimension
- **first** Chunk along the first dimension
- **last** Chunk along the last dimension

`SeuratDisk.dimreducs.allglobal` Treat all DimReducs as global, regardless of actual global status

Author(s)

Maintainer: Paul Hoffman `<phoffman@nygenome.org>` (ORCID)

Other contributors:

- Rahul Satija `<rsatija@nygenome.org>` (ORCID) [contributor]

See Also

Useful links:

- `https://mojaveazure.github.io/seurat-disk/`
- `https://github.com/mojaveazure/seurat-disk`
- Report bugs at `https://github.com/mojaveazure/seurat-disk/issues`

`AppendData` *Append data from an h5Seurat file to a preexisting `Seurat` object*

Description

Append data from an h5Seurat file to a preexisting `Seurat` object

Usage

```
AppendData(
  file,
  object,
  assays = NULL,
  reductions = NULL,
  graphs = NULL,
  images = NULL,
  extras = "commands",
  overwrite = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'character'
AppendData(
  file,
  object,
  assays = NULL,
  reductions = NULL,
  graphs = NULL,
  images = NULL,
  extras = "commands",
  overwrite = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'H5File'
AppendData(
  file,
  object,
  assays = NULL,
  reductions = NULL,
  graphs = NULL,
  images = NULL,
  extras = "commands",
  overwrite = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'h5Seurat'
AppendData(
  file,
  object,
  assays = NULL,
  reductions = NULL,
```

```
  graphs = NULL,
  images = NULL,
  extras = "commands",
  overwrite = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`object`	A `Seurat` object to append data to
`assays`	One of: • A character vector with names of assays • A character vector with one or more of `counts`, `data`, `scale.data` describing which slots of **all assays** to load • A named list where each entry is either the name of an assay or a vector describing which slots (described above) to take from which assay • `NULL` for all assays • `FALSE` for no assays
`reductions`	One of: • A character vector with names of reductions • `NULL` for all reductions • `NA` for global reductions • `FALSE` for no reductions **Note**: Only reductions associated with an assay loaded in `assays` or marked as global will be loaded
`graphs`	One of: • A character vector with names of graphs • `NULL` for all graphs • `FALSE` for no graphs **Note**: Only graphs associated with an assay loaded in `assays` will be loaded
`images`	One of: • A character vector with names of images • `NULL` for all images • `NA` for global images • `FALSE` for no images
`extras`	Extra information to load; supports any combination of the following values: **“commands”** Load command logs. If `overwrite = TRUE`, replaces existing command logs
`overwrite`	Overwrite existing data in `object` with data from `file`
`verbose`	Show progress updates
`...`	Arguments passed to other methods

Value

`object` with the extra data requested

`Connect` *Connect to a single-cell HDF5 dataset*

Description

Connect to a single-cell HDF5 dataset

Usage

```
Connect(filename, type = NULL, mode = c("r", "r+"), force = FALSE)
```

Arguments

`filename`	Name of on-disk file
`type`	Type of single-cell dataset to connect as; choose from: • h5seurat Leave as `NULL` to guess type from file extension
`mode`	Mode to connect to data as; choose from: **r** Open existing dataset in read-only mode **r+** Open existing dataset in read/write mode
`force`	Force a connection if validation steps fail; returns a `H5File` object

Value

An object of class `type`, opened in mode `mode`

`Convert` *Convert an on-disk single-cell dataset to another format*

Description

HDF5-based single-cell datasets can be converted from one format to another using minimal memory. Details about conversion formats implemented are provided below

Usage

```
Convert(source, dest, assay, overwrite = FALSE, verbose = TRUE, ...)

## S3 method for class 'character'
Convert(source, dest, assay, overwrite = FALSE, verbose = TRUE, ...)

## S3 method for class 'H5File'
Convert(
  source,
  dest = "h5seurat",
  assay = "RNA",
  overwrite = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'h5Seurat'
Convert(
  source,
  dest = "h5ad",
  assay = DefaultAssay(object = source),
  overwrite = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`source`	Source dataset
`dest`	Name of destination dataset
`assay`	Converting from `h5Seurat`: name of assay to write out; converting to `h5Seurat`: name to store assay data as
`overwrite`	Overwrite existing `dest`
`verbose`	Show progress updates
`...`	Arguments passed to other methods

Value

If source is a `character`, invisibly returns `dest`; otherwise, returns an `H5File`, or filetype-specific subclass of `H5File` (eg. `h5Seurat`), connection to `dest`

AnnData/H5AD to h5Seurat

The AnnData/H5AD to h5Seurat conversion will try to automatically fill in datasets based on data presence. It works in the following manner:

Expression data: The expression matrices `counts`, `data`, and `scale.data` are filled by `/X` and `/raw/X` in the following manner:

- `counts` will be filled with `/raw/X` if present; otherwise, it will be filled with `/X`
- `data` will be filled with `/raw/X` if `/raw/X` is present and `/X` is dense; otherwise, it will be filled with `/X`
- `scale.data` will be filled with `/X` if it dense; otherwise, it will be empty

Feature names are taken from the feature-level metadata

Feature-level metadata: Feature-level metadata is added to the `meta.features` datasets in each assay. Feature names are taken from the dataset specified by the "_index" attribute, the "_index" dataset, or the "index" dataset, in that order. Metadata is populated with `/raw/var` if present, otherwise with `/var`; if both `/raw/var` and `/var` are present, then `meta.features` will be populated with `/raw/var` first, then `/var` will be added to it. For columns present in both `/raw/var` and `/var`, the values in `/var` will be used instead. **Note**: it is possible for `/var` to have fewer features than `/raw/var`; if this is the case, then only the features present in `/var` will be overwritten, with the metadata for features *not* present in `/var` remaining as they were in `/raw/var` or empty

Cell-level metadata: Cell-level metadata is added to `meta.data`; the row names of the metadata (as determined by the value of the "_index" attribute, the "_index" dataset, or the "index" dataset, in that order) are added to the "cell.names" dataset instead. If the "__categories" dataset is present, each dataset within "__categories" will be stored as a factor group. Cell-level metadata will be added as an HDF5 group unless factors are **not** present and `SeuratDisk.dtype.dataframe_as_group` is `FALSE`

Dimensional reduction information:: Cell embeddings are taken from `/obsm`; dimensional reductions are named based on their names from `obsm` by removing the preceding "X_".For example, if a dimensional reduction is named "X_pca" in `/obsm`, the resulting dimensional reduction information will be named "pca". The key will be set to one of the following:

- "PC_" if "pca" is present in the dimensional reduction name (`grepl("pca", reduction.name, ignore.case = TRUE)`)
- "tSNE_" if "tsne" is present in the dimensional reduction name (`grepl("tsne", reduction.name, ignore.case = TRUE)`)
- `reduction.name_` for all other reductions

Remember that the preceding "X_" will be removed from the reduction name before converting to a key. Feature loadings are taken from `/varm` and placed in the associated dimensional reduction. The dimensional reduction is determine from the loadings name in `/varm`:

- "PCs" will be added to a dimensional reduction named "pca"
- All other loadings in `/varm` will be added to a dimensional reduction named `tolower(loading)` (eg. a loading named "ICA" will be added to a dimensional reduction named "ica")

If a dimensional reduction cannot be found according to the rules above, the loading will not be taken from the AnnData/H5AD file. Miscellaneous information will be taken from `/uns/reduction` where `reduction` is the name of the reduction in `/obsm` without the preceding "X_"; if no dimensional reduction information present, then miscellaneous information will not be taken from the AnnData/H5AD file. Standard deviations are taken from a dataset `/uns/reduction/variance`; the variances will be converted to standard deviations and added to the `stdev` dataset of a dimensional reduction

Nearest-neighbor graph: If a nearest neighbor graph is present in `/uns/neighbors/distances`, it will be added as a graph dataset in the h5Seurat file and associated with `assay`; if a value is present in `/uns/neighbors/params/method`, the name of the graph will be `assay_method`, otherwise, it will be `assay_anndata`

Layers: TODO: add this

Miscellaneous information: All groups and datasets from `/uns` will be copied to `misc` in the h5Seurat file except for the following:

- Any group or dataset named the same as a dimensional reduction (eg. `/uns/pca`)
- `/uns/neighbors`

h5Seurat to AnnData/H5AD

The h5Seurat to AnnData/H5AD conversion will try to automatically fill in datasets based on data presence. Data presense is determined by the h5Seurat index (`source$index()`). It works in the following manner:

Assay data:

- `X` will be filled with `scale.data` if `scale.data` is present; otherwise, it will be filled with `data`
- `var` will be filled with `meta.features` **only** for the features present in `X`; for example, if `X` is filled with `scale.data`, then `var` will contain only features that have been scaled
- `raw.X` will be filled with `data` if `X` is filled with `scale.data`; otherwise, it will be filled with `counts`. If `counts` is not present, then `raw` will not be filled
- `raw.var` will be filled with `meta.features` with the features present in `raw.X`; if `raw.X` is not filled, then `raw.var` will not be filled

Cell-level metadata: Cell-level metadata is added to `obs`

Dimensional reduction information: Only dimensional reductions associated with `assay` or marked as global will be transfered to the H5AD file. For every reduction `reduc`:

- cell embeddings are placed in `obsm` and renamed to `X_reduc`
- feature loadings, if present, are placed in `varm` and renamed to either "PCs" if `reduc` is "pca" otherwise `reduc` in all caps

For example, if `reduc` is "ica", then cell embeddings will be "X_ica" in `obsm` and feature loaodings, if present, will be "ICA" in `varm`

Nearest-neighbor graphs: If a nearest-neighbor graph is associated with `assay`, it will be added to `uns/neighbors/distances`; if more than one graph is present, then **only** the last graph according to the index will be added.

Layers: Data from other assays can be added to `layers` if they have the same shape as `X` (same number of cells and features). To determine this, the shape of each alternate assays's `scale.data` and `data` slots are determined. If they are the same shape as `X`, then that slot (`scale.data` is given priority over `data`) will be added as a layer named the name of the assay (eg. "SCT"). In addition, the features names will be added to `var` as `assay_features` (eg. "SCT_features").

`h5Seurat-bindings` *Seurat bindings for h5Seurat files*

Description

Seurat bindings for h5Seurat files

Usage

```
## S3 method for class 'h5Seurat'
Cells(x)

## S3 method for class 'h5Seurat'
DefaultAssay(object, ...)

## S3 replacement method for class 'h5Seurat'
DefaultAssay(object, ...) <- value

## S3 method for class 'h5Seurat'
Idents(object, ...)

## S3 method for class 'H5Group'
IsGlobal(object)

## S3 method for class 'H5Group'
Key(object, ...)

## S3 method for class 'h5Seurat'
Project(object, ...)

## S3 replacement method for class 'h5Seurat'
Project(object, ...) <- value

## S3 method for class 'h5Seurat'
Stdev(object, reduction = "pca")
```

`h5Seurat-class` *A class for connections to h5Seurat files*

Description

A class for connections to h5Seurat files

A class for connections to h5Seurat files

Format

An `R6Class` object

Super classes

`hdf5r::H5RefClass` -> `hdf5r::H5File` -> `SeuratDisk::scdisk` -> `h5Seurat`

Methods

Public methods:

- `h5Seurat$index()`
- `h5Seurat$set.version()`
- `h5Seurat$version()`

Method `index()`**:** Get the index for this h5Seurat file

Usage:

```
h5Seurat$index()
```

Method `set.version()`**:** Set the version attribute

Usage:

```
h5Seurat$set.version(version)
```

Arguments:

`version`	A version number matching the regex `^\d+(\.\d+){2}(\.9\d{3})?$`

Method `version()`**:** Get the version attribute

Usage:

```
h5Seurat$version()
```

See Also

`H5File`

`LoadH5Seurat` *Load a saved* `Seurat` *object from an h5Seurat file*

Description

Load a saved `Seurat` object from an h5Seurat file

Usage

```
LoadH5Seurat(file, ...)

## S3 method for class 'character'
LoadH5Seurat(
  file,
  assays = NULL,
  reductions = NULL,
  graphs = NULL,
  neighbors = NULL,
  images = NULL,
  meta.data = TRUE,
  commands = TRUE,
  misc = is.null(x = assays),
  tools = is.null(x = assays),
  verbose = TRUE,
  ...
)

## S3 method for class 'H5File'
LoadH5Seurat(
  file,
  assays = NULL,
  reductions = NULL,
  graphs = NULL,
  neighbors = NULL,
  images = NULL,
  meta.data = TRUE,
  commands = TRUE,
  misc = is.null(x = assays),
  tools = is.null(x = assays),
  verbose = TRUE,
  ...
)

## S3 method for class 'h5Seurat'
LoadH5Seurat(
  file,
  assays = NULL,
  reductions = NULL,
  graphs = NULL,
  neighbors = NULL,
  images = NULL,
  meta.data = TRUE,
  commands = TRUE,
  misc = is.null(x = assays),
  tools = is.null(x = assays),
  verbose = TRUE,
```

```
  ...
)

## S3 method for class 'h5Seurat'
as.Seurat(
  x,
  assays = NULL,
  reductions = NULL,
  graphs = NULL,
  neighbors = NULL,
  images = NULL,
  meta.data = TRUE,
  commands = TRUE,
  misc = TRUE,
  tools = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`file`, `x`	Name of h5Seurat or connected h5Seurat file to load
`...`	Arguments passed to other methods
`assays`	One of: • A character vector with names of assays • A character vector with one or more of `counts`, `data`, `scale.data` describing which slots of **all assays** to load • A named list where each entry is either the name of an assay or a vector describing which slots (described above) to take from which assay • `NULL` for all assays
`reductions`	One of: • A character vector with names of reductions • `NULL` for all reductions • `NA` for global reductions • `FALSE` for no reductions **Note**: Only reductions associated with an assay loaded in `assays` or marked as global will be loaded
`graphs`	One of: • A character vector with names of graphs • `NULL` for all graphs • `FALSE` for no graphs **Note**: Only graphs associated with an assay loaded in `assays` will be loaded
`neighbors`	One of: • A character vector with the names of neighbors

	- `NULL` for all neighbors - `FALSE` for no neighbors
`images`	One of: - A character vector with names of images - `NULL` for all images - `NA` for global images - `FALSE` for no images
`meta.data`	Load object metadata
`commands`	Load command information **Note**: only commands associated with an assay loaded in `assays` will be loaded
`misc`	Load miscellaneous data
`tools`	Load tool-specific information
`verbose`	Show progress updates

Value

A `Seurat` object with the data requested

`LoadLoom` *Loom-file Loading*

Description

Load data from a loom file into a `Seurat` object

Usage

```
LoadLoom(
  file,
  assay = NULL,
  cells = "CellID",
  features = "Gene",
  normalized = NULL,
  scaled = NULL,
  filter = c("cells", "features", "all", "none"),
  verbose = TRUE,
  ...
)

## S3 method for class 'character'
LoadLoom(file, ...)

## S3 method for class 'H5File'
LoadLoom(file, ...)
```

```
## S3 method for class 'loom'
LoadLoom(file, ...)

## S3 method for class 'loom'
as.Seurat(
  x,
  assay = NULL,
  cells = "CellID",
  features = "Gene",
  normalized = NULL,
  scaled = NULL,
  filter = c("cells", "features", "all", "none"),
  verbose = TRUE,
  ...
)
```

Arguments

`file, x`	Name of loom file or a `loom` object to load data from
`assay`	Name of assay to store expression data as; if `NULL`, will search for an HDF5 attribute named `SEURAT_ASSAY` or an attribute dataset named `/attrs/SEURAT_ASSAY` for assay name. If not found, defaults to “RNA”
`cells`	Name of dataset in `/col_attrs` with cell names
`features`	Name of dataset in `/row_attrs` with feature names
`normalized`	Name of matrix in `/layers` to store normalized data as; pass “/matrix” to store `/matrix` as normalized data instead of raw counts
`scaled`	Name of dataset in `/layers` to store scaled data as
`filter`	Keep only selected cells and/or features as specified by `/col_attrs/Valid` and `/row_attrs/Valid`, respectively
`verbose`	Show progress updates
`...`	Arguments passed to other methods

Details

`LoadLoom` will try to automatically fill slots of a `Seurat` object based on data presence or absence in a given loom file. This method varies by loom specification version. For version-specific details, see sections below

Value

A `Seurat` object

Loom 0.1 Loading

Loading data from loom files less than version 3.0.0 is not currently supported

Loom 3.0.0 Loading

blah

See Also

Loom file conventions

`loom-bindings` *Seurat binding for loom files*

Description

Seurat binding for loom files

Usage

```
## S3 method for class 'loom'
DefaultAssay(object, ...)

## S3 method for class 'loom'
dim(x)
```

`loom-class` *A class for connections to loom files*

Description

A class for connections to loom files

A class for connections to loom files

Format

An `R6Class` object

Super classes

`hdf5r::H5RefClass` -> `hdf5r::H5File` -> `SeuratDisk::scdisk` -> `loom`

Methods

Public methods:

- `loom$add_attribute()`
- `loom$add_graph()`
- `loom$add_layer()`
- `loom$version()`
- `loom$timestamp()`
- `loom$last.modified()`

Method `add_attribute()`**:** Add an attribute

Usage:

```
loom$add_attribute(x, name, type = c("global", "row", "col"))
```

Arguments:

`x` Object to add as an attribute

`name` Name to store attribute as

`type` Type of attribute to add

Method `add_graph()`**:** Add a graph

Usage:

```
loom$add_graph(x, name, type = c("col", "row"), verbose = TRUE)
```

Arguments:

`x` ...

`name` ...

`type` ...

`verbose` ...

Method `add_layer()`**:** Add a layer to this loom file

Usage:

```
loom$add_layer(x, name, transpose = TRUE, verbose = TRUE)
```

Arguments:

`x` An object to save as a layer

`name` Name to store layer as

`transpose` ...

`verbose` ...

Returns: Invisibly returns `NULL`

Method `version()`**:** Get version information

Usage:

```
loom$version()
```

Returns: A `numeric_version` object with the loom specification version information

Method `timestamp()`**:** Add a timestamp to a dataset or group as an HDF5 attribute

Usage:

`loom$timestamp(name = NULL)`

Arguments:

`name` Name of dataset or group to add timestamp to; if `NULL`, timestamps the file as a whole

Returns: Invisibly returns the object

Method `last.modified()`**:** Retrieve a timestamp from a dataset or group

Usage:

`loom$last.modified(name = NULL, locale = FALSE)`

Arguments:

`name` Name of dataset or group to retrieve timestamp from; if `NULL`, retrieves timestamp from at the file-level

`locale` Change the timestamp of to the timezone of the locale

Returns: A character with the timestamp

See Also

`H5File`

ReadH5 *Load data from an HDF5 File*

Description

HDF5 allows storing data in an arbitrary fashion, which makes reading data into memory a hassle. The methods here serve as convenience functions for reading data stored in a certain format back into a certain R object. For details regarding how data should be stored on disk, please see the h5Seurat file specification.

Usage

```
## S3 method for class 'H5D'
as.array(x, ...)

## S3 method for class 'H5D'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'H5Group'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S4 method for signature 'H5Group'
as.factor(x)

## S4 method for signature 'H5Group'
as.list(x, which = NULL, ...)
```

```
## S3 method for class 'H5D'
as.logical(x, ...)

## S3 method for class 'H5D'
as.matrix(x, transpose = FALSE, ...)

## S3 method for class 'H5Group'
as.matrix(x, ...)

## S3 method for class 'H5D'
as.sparse(x, verbose = TRUE, ...)

## S3 method for class 'H5Group'
as.sparse(x, ...)

## S3 method for class 'H5D'
dimnames(x)
```

Arguments

`x`	An HDF5 dataset or group
`...`	Arguments passed to other methods
`row.names`	`NULL` or a character vector giving the row names for the data frame. Missing values are not allowed.
`optional`	logical. If `TRUE`, setting row names and converting column names (to syntactic names: see `make.names`) is optional. Note that all of R's **base** package `as.data.frame()` methods use `optional` only for column names treatment, basically with the meaning of `data.frame(*, check.names = !optional)`. See also the `make.names` argument of the `matrix` method.
`transpose`	Transpose the data upon reading it in, used when writing data in row-major order (eg. from C or Python)

Value

`as.array`: returns an `array` with the data from the HDF5 dataset

`as.data.frame`: returns a `data.frame` with the data from the HDF5 dataset or group

`as.factor`: returns a `factor` with the data from the HDF5 group

`as.list`: returns a `list` with the data from the HDF5 group

`as.logical`: returns a `logical` with the data from the HDF5 dataset

`as.matrix`, `H5D` method: returns a `matrix` with the data from the HDF5 dataset

`as.sparse`, `H5D` method: returns a sparse matrix with the data from the HDF5 dataset

`as.sparse`, `as.matrix`, `H5Group` method: returns a `sparseMatrix` with the data from the HDF5 group

`dimnames`: returns a two-length list of character vectors for row and column names. Row names should be in a column named `index`

`RegisterSCDisk` *Get and Register `scdisk` Subclasses*

Description

Mechanisms for registration of `scdisk` subclass generators for use in functions that rely on the class definition instead of an object.

Usage

```
GetSCDisk(r6class = NULL)

RegisterSCDisk(r6class)
```

Arguments

`r6class`	An R6 class generator or a character name of an R6 class generator

Details

While `scdisk`-subclassed objects (eg. `h5Seurat` objects) follow traditional inheritance patterns (can be determined through `inherits`), the class definitions and object generators do not. These functions provide a simple mechanism for adding and getting the defintions of `scdisk` subclasses for functions that utilize the object generators or other aspects of the class definition (such as `Convert`)

To register a subclass of `scdisk`, simply add a call to `RegisterSCDisk` in your load hook

```
.onLoad <- function(libname, pkgname) {
  RegisterSCDisk(classgen)
  # Other code to be run on load
}
```

Value

`GetSCDisk`: if `r6class` is `NULL`, then a vector of all registered `scdisk` subclasses; otherwise, a generator for the requested `scdisk` subclass

`RegisterSCDisk`: adds `r6class` to the internal subclass registry and invisibly returns `NULL`

Examples

```
GetSCDisk()
GetSCDisk("h5Seurat")

## Not run:
RegisterSCDisk(h5Seurat)

## End(Not run)
```

`SaveH5Seurat` *Save a* `Seurat` *object to an h5Seurat file*

Description

Save a `Seurat` object to an h5Seurat file

Usage

```
SaveH5Seurat(object, filename, overwrite = FALSE, verbose = TRUE, ...)

as.h5Seurat(x, ...)

## Default S3 method:
SaveH5Seurat(object, filename, overwrite = FALSE, verbose = TRUE, ...)

## S3 method for class 'Seurat'
SaveH5Seurat(
  object,
  filename = paste0(Project(object = object), ".h5Seurat"),
  overwrite = FALSE,
  verbose = TRUE,
  ...
)

## Default S3 method:
as.h5Seurat(x, filename, overwrite = FALSE, verbose = TRUE, ...)

## S3 method for class 'H5File'
as.h5Seurat(x, ...)

## S3 method for class 'Seurat'
as.h5Seurat(
  x,
  filename = paste0(Project(object = x), ".h5seurat"),
  overwrite = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`object, x`	An object
`filename`	Name of file to save the object to
`overwrite`	Overwrite `filename` if present
`verbose`	Show progress updates
`...`	Arguments passed to other methods

Value

`SaveH5Seurat`: Invisbly returns `filename`

`as.h5Seurat`: An `h5Seurat` object

`SaveLoom` *Save a `Seurat` object to a loom file*

Description

Save a `Seurat` object to a loom file

Usage

```
SaveLoom(object, filename, overwrite = FALSE, verbose = TRUE, ...)

as.loom(x, ...)

## Default S3 method:
SaveLoom(object, filename, overwrite = FALSE, verbose = TRUE, ...)

## S3 method for class 'Seurat'
SaveLoom(
  object,
  filename = paste0(Project(object = object), ".loom"),
  overwrite = FALSE,
  verbose = TRUE,
  ...
)

## Default S3 method:
as.loom(x, filename, overwrite = FALSE, verbose = TRUE)

## S3 method for class 'H5File'
as.loom(x, ...)

## S3 method for class 'Seurat'
as.loom(
  x,
  filename = paste0(Project(object = x), ".loom"),
  overwrite = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`object, x`	An object
`filename`	Name of file to save the object to
`overwrite`	Overwrite `filename` if present
`verbose`	Show progress updates
`...`	Arguments passed to other methods

Value

`SaveLoom`: Invisibly returns `filename`

`as.loom`: A `loom` object

`scdisk-class` *A disk-based object for single-cell analysis*

Description

A disk-based object for single-cell analysis

A disk-based object for single-cell analysis

Format

An `R6Class` object

Super classes

`hdf5r::H5RefClass` -> `hdf5r::H5File` -> `scdisk`

Methods

Public methods:

- `scdisk$new()`
- `scdisk$finalizer()`
- `scdisk$chunk.points()`
- `scdisk$timestamp()`
- `scdisk$last.modified()`

Method `new()`**:** Create a new `scdisk` object

Usage:

```
scdisk$new(
  filename = NULL,
  mode = c("a", "r", "r+", "w", "w-", "x"),
  validate = TRUE,
  ...
)
```

Arguments:

`filename` Name of on-disk file to connect to

`mode` How to open the file, choose from:

  - **a** Create new or open existing file, allow read and write
  - **r** Open existing file, allow read only
  - **r+** Open existing file, allow read and write
  - **w** Create new file (deleting any existing one), allow read and write
  - **w-, x** Create new file (error if exists), allow read and write

`validate` Validate the file upon connection

`...` Extra arguments passed to validation routine

Method `finalizer()`**:** Handle the loss of reference to this `scdisk` object

Usage:

```
scdisk$finalizer()
```

Method `chunk.points()`**:** Generate chunk points for a dataset

Usage:

```
scdisk$chunk.points(
  dataset,
  MARGIN = getOption(x = "SeuratDisk.chunking.MARGIN", default = "largest"),
  csize = NULL
)
```

Arguments:

`dataset` Name of dataset

`MARGIN` Direction to chunk in; defaults to largest dimension of dataset

`csize` Size of chunk; defaults to hdf5r-suggested chunk size

Returns: A matrix where each row is a chunk, column 1 is start points, column 2 is end points

Method `timestamp()`**:** Add a timestamp to a dataset or group as an HDF5 attribute

Usage:

```
scdisk$timestamp(
  name = NULL,
  attr = "ts",
  tz = "UTC",
  format = TSFormats(type = "R")
)
```

Arguments:

`name` Name of dataset or group to add timestamp to; if `NULL`, timestamps the file as a whole

`attr` Name of attribute to store timestamp ass

`tz, format` See `Timestamp`

Returns: Invisilby returns the object

Method `last.modified()`**:** Retrieve a timestamp from a dataset or group

Usage:

```
scdisk$last.modified(
  name = NULL,
  attr = "ts",
  locale = TRUE,
  tz = "UTC",
  format = TSFormats(type = "R")
)
```

Arguments:

`name` Name of dataset or group to retrieve timestamp from; if `NULL`, retrieves timestamp from at the file-level

`attr` Name of attribute to retrieve timestamp from

`locale` Change the timestamp of to the timezone of the locale

`tz`, `format` See `Timestamp`

Returns: A character with the timestamp

See Also

`H5File`

`Transpose` *Transpose a matrix*

Description

Transpose a matrix

Usage

```
Transpose(x, ...)

## S3 method for class 'dgCMatrix'
Transpose(x, ...)

## S3 method for class 'H5D'
Transpose(
  x,
  dest = GetParent(x = x),
  dname = paste0("t_", basename(path = x$get_obj_name())),
  overwrite = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'H5Group'
Transpose(
  x,
```

```
    dest = GetParent(x = x),
    dname = paste0("t_", basename(path = x$get_obj_name())),
    overwrite = FALSE,
    ...
  )
```

Arguments

`x`	A matrix to transpose
`...`	Arguments passed to other methods
`dest`	...
`dname`	...
`overwrite`	...
`verbose`	Show progress updates

Value

`dgCMatrix` method: returns a `dgCMatrix` with the data of `x` transposed

`H5D` and `H5Group` methods: Invisibly returns `NULL`

`WriteH5Group` *Write data to an HDF5 group*

Description

Writing data to HDF5 files can be done simply with usually sensible defaults. However, when wanting any semblance of control over how an R object is written out, the code constructs get complicated quickly. `WriteH5Group` provides a wrapper with sensible defaults over some of these complex code constructs to provide greater control over how data are written to disk. These defaults were chosen to fit best with h5Seurat files, see `vignette("h5Seurat-spec")` for more details

Usage

```
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'ANY'
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'array'
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'Assay'
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'data.frame'
```

```
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'dgCMatrix'
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'DimReduc'
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'factor'
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'Graph'
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'list'
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'logical'
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'Neighbor'
WriteH5Group(x, name, hgroup, verbose = TRUE)

## S4 method for signature 'SeuratCommand'
WriteH5Group(x, name, hgroup, verbose = TRUE)
```

Arguments

`x`	An object
`name`	Name to save data as
`hgroup`	An HDF5 file or group (`H5File` or `H5Group` objects from hdf5r)
`verbose`	Show progress updates

Value

Invisibly returns `NULL`

Examples

```
# Setup an HDF5 file
hfile <- hdf5r::H5File$new(filename = tempfile(fileext = '.h5'), mode = 'a')



# Data frames are stored as either datasets or groups, depending on the
# presence of factor columns
df <- data.frame(
  x = c('g1', 'g1', 'g2', 'g1', 'g2'),
  y = 1:5,
```

```
  stringsAsFactors = FALSE
)

# When no factor columns are present, the data frame is written as a single
# HDF5 compound dataset
WriteH5Group(x = df, name = 'df', hgroup = hfile)
hfile[['df']]

# When factors are present, the data frame is written as a group
# This is because h5py does not implement HDF5 Enums, so factor level
# information would be lost
df$x <- factor(x = df$x)
WriteH5Group(x = df, name = 'df.factor', hgroup = hfile)
hfile[['df.factor']]



# Factors turn into a group with two components: values and levels
# This is to preserve level information for HDF5 APIs that don't implement
# the HDF5 Enum type (eg. h5py)
# values corresponds to the integer values of each member of a factor
# levels is a string dataset with one entry per level
fctr <- factor(x = c('g1', 'g1', 'g2', 'g1', 'g2'))
WriteH5Group(x = fctr, name = 'factor', hgroup = hfile)
hfile[['factor']]



# Logicals get encoded as integers with the following mapping
# FALSE becomes 0L
# TRUE becomes 1L
# NA becomes 2L
# These are stored as H5T_INTEGERS instead of H5T_LOGICALS
# Additionally, an attribute called "s3class" is written with the value of "logical"
WriteH5Group(c(TRUE, FALSE, NA), name = "logicals", hgroup = hfile)
hfile[["logicals"]]
hfile[["logicals"]]$attr_open("s3class")$read()



# Close and remove the HDF5 file
hfile$close_all()
file.remove(hfile$filename)
```

Index