

Package: tiledbsoma (via r-universe)

September 22, 2026

Type Package

Title 'TileDB' Stack of Matrices, Annotated ('SOMA')

Description Interface for working with 'TileDB'-based Stack of Matrices, Annotated ('SOMA'): an open data model for representing annotated matrices, like those commonly used for single cell data analysis. It is documented at <https://github.com/single-cell-data>; a formal specification available is at https://github.com/single-cell-data/SOMA/blob/main/abstract_specification.md.

Version 2.3.0

URL <https://single-cell-data.github.io/TileDB-SOMA/>,
<https://github.com/single-cell-data/TileDB-SOMA>

BugReports <https://github.com/single-cell-data/TileDB-SOMA/issues>

License MIT + file LICENSE

Encoding UTF-8

Additional_repositories <https://tiledb-inc.r-universe.dev>,
<https://bnprks.r-universe.dev/>

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Depends R (>= 4.1.0)

Imports arrow (>= 19.0.1), bit64 (>= 4.6.0.1), data.table, fs, glue, lifecycle, Matrix (>= 1.6.4), methods, nanoarrow, R6, Rcpp, rlang (>= 1.1.5), stats, tiledb (>= 0.34.0), tools, utils

LinkingTo nanoarrow, Rcpp, RcppInt64

Suggests BPCells, datasets, iterators, itertools, jsonlite, knitr, pbmc3k, pbmc3k.tiledb, rmarkdown, S4Vectors, SeuratObject (>= 4.1.0), SingleCellExperiment (>= 1.20.0), SummarizedExperiment (>= 1.28.0), testthat (>= 3.0.0), tibble, withr

Config/testthat/edition 2

OS_type unix

SystemRequirements C++20, cmake, git

Config/pak/sysreqs cmake git make libuv1-dev libzstd-dev libssl-dev

Repository <https://chanzuckerberg.r-universe.dev>

Date/Publication 2026-01-27 15:20:09 UTC

RemoteUrl <https://github.com/single-cell-data/TileDB-SOMA>

RemoteRef 2.3.0

RemoteSha aac88138d12a5fb87ed2601500fdab103ba8a8a0

RemoteSubdir apis/r

Contents

ConfigList	3
example-datasets	4
get_default_context	5
IntIndexer	6
matrixZeroBasedView	7
PlatformConfig	9
set_default_context	11
set_log_level	12
show_package_versions	12
SOMAAxisQuery	13
SOMAAxisQueryResult	14
SOMACollection	15
SOMACollectionCreate	16
SOMACollectionOpen	17
SOMAContext	18
SOMADDataFrame	19
SOMADDataFrameCreate	24
SOMADDataFrameOpen	26
SOMADenseNDArray	27
SOMADenseNDArrayCreate	29
SOMADenseNDArrayOpen	30
SOMAEperiment	31
SOMAEperimentAxisQuery	34
SOMAEperimentCreate	41
SOMAEperimentOpen	42
SOMAMeasurement	43
SOMAMeasurementCreate	45
SOMAMeasurementOpen	46
SOMAOpen	48
SOMASparseNDArray	49
SOMASparseNDArrayCreate	51
SOMASparseNDArrayOpen	52

SOMATileDBContext	53
SparseReadIter	55
TableReadIter	57
tiledsoma_stats	58
write_soma	58
write_soma.Seurat	61
write_soma.SingleCellExperiment	65
write_soma.SummarizedExperiment	67
Index	69

ConfigList	<i>A Configuration List</i>
------------	-----------------------------

Description

An R6 mapping type for configuring various “parameters”. Essentially, serves as a nested map where the inner map is a [ScalarMap](#): {<param>: {<key>: <value>}}

Super class

[tiledsoma::MappingBase](#) -> ConfigList

Methods

Public methods:

- [ConfigList\\$get\(\)](#)
- [ConfigList\\$set\(\)](#)
- [ConfigList\\$setv\(\)](#)
- [ConfigList\\$clone\(\)](#)

Method `get()`:

Usage:

```
ConfigList$get(param, key = NULL, default = quote(expr = ))
```

Arguments:

param Outer key or “parameter” to fetch

key Inner key to fetch; pass NULL to return the [map](#) for param

default Default value to fetch if key is not found; defaults to NULL

Returns: The value of key for param in the map, or default if key is not found

Method `set()`:

Usage:

```
ConfigList$set(param, key, value)
```

Arguments:

param Outer key or “parameter” to set

key Inner key to set
value Value to add for key, or NULL to remove the entry for key; optionally provide only param and value as a [ScalarMap](#) to update param with the keys and values from value

Returns: `\[chainable\]` Invisibly returns self with value added for key in param

Method setv():

Usage:

`ConfigList$setv(...)`

Arguments:

... Ignored

Returns: Nothing; `setv()` is disabled for `ConfigList` objects

Method clone(): The objects of this class are cloneable with this method.

Usage:

`ConfigList$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
(cfg <- ConfigList$new())
cfg$set("op1", "a", 1L)
cfg
cfg$get("op1")
```

example-datasets

SOMA Example Datasets

Description

Access example SOMA objects bundled with the `tiledsoma` package.

Use `list_datasets()` to list the available datasets and `load_dataset()` to load a dataset into memory using the appropriate SOMA class. The `extract_dataset()` function returns the path to the extracted dataset without loading it into memory.

Usage

`list_datasets()`

`extract_dataset(name, dir = tempdir())`

`load_dataset(name, dir = tempdir(), tiledbsoma_ctx = NULL, context = NULL)`

Arguments

name	The name of the dataset.
dir	The directory where the dataset will be extracted to (default: <code>tempdir()</code>).
tiledb_soma_ctx	Optional (DEPRECATED) TileDB “Context” object that defaults to NULL.
context	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Details

The SOMA objects are stored as `tar.gz` files in the package’s `extdata` directory. Calling `load_dataset()` extracts the `tar.gz` file to the specified `dir`, inspects its metadata to determine the appropriate SOMA class to instantiate, and returns the SOMA object.

Value

`list_datasets()`: returns a character vector of the available datasets.
`extract_dataset()`: returns the path to the extracted dataset.
`load_dataset()`: returns a SOMA object.

Examples

```
list_datasets()

dir <- withr::local_tempfile(pattern = "pbmc-small")
dir.create(dir, recursive = TRUE)
dest <- extract_dataset("soma-exp-pbmc-small", dir)
list.files(dest)

dir <- withr::local_tempfile(pattern = "pbmc_small")
dir.create(dir, recursive = TRUE)
(exp <- load_dataset("soma-exp-pbmc-small", dir))
```

get_default_context	<i>Get the Default SOMA Context</i>
---------------------	-------------------------------------

Description

Retrieve the current default `SOMAContext` used by TileDB-SOMA operations.

Usage

```
get_default_context()
```

Details

This function returns the context that was either:

- Explicitly set via `set_default_context`, or
- Automatically created when a SOMA object was first created

An error is raised if no default context is set.

Value

The context that will be used for TileDB-SOMA API when no context is provided by the user.

IntIndexer

The SOMA Re-Indexer

Description

A re-indexer for unique integer indices

Methods

Public methods:

- `IntIndexer$new()`
- `IntIndexer$get_indexer()`
- `IntIndexer$clone()`

Method `new()`: Create a new re-indexer

Usage:

```
IntIndexer$new(data)
```

Arguments:

`data` Integer keys used to build the index (hash) table

Method `get_indexer()`: Get the underlying indices for the target data

Usage:

```
IntIndexer$get_indexer(target, nomatch_na = FALSE)
```

Arguments:

`target` Data to re-index

`nomatch_na` Set non-matches to NA instead of -1

Returns: A vector of 64-bit integers with target re-indexed

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
IntIndexer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
(keys <- c(-10000, -100000, 200000, 5, 1, 7))
(lookups <- unlist(replicate(n = 4L, c(-1L, 1:5), simplify = FALSE)))

indexer <- IntIndexer$new(keys)
indexer$get_indexer(lookups)
indexer$get_indexer(lookups, nomatch_na = TRUE)
```

matrixZeroBasedView *Zero-based Wrapper for Sparse Matrices*

Description

Zero-based Wrapper for Sparse Matrices

Zero-based Wrapper for Sparse Matrices

Details

matrixZeroBasedView is a wrapper shim for a matrix or `Matrix::sparseMatrix` that allows elemental matrix access using zero-based indices.

Methods**Public methods:**

- `matrixZeroBasedView$new()`
- `matrixZeroBasedView$take()`
- `matrixZeroBasedView$dim()`
- `matrixZeroBasedView$nrow()`
- `matrixZeroBasedView$ncol()`
- `matrixZeroBasedView$get_one_based_matrix()`
- `matrixZeroBasedView$sum()`
- `matrixZeroBasedView$print()`
- `matrixZeroBasedView$clone()`

Method `new()`: Initialize (lifecycle: maturing).

Usage:

```
matrixZeroBasedView$new(x)
```

Arguments:

x A matrix.

Method `take()`: Zero-based matrix element access.

Usage:

```
matrixZeroBasedView$take(i = NULL, j = NULL)
```

Arguments:

- i Row index (zero-based).
- j Column index (zero-based).

Returns: The specified matrix slice as another matrixZeroBasedView.

Method dim(): dim.

Usage:

matrixZeroBasedView\$dim()

Returns: The dimensions of the matrix.

Method nrow(): nrow.

Usage:

matrixZeroBasedView\$nrow()

Returns: Matrix row count.

Method ncol(): ncol.

Usage:

matrixZeroBasedView\$ncol()

Returns: Matrix column count.

Method get_one_based_matrix(): Get the one-based R matrix with its original class.

Usage:

matrixZeroBasedView\$get_one_based_matrix()

Returns: One-based matrix.

Method sum(): Perform arithmetic sum between this matrixZeroBasedView and another matrixZeroBasedView.

Usage:

matrixZeroBasedView\$sum(x)

Arguments:

- x the matrixZeroBasedView to sum.

Returns: The result of the sum as a matrixZeroBasedView.

Method print(): print.

Usage:

matrixZeroBasedView\$print()

Returns: Invisibly returns self.

Method clone(): The objects of this class are cloneable with this method.

Usage:

matrixZeroBasedView\$clone(deep = FALSE)

Arguments:

- deep Whether to make a deep clone.

Examples

```
(mat <- Matrix::rsparsematrix(3L, 3L, 0.3))
(mat0 <- matrixZeroBasedView$new(mat))

mat0$take(0, 0)
mat0$take(0, 0:2)$get_one_based_matrix()
```

PlatformConfig

*Platform Configuration***Description**

An R6 mapping type for configuring various “parameters” for multiple “platforms”, essentially serves a multi-nested map where the inner map is a [ScalarMap](#) contained within a [ConfigList](#) (middle map): {platform: {param: {key: value}}}

Super class

[tiledbsoma::MappingBase](#) -> PlatformConfig

Methods**Public methods:**

- [PlatformConfig\\$platforms\(\)](#)
- [PlatformConfig\\$params\(\)](#)
- [PlatformConfig\\$get\(\)](#)
- [PlatformConfig\\$get_params\(\)](#)
- [PlatformConfig\\$set\(\)](#)
- [PlatformConfig\\$setv\(\)](#)
- [PlatformConfig\\$clone\(\)](#)

Method platforms():

Usage:

`PlatformConfig$platforms()`

Returns: The names of the “platforms” (outer keys)

Method params():

Usage:

`PlatformConfig$params(platform = NULL)`

Arguments:

`platform` The “platform” to pull parameter names (middle keys) for; pass TRUE to return all possible parameter names

Returns: The parameter names (middle keys) for platform

Method get():*Usage:*

```
PlatformConfig$get(
  platform,
  param = NULL,
  key = NULL,
  default = quote(expr = )
)
```

Arguments:

platform The name of the “platform” (outer key) to fetch

param The name of the “parameters” of platform to fetch; if NULL, returns the [configuration](#) for platform

key The “key” (inner key) for param in platform to fetch; if NULL and param is passed, returns the [map](#) for param in platform

default Default value to fetch if key is not found; defaults to null

Returns: The value of key for param in platform in the map, or default if key is not found

Method get_params():*Usage:*

```
PlatformConfig$get_params(platform)
```

Arguments:

platform The name of the “platform” (outer key) to fetch

Returns: The [ConfigList](#) for platform

Method set():*Usage:*

```
PlatformConfig$set(platform, param, key, value)
```

Arguments:

platform The name of the “platform” (outer key) to set

param Name of the “parameter” (middle key) in platform to set

key Inner key to set

value Value to add for key, or NULL to remove the entry for key; optionally provide only platform, param, and value as a [ScalarMap](#) to update param for platform with the keys and values from value

Returns: `\[chainable\]` Invisibly returns self with value added for key in param for platform

Method setv():*Usage:*

```
PlatformConfig$setv(...)
```

Arguments:

... Ignored

Returns: Nothing; setv() is disabled for PlatformConfig objects

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PlatformConfig$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
(cfg <- PlatformConfig$new())
cfg$set("plat1", "op1", "a", 1L)
cfg

cfg$get("plat1")
cfg$get("plat1")$get("op1")
```

set_default_context	<i>Set the Default Global Context</i>
---------------------	---------------------------------------

Description

Configure a default [SOMAContext](#) to be used by all TileDB-SOMA operations when no explicit context is provided.

Usage

```
set_default_context(config = NULL, replace = FALSE)
```

Arguments

config	...
replace	Allow replacing the existing default context with new configuration parameters.

Details

This function should be called once at the beginning of your session before opening any SOMA objects if you want to customize the TileDB context parameters that will apply to all subsequent operations. Otherwise, a default context will be created automatically with standard parameters when you first open a SOMA object.

If the global context was already set, an error will be raised unless `replace=True`. Setting a new default context will not change the context for TileDB-SOMA objects that were already created.

Value

Invisibly, the default default context object.

set_log_level	<i>Set TileDB-SOMA Logging Level</i>
---------------	--------------------------------------

Description

Set the logging level for the R package and underlying C++ library

Usage

```
set_log_level(level)
```

Arguments

level	A character value with logging level. May be “trace”, “debug”, “info”, or “warn”
-------	--

Value

Invisibly returns NULL

show_package_versions	<i>Display package versions</i>
-----------------------	---------------------------------

Description

Print version information for **tiledb** (R package), libtiledbsoma, and TileDB embedded, suitable for assisting with bug reports.

Usage

```
show_package_versions()
```

Examples

```
show_package_versions()
```

SOMAAxisQuery

*SOMA Axis Query***Description**

Construct a single-axis query object with a combination of coordinates and/or value filters for use with [SOMAXperimentAxisQuery](#). (lifecycle: maturing)

Per dimension, the SOMAAxisQuery can have value of:

- None (i.e., coords = NULL and value_filter = NULL) - read all values
- Coordinates - a set of coordinates on the axis dataframe index, expressed in any type or format supported by [SOMADataFrame](#)'s read() method.
- A SOMA value_filter across columns in the axis dataframe, expressed as string
- Or, a combination of coordinates and value filter.

Public fields

coords The coordinates for the query.

value_filter The value filter for the query.

Methods**Public methods:**

- [SOMAAxisQuery\\$new\(\)](#)
- [SOMAAxisQuery\\$clone\(\)](#)

Method new(): Create a new SOMAAxisQuery object.

Usage:

```
SOMAAxisQuery$new(value_filter = NULL, coords = NULL)
```

Arguments:

value_filter Optional string containing a logical expression that is used to filter the returned values.

coords Optional indices specifying the rows to read: either a vector of the appropriate type or a named list of vectors corresponding to each dimension.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
SOMAAxisQuery$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

[tiledb::parse_query_condition\(\)](#) for more information about valid value filters.

SOMAAxisQueryResult	SOMAXperiment <i>Axis Query Result</i>
---------------------	--

Description

Access [SOMAXperimentAxisQuery](#) results.

Active bindings

obs [arrow::Table](#) containing obs query slice.

var [arrow::Table](#) containing var query slice. measurement_name.

X_layers named list of [arrow::Tables](#) for each X layer.

Methods

Public methods:

- [SOMAAxisQueryResult\\$new\(\)](#)
- [SOMAAxisQueryResult\\$clone\(\)](#)

Method [new\(\)](#): Create a new SOMAAxisQueryResult object.

Usage:

```
SOMAAxisQueryResult$new(obs, var, X_layers)
```

Arguments:

obs, var [arrow::Table](#) containing obs or var query slice.

X_layers named list of [arrow::Tables](#), one for each X layer.

Method [clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

```
SOMAAxisQueryResult$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

SOMACollection

*SOMA Collection***Description**

Contains a key-value mapping where the keys are string names and the values are any SOMA-defined foundational or composed type, including [SOMACollection](#), [SOMADDataFrame](#), [SOMADenseNDArray](#), [SOMASparseNDArray](#), or [SOMAExperiment](#) (lifecycle: maturing).

Adding new objects to a collection

The [SOMACollection](#) class provides a number of type-specific methods for adding new a object to the collection, such as `add_new_sparse_ndarray()` and `add_new_dataframe()`. These methods will create the new object and add it as member of the [SOMACollection](#). The new object will always inherit the parent context (see [SOMATileDBContext](#)) and, by default, its platform configuration (see [PlatformConfig](#)). However, the user can override the default platform configuration by passing a custom configuration to the `platform_config` argument.

Carrara (TileDB v3) behavior

When working with Carrara URIs (`tiledb://workspace/teamspace/...`), child objects created at a URI nested under a parent collection are **automatically added** as members of the parent. This means:

- You do not need to call `add_new_collection()` after creating a child at a nested URI—the child is already a member.
- For backward compatibility, calling `add_new_collection()` on an already-registered child is a **no-op** and will not cause an error.
- The member name must match the relative URI segment (e.g., creating at `parent_uri/child` automatically adds the child with key "child").

Super classes

```
tiledb::SOMAObject -> tiledbsoma::SOMACollectionBase -> SOMACollection
```

Methods**Public methods:**

- [SOMACollection\\$clone\(\)](#)

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
SOMACollection$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-collection")

(col <- SOMACollectionCreate(uri))
col$add_new_sparse_ndarray("sparse", arrow::float64(), shape = c(100L, 100L))
col$close()

(col <- SOMACollectionOpen(uri))
col$names()
```

SOMACollectionCreate *Create a SOMA Collection*

Description

Factory function to create a [SOMA collection](#) for writing (lifecycle: maturing).

Usage

```
SOMACollectionCreate(
  uri,
  ingest_mode = c("write", "resume"),
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  tiledb_timestamp = NULL,
  context = NULL
)
```

Arguments

uri	URI for the TileDB object.
ingest_mode	Ingestion mode when creating the TileDB object; choose from: “write”: create a new TileDB object and error if it already exists. “resume”: attempt to create a new TileDB object; if it already exists, simply open it for writing.
platform_config	Optional platform configuration.
tiledbsoma_ctx	Optional (DEPRECATED) SOMATileDBContext.
tiledb_timestamp	Optional Datetime (POSIXct) for TileDB timestamp.
context	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

A new [SOMA collection](#) stored at `uri` opened for writing.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-collection")

(col <- SOMACollectionCreate(uri))
col$add_new_sparse_ndarray("sparse", arrow::float64(), shape = c(100L, 100L))
col$close()

(col <- SOMACollectionOpen(uri))
col$names()
```

SOMACollectionOpen	<i>Open a SOMA Collection</i>
--------------------	-------------------------------

Description

Factory function to open a [SOMA collection](#) for reading (lifecycle: maturing).

Usage

```
SOMACollectionOpen(
  uri,
  mode = "READ",
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  tiledb_timestamp = NULL,
  context = NULL
)
```

Arguments

<code>uri</code>	URI for the TileDB object.
<code>mode</code>	One of "READ" or "WRITE".
<code>platform_config</code>	Optional platform configuration.
<code>tiledbsoma_ctx</code>	Optional (DEPRECATED) SOMATileDBContext.
<code>tiledb_timestamp</code>	Optional Datetime (POSIXct) for TileDB timestamp; defaults to the current time. If not NULL, all members accessed through the collection inherit the timestamp.
<code>context</code>	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

A [SOMA collection](#) stored at uri opened in mode mode.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-collection")

(col <- SOMACollectionCreate(uri))
col$add_new_sparse_ndarray("sparse", arrow::float64(), shape = c(100L, 100L))
col$close()

(col <- SOMACollectionOpen(uri))
col$names()
```

SOMAContext	<i>SOMA Context</i>
-------------	---------------------

Description

Context map for TileDB-SOMA objects

Active bindings

handle External pointer to the C++ interface

Methods

Public methods:

- [SOMAContext\\$new\(\)](#)
- [SOMAContext\\$get_config\(\)](#)
- [SOMAContext\\$get_data_protocol\(\)](#)
- [SOMAContext\\$is_tiledbv2\(\)](#)
- [SOMAContext\\$is_tiledbv3\(\)](#)
- [SOMAContext\\$clone\(\)](#)

Method new():

Usage:

```
SOMAContext$new(config = NULL)
```

Arguments:

config ...

Returns: An instantiated SOMATileDBContext object

Method get_config():

Usage:

SOMAContext\$get_config()

Returns: A character vector with the current config options set on the context.

Method get_data_protocol():

Usage:

SOMAContext\$get_data_protocol(uri)

Arguments:

uri A URI for a SOMA object

Returns: The data protocol to use for the URI.

Method is_tiledbv2():

Usage:

SOMAContext\$is_tiledbv2(uri)

Arguments:

uri A URI for a SOMA object

Returns: TRUE if the URI will use tiledbv2 semantics.

Method is_tiledbv3():

Usage:

SOMAContext\$is_tiledbv3(uri)

Arguments:

uri A URI for a SOMA object

Returns: TRUE if the URI will use tiledbv3 semantics.

Method clone(): The objects of this class are cloneable with this method.

Usage:

SOMAContext\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

SOMADataFrame

SOMADataFrame

Description

A SOMA data frame is a multi-column table that must contain a column called “soma_joinid” of type int64, which contains a unique value for each row and is intended to act as a join key for other objects, such as [SOMASparseNDArray](#) (lifecycle: maturing).

Super classes

[tiledbsoma::SOMAObject](#) -> [tiledbsoma::SOMAArrayBase](#) -> SOMADataFrame

Methods

Public methods:

- [SOMADataFrame\\$create\(\)](#)
- [SOMADataFrame\\$write\(\)](#)
- [SOMADataFrame\\$read\(\)](#)
- [SOMADataFrame\\$update\(\)](#)
- [SOMADataFrame\\$levels\(\)](#)
- [SOMADataFrame\\$shape\(\)](#)
- [SOMADataFrame\\$maxshape\(\)](#)
- [SOMADataFrame\\$domain\(\)](#)
- [SOMADataFrame\\$maxdomain\(\)](#)
- [SOMADataFrame\\$tiledsoma_has_upgraded_domain\(\)](#)
- [SOMADataFrame\\$tiledsoma_resize_soma_joinid_shape\(\)](#)
- [SOMADataFrame\\$tiledsoma_upgrade_domain\(\)](#)
- [SOMADataFrame\\$change_domain\(\)](#)
- [SOMADataFrame\\$clone\(\)](#)

Method [create\(\)](#): Create a SOMA data frame (lifecycle: maturing).

Note: [\\$create\(\)](#) is considered internal and should not be called directly; use factory functions (eg. [SOMADataFrameCreate\(\)](#)) instead.

Usage:

```
SOMADataFrame$create(
  schema,
  index_column_names = c("soma_joinid"),
  domain = NULL,
  platform_config = NULL
)
```

Arguments:

`schema` An [Arrow schema](#).

`index_column_names` A vector of column names to use as user-defined index columns. All named columns must exist in the schema, and at least one index column name is required.

`domain` An optional list specifying the domain of each index column. Each slot in the list must have its name being the name of an index column, and its value being a length-two vector consisting of the minimum and maximum values storable in the index column. For example, if there is a single int64-valued index column `soma_joinid`, then domain might be `list(soma_joinid=c(100, 200))` to indicate that values between 100 and 200, inclusive, can be stored in that column. If provided, this sequence must have the same length as `index_column_names`, and the index-column domain will be as specified. Omitting or setting the domain to `NULL` is deprecated. See also `change_domain` which allows you to expand the domain after create.

`platform_config` A [platform configuration](#) object

Returns: Returns self.

Method `write()`: Write values to the data frame (lifecycle: maturing).

Usage:

```
SOMADDataFrame$write(values)
```

Arguments:

`values` An [Arrow table](#) or [Arrow record batch](#) containing all columns, including any index columns. The schema for values must match the schema for the data frame.

Returns: Invisibly returns self.

Method `read()`: Read a user-defined subset of data, addressed by the data frame indexing column, and optionally filtered (lifecycle: maturing).

Usage:

```
SOMADDataFrame$read(
  coords = NULL,
  column_names = NULL,
  value_filter = NULL,
  result_order = "auto",
  log_level = "auto"
)
```

Arguments:

`coords` Optional named list of indices specifying the rows to read; each (named) list element corresponds to a dimension of the same name.

`column_names` Optional character vector of column names to return.

`value_filter` Optional string containing a logical expression that is used to filter the returned values. See [tiledb::parse_query_condition\(\)](#) for more information.

`result_order` Optional order of read results. This can be one of either "ROW_MAJOR", "COL_MAJOR", or "auto" (default).

`log_level` Optional logging level with default value of "warn".

Returns: An [Arrow table](#) or [TableReadIter](#)

Method `update()`: Update (lifecycle: maturing).

Usage:

```
SOMADDataFrame$update(values, row_index_name = NULL)
```

Arguments:

`values` A data frame, [Arrow table](#), or [Arrow record batch](#).

`row_index_name` An optional scalar character. If provided, and if the values argument is a data frame with row names, then the row names will be extracted and added as a new column to the data frame prior to performing the update. The name of this new column will be set to the value specified by `row_index_name`.

Details: Update the existing SOMADDataFrame to add or remove columns based on the input:

- columns present in the current the SOMADDataFrame but absent from the new values will be dropped.
- columns absent in current SOMADDataFrame but present in the new values will be added.

- any columns present in both will be left alone, with the exception that if values has a different type for the column, the entire update will fail because attribute types cannot be changed.

Furthermore, values must contain the same number of rows as the current SOMADDataFrame.

Returns: Invisibly returns NULL

Method `levels()`: Get the levels for an enumerated (factor) column.

Usage:

```
SOMADDataFrame$levels(column_names = NULL, simplify = TRUE)
```

Arguments:

`column_names` Optional character vector of column names to pull enumeration levels for; defaults to all enumerated columns.

`simplify` Simplify the result down to a vector or matrix.

Returns: If `simplify` returns one of the following:

- a vector of there is only one enumerated column.
- a matrix if there are multiple enumerated columns with the same number of levels.
- a named list if there are multiple enumerated columns with differing numbers of levels.

Otherwise, returns a named list.

Method `shape()`: Retrieve the shape; as SOMADDataFrames are shapeless, simply raises an error.

Usage:

```
SOMADDataFrame$shape()
```

Returns: None, instead a `.NotYetImplemented()` error is raised.

Method `maxshape()`: Retrieve the max shape; as SOMADDataFrames are shapeless, simply raises an error.

Usage:

```
SOMADDataFrame$maxshape()
```

Returns: None, instead a `.NotYetImplemented()` error is raised.

Method `domain()`: Returns a named list of minimum/maximum pairs, one per index column, currently storable on each index column of the data frame. These can be resized up to `maxdomain` (lifecycle: maturing).

Usage:

```
SOMADDataFrame$domain()
```

Returns: Named list of minimum/maximum values.

Method `maxdomain()`: Returns a named list of minimum/maximum pairs, one per index column, which are the limits up to which the data frame can have its domain resized (lifecycle: maturing).

Usage:

```
SOMADDataFrame$maxdomain()
```

Returns: Named list of minimum/maximum values.

Method `tiledb_soma_has_upgraded_domain()`: Test if the array has the upgraded resizable domain feature from TileDB-SOMA 1.15, the array was created with this support, or it has had `$upgrade_domain()` applied to it (lifecycle: maturing).

Usage:

```
SOMADataFrame$tiledb_soma_has_upgraded_domain()
```

Returns: Returns TRUE if the array has the upgraded resizable domain feature; otherwise, returns FALSE.

Method `tiledb_soma_resize_soma_joinid_shape()`: Increases the shape of the data frame on the `soma_joinid` index column, if it indeed is an index column, leaving all other index columns as-is. If the `soma_joinid` is not an index column, no change is made. This is a special case of `upgrade_domain()`, but simpler to keystroke, and handles the most common case for data frame domain expansion. Raises an error if the data frame doesn't already have a domain; in that case please call `$tiledb_soma_upgrade_domain()`.

Usage:

```
SOMADataFrame$tiledb_soma_resize_soma_joinid_shape(new_shape)
```

Arguments:

`new_shape` An integer, greater than or equal to 1 + the `soma_joinid` domain slot.

Returns: Invisibly returns NULL

Method `tiledb_soma_upgrade_domain()`: Allows you to set the domain of a SOMADataFrame, when the SOMADataFrame does not have a domain set yet. The argument must be a list of pairs of low/high values for the desired domain, one pair per index column. For string index columns, you must offer the low/high pair as `c("", "")`, or as NULL. If `check_only` is True, returns whether the operation would succeed if attempted, or a reason why it would not. The domain being requested must be contained within what `$maxdomain()` returns.

Usage:

```
SOMADataFrame$tiledb_soma_upgrade_domain(new_domain, check_only = FALSE)
```

Arguments:

`new_domain` A named list, keyed by index-column name, with values being two-element vectors containing the desired lower and upper bounds for the domain.

`check_only` If true, does not apply the operation, but only reports whether it would have succeeded.

Returns: If `check_only`, returns the empty string if no error is detected, else a description of the error. Otherwise, invisibly returns NULL

Method `change_domain()`: Allows you to set the domain of a SOMADataFrame, when the SOMADataFrame already has a domain set yet. The argument must be a list of pairs of low/high values for the desired domain, one pair per index column. For string index columns, you must offer the low/high pair as `c("", "")`, or as NULL. If `check_only` is True, returns whether the operation would succeed if attempted, or a reason why it would not. The return value from domain must be contained within the requested `new_domain`, and the requested `new_domain` must be contained within the return value from `$maxdomain()` (lifecycle: maturing).

Usage:

```
SOMADataFrame$change_domain(new_domain, check_only = FALSE)
```

Arguments:

`new_domain` A named list, keyed by index-column name, with values being two-element vectors containing the desired lower and upper bounds for the domain.

`check_only` If true, does not apply the operation, but only reports whether it would have succeeded.

Returns: If `check_only`, returns the empty string if no error is detected, else a description of the error. Otherwise, invisibly returns NULL

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
SOMADataFrame$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-data-frame")
df <- data.frame(
  soma_joinid = bit64::seq.integer64(0L, 99L),
  group = sample(factor(c("g1", "g2")), size = 100L, replace = TRUE),
  nCount = stats::rbinom(100L, 10L, 0.3)
)
(sch <- arrow::infer_schema(df))
(sdf <- SOMADataFrameCreate(uri, sch, domain = list(soma_joinid = c(0, 100))))
sdf$write(arrow::as_arrow_table(df, schema = sch))
sdf$close()

(sdf <- SOMADataFrameOpen(uri))
head(as.data.frame(sdf$read())$concat())
```

SOMADataFrameCreate *Create a SOMA Data Frame*

Description

Factory function to create a [SOMA data frame](#) for writing (lifecycle: maturing).

Usage

```
SOMADataFrameCreate(
  uri,
  schema,
  index_column_names = c("soma_joinid"),
  domain = NULL,
  ingest_mode = c("write", "resume"),
```

```

platform_config = NULL,
tiledb_soma_ctx = NULL,
tiledb_timestamp = NULL,
context = NULL
)

```

Arguments

<code>uri</code>	URI for the TileDB object.
<code>schema</code>	Arrow schema argument for the SOMA dataframe .
<code>index_column_names</code>	A vector of column names to use as user-defined index columns; all named columns must exist in the schema, and at least one index column name is required.
<code>domain</code>	An optional list of 2-element vectors specifying the domain of each index column. Each vector should be a pair consisting of the minimum and maximum values storable in the index column. For example, if there is a single int64-valued index column, then domain might be <code>c(100, 200)</code> to indicate that values between 100 and 200, inclusive, can be stored in that column. If provided, this list must have the same length as <code>index_column_names</code> , and the index-column domain will be as specified. If omitted entirely, or if NULL in a given dimension, the corresponding index-column domain will use the minimum and maximum possible values for the column's datatype. This makes a SOMA data frame growable.
<code>ingest_mode</code>	Ingestion mode when creating the TileDB object; choose from: “write”: create a new TileDB object and error if it already exists. “resume”: attempt to create a new TileDB object; if it already exists, simply open it for writing.
<code>platform_config</code>	Optional platform configuration.
<code>tiledb_soma_ctx</code>	Optional (DEPRECATED) <code>SOMATileDBContext</code> .
<code>tiledb_timestamp</code>	Optional Datetime (POSIXct) for TileDB timestamp.
<code>context</code>	Optional <code>SOMAContext</code> object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

A new [SOMA data frame](#) stored at `uri` opened for writing.

Examples

```

uri <- withr::local_tempfile(pattern = "soma-data-frame")
df <- data.frame(
  soma_joinid = bit64::seq.integer64(0L, 99L),
  group = sample(factor(c("g1", "g2")), size = 100L, replace = TRUE),

```

```

    nCount = stats::rbinom(100L, 10L, 0.3)
  )
  (sch <- arrow::infer_schema(df))
  (sdf <- SOMADataFrameCreate(uri, sch, domain = list(soma_joinid = c(0, 100))))
  sdf$write(arrow::as_arrow_table(df, schema = sch))
  sdf$close()

  (sdf <- SOMADataFrameOpen(uri))
  head(as.data.frame(sdf$read())$concat())

```

SOMADataFrameOpen	<i>Open a SOMA Data Frame</i>
-------------------	-------------------------------

Description

Factory function to open a [SOMA data frame](#) for reading (lifecycle: maturing).

Usage

```

SOMADataFrameOpen(
  uri,
  mode = "READ",
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  tiledb_timestamp = NULL,
  context = NULL
)

```

Arguments

<code>uri</code>	URI for the TileDB object.
<code>mode</code>	One of “READ” or “WRITE”.
<code>platform_config</code>	Optional platform configuration.
<code>tiledbsoma_ctx</code>	Optional (DEPRECATED) SOMATileDBContext.
<code>tiledb_timestamp</code>	Optional Datetime (POSIXct) for TileDB timestamp; defaults to the current time.
<code>context</code>	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

A [SOMA data frame](#) stored at `uri` opened in mode `mode`.

Examples

```

uri <- withr::local_tempfile(pattern = "soma-data-frame")
df <- data.frame(
  soma_joinid = bit64::seq.integer64(0L, 99L),
  group = sample(factor(c("g1", "g2")), size = 100L, replace = TRUE),
  nCount = stats::rbinom(100L, 10L, 0.3)
)
(sch <- arrow::infer_schema(df))
(sdf <- SOMADDataFrameCreate(uri, sch, domain = list(soma_joinid = c(0, 100))))
sdf$write(arrow::as_arrow_table(df, schema = sch))
sdf$close()

(sdf <- SOMADDataFrameOpen(uri))
head(as.data.frame(sdf$read()$concat()))

```

SOMADenseNDArray

*SOMA Dense Nd-Array***Description**

SOMADenseNDArray is a dense, N-dimensional array of a primitive type, with offset (zero-based) int64 integer indexing on each dimension with domain $[0, \text{maxInt64})$. The SOMADenseNDArray has a user-defined schema, which includes:

- type: a primitive type, expressed as an Arrow type (e.g., `int64`, `float32`, etc), indicating the type of data contained within the array.
- shape: the shape of the array, i.e., number and length of each dimension. This is a soft limit which can be increased using `$resize()` up to the `maxshape`.
- maxshape: the hard limit up to which shape may be increased using `$resize()`.

All dimensions must have a positive, non-zero length, and there must be 1 or more dimensions.

The default “fill” value for SOMADenseNDArray is the zero or null value of the array type (e.g., `arrow::float32()` defaults to 0.0).

Super classes

```

tildebsoma::SOMAObject -> tildebsoma::SOMAArrayBase -> tildebsoma::SOMANDArrayBase
-> SOMADenseNDArray

```

Methods**Public methods:**

- `SOMADenseNDArray$read_arrow_table()`
- `SOMADenseNDArray$read_dense_matrix()`
- `SOMADenseNDArray$write()`

- [SOMADenseNDArray\\$clone\(\)](#)

Method `read_arrow_table()`: Read as an [Arrow table](#) (lifecycle: maturing).

Usage:

```
SOMADenseNDArray$read_arrow_table(
  coords = NULL,
  result_order = "auto",
  log_level = "auto"
)
```

Arguments:

`coords` Optional list of integer vectors, one for each dimension, with a length equal to the number of values to read. If NULL, all values are read. List elements can be named when specifying a subset of dimensions.

`result_order` Optional order of read results. This can be one of either "ROW_MAJOR", "COL_MAJOR", or "auto" (default).

`log_level` Optional logging level with default value of "warn".

Returns: An [Arrow table](#).

Method `read_dense_matrix()`: Read as a dense matrix (lifecycle: maturing).

Usage:

```
SOMADenseNDArray$read_dense_matrix(
  coords = NULL,
  result_order = "ROW_MAJOR",
  log_level = "warn"
)
```

Arguments:

`coords` Optional list of integer vectors, one for each dimension, with a length equal to the number of values to read. If NULL, all values are read. List elements can be named when specifying a subset of dimensions.

`result_order` Optional order of read results. This can be one of either "ROW_MAJOR", "COL_MAJOR", or "auto" (default).

`log_level` Optional logging level with default value of "warn".

Returns: A matrix.

Method `write()`: Write matrix data to the array (lifecycle: maturing).

Note: The `$write()` method is currently limited to writing from two-dimensional matrices (lifecycle: maturing).

Usage:

```
SOMADenseNDArray$write(values, coords = NULL)
```

Arguments:

`values` A matrix. Character dimension names are ignored because SOMANDArrays use integer indexing.

coords A list of integer vectors, one for each dimension, with a length equal to the number of values to write. If NULL, the default, the values are taken from the row and column names of values.

Returns: Invisibly returns self.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
SOMADenseNDArray$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-dense-array")
mat <- matrix(stats::rnorm(100L ^ 2L), nrow = 100L, ncol = 100L)
mat[1:3, 1:5]

(arr <- SOMADenseNDArrayCreate(uri, arrow::float64(), shape = dim(mat)))
arr$write(mat)
arr$close()

(arr <- SOMADenseNDArrayOpen(uri))
arr$read_arrow_table()
```

SOMADenseNDArrayCreate

Create a SOMA Dense ND Array

Description

Factory function to create a [SOMA dense ND array](#) for writing (lifecycle: maturing).

Usage

```
SOMADenseNDArrayCreate(
  uri,
  type,
  shape,
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  tiledb_timestamp = NULL,
  context = NULL
)
```

Arguments

<code>uri</code>	URI for the TileDB object.
<code>type</code>	An Arrow type defining the type of each element in the array.
<code>shape</code>	A vector of integers defining the shape of the array.
<code>platform_config</code>	Optional platform configuration.
<code>tiledsoma_ctx</code>	Optional (DEPRECATED) SOMATileDBContext.
<code>tiledb_timestamp</code>	Optional Datetime (POSIXct) for TileDB timestamp.
<code>context</code>	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

A new [SOMA dense ND array](#) stored at `uri` opened for writing.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-dense-array")
mat <- matrix(stats::rnorm(100L ^ 2L), nrow = 100L, ncol = 100L)
mat[1:3, 1:5]

(arr <- SOMADenseNDArrayCreate(uri, arrow::float64(), shape = dim(mat)))
arr$write(mat)
arr$close()

(arr <- SOMADenseNDArrayOpen(uri))
arr$read_arrow_table()
```

SOMADenseNDArrayOpen *Open a SOMA Dense Nd Array*

Description

Factory function to open a [SOMA dense ND array](#) for reading (lifecycle: maturing).

Usage

```
SOMADenseNDArrayOpen(
  uri,
  mode = "READ",
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
```

```

    tiledb_timestamp = NULL,
    context = NULL
)

```

Arguments

<code>uri</code>	URI for the TileDB object.
<code>mode</code>	One of “READ” or “WRITE”.
<code>platform_config</code>	Optional platform configuration.
<code>tiledbsoma_ctx</code>	Optional (DEPRECATED) SOMATileDBContext.
<code>tiledb_timestamp</code>	Optional Datetime (POSIXct) for TileDB timestamp; defaults to the current time.
<code>context</code>	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

A [SOMA dense ND array](#) stored at `uri` opened in mode `mode`.

Examples

```

uri <- withr::local_tempfile(pattern = "soma-dense-array")
mat <- matrix(stats::rnorm(100L ^ 2L), nrow = 100L, ncol = 100L)
mat[1:3, 1:5]

(arr <- SOMADenseNDArrayCreate(uri, arrow::float64(), shape = dim(mat)))
arr$write(mat)
arr$close()

(arr <- SOMADenseNDArrayOpen(uri))
arr$read_arrow_table()

```

SOMAEperiment

SOMA Experiment

Description

SOMAEperiment is a specialized [SOMACollection](#), representing one or more modes of measurement across a single collection of cells (aka a “multimodal dataset”) with pre-defined fields: `obs` and `ms` (see *Active Bindings* below for details) (lifecycle: maturing).

Adding new objects to a collection

The `SOMAEperiment` class provides a number of type-specific methods for adding new a object to the collection, such as `add_new_sparse_ndarray()` and `add_new_dataframe()`. These methods will create the new object and add it as member of the `SOMAEperiment`. The new object will always inherit the parent context (see `SOMATileDBContext`) and, by default, its platform configuration (see `PlatformConfig`). However, the user can override the default platform configuration by passing a custom configuration to the `platform_config` argument.

Carrara (TileDB v3) behavior

When working with Carrara URIs (`tiledb://workspace/teamspace/...`), child objects created at a URI nested under a parent collection are **automatically added** as members of the parent. This means:

- You do not need to call `add_new_collection()` after creating a child at a nested URI—the child is already a member.
- For backward compatibility, calling `add_new_collection()` on an already-registered child is a **no-op** and will not cause an error.
- The member name must match the relative URI segment (e.g., creating at `parent_uri/child` automatically adds the child with key "child").

Super classes

`tiledsoma::SOMAObject` -> `tiledsoma::SOMACollectionBase` -> `SOMAEperiment`

Active bindings

obs A `SOMADataFrame` containing primary annotations on the observation axis. The contents of the `soma_joinid` column define the observation index domain, `obs_id`. All observations for the `SOMAEperiment` must be defined in this data frame.

ms A `SOMACollection` of named `SOMAMeasurements`.

Methods

Public methods:

- `SOMAEperiment$axis_query()`
- `SOMAEperiment$update_obs()`
- `SOMAEperiment$update_var()`
- `SOMAEperiment$clone()`

Method `axis_query()`: Subset and extract data from a `SOMAMeasurement` by querying the `obs/var` axes.

Usage:

```
SOMAEperiment$axis_query(measurement_name, obs_query = NULL, var_query = NULL)
```

Arguments:

`measurement_name` The name of the measurement to query.

obs_query, var_query An [SOMAAxisQuery](#) object for the obs/var axis.

Returns: A [SOMAEperimentAxisQuery](#) object.

Method `update_obs()`: Update the obs data frame to add or remove columns. See [SOMADataFrame\\$update\(\)](#) for more details.

Usage:

```
SOMAEperiment$update_obs(values, row_index_name = NULL)
```

Arguments:

values A data frame, [Arrow table](#), or [Arrow record batch](#).

row_index_name An optional scalar character. If provided, and if the values argument is a data frame with row names, then the row names will be extracted and added as a new column to the data frame prior to performing the update. The name of this new column will be set to the value specified by row_index_name.

Method `update_var()`: Update the var data frame to add or remove columns. See [SOMADataFrame\\$update\(\)](#) for more details.

Usage:

```
SOMAEperiment$update_var(values, measurement_name, row_index_name = NULL)
```

Arguments:

values A data frame, [Arrow table](#), or [Arrow record batch](#).

measurement_name The name of the [SOMAMeasurement](#) whose var will be updated.

row_index_name An optional scalar character. If provided, and if the values argument is a data frame with row names, then the row names will be extracted and added as a new column to the data frame prior to performing the update. The name of this new column will be set to the value specified by row_index_name.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
SOMAEperiment$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-experiment")
obs <- data.frame(
  soma_joinid = bit64::seq.integer64(0L, 99L),
  obs_id = paste0("cell_", seq_len(100L))
)
sch <- arrow::infer_schema(obs)

(exp <- SOMAEperimentCreate(uri))
sdf <- exp$add_new_dataframe(
  "obs",
  sch,
  "soma_joinid",
```

```

    list(soma_joinid = c(0, 100))
  )
  sdf$write(arrow::as_arrow_table(obs, schema = sch))
  sdf$close()
  exp$close()

  (exp <- SOMAXperimentOpen(uri))
  exp$obs

```

SOMAXperimentAxisQuery

SOMAXperiment *Axis Query*

Description

Perform an axis-based query against a [SOMAXperiment](#).

SOMAXperimentAxisQuery allows easy selection and extraction of data from a single [SOMAMeasurement](#) in a [SOMAXperiment](#), by obs/var (axis) coordinates and/or value filter. The primary use for this class is slicing [SOMAXperiment](#) X layers by obs or var value and/or coordinates. (lifecycle: maturing)

X Layer Support:

Slicing on [SOMASparseNDArray](#) X matrices is supported; slicing on [SOMADenseNDArray](#) is not supported at this time.

Result Size:

SOMAXperimentAxisQuery query class assumes it can store the full result of both axis dataframe queries in memory, and only provides incremental access to the underlying X NDArray. Accessors such as `n_obs` and `n_vars` codify this in the class.

Active bindings

`experiment` The parent [SOMAXperiment](#) object.

`indexer` The [SOMAXaxisIndexer](#) object.

`obs_query` The obs [SOMAXaxisQuery](#) object.

`var_query` The var [SOMAXaxisQuery](#) object.

`n_obs` The number of obs axis query results.

`n_vars` The number of var axis query results.

`obs_df` The obs [SOMADataFrame](#) object.

`var_df` The var [SOMADataFrame](#) object for the specified `measurement_name`.

`ms` The [SOMAMeasurement](#) object for the specified `measurement_name`.

Methods

Public methods:

- `SOMAXperimentAxisQuery$new()`
- `SOMAXperimentAxisQuery$obs()`
- `SOMAXperimentAxisQuery$var()`
- `SOMAXperimentAxisQuery$obs_joinids()`
- `SOMAXperimentAxisQuery$var_joinids()`
- `SOMAXperimentAxisQuery$X()`
- `SOMAXperimentAxisQuery$obsm()`
- `SOMAXperimentAxisQuery$varm()`
- `SOMAXperimentAxisQuery$obs_p()`
- `SOMAXperimentAxisQuery$var_p()`
- `SOMAXperimentAxisQuery$read()`
- `SOMAXperimentAxisQuery$to_sparse_matrix()`
- `SOMAXperimentAxisQuery$to_seurat()`
- `SOMAXperimentAxisQuery$to_seurat_assay()`
- `SOMAXperimentAxisQuery$to_seurat_reduction()`
- `SOMAXperimentAxisQuery$to_seurat_graph()`
- `SOMAXperimentAxisQuery$to_single_cell_experiment()`
- `SOMAXperimentAxisQuery$clone()`

Method `new()`: Create a new `SOMAXperimentAxisQuery` object.

Usage:

```
SOMAXperimentAxisQuery$new(
  experiment,
  measurement_name,
  obs_query = NULL,
  var_query = NULL
)
```

Arguments:

`experiment` A `SOMAXperiment` object.

`measurement_name` The name of the measurement to query.

`obs_query`, `var_query` An `SOMAXaxisQuery` object for the obs/var axis.

Method `obs()`: Retrieve obs `TableReadIter`

Usage:

```
SOMAXperimentAxisQuery$obs(column_names = NULL)
```

Arguments:

`column_names` A character vector of column names to retrieve

Method `var()`: Retrieve var `arrow::Table`

Usage:

```
SOMAXperimentAxisQuery$var(column_names = NULL)
```

Arguments:

`column_names` A character vector of column names to retrieve

Method `obs_joinids()`: Retrieve `soma_joinids` as an [arrow::Array](#) for obs.

Usage:

`SOMAXperimentAxisQuery$obs_joinids()`

Method `var_joinids()`: Retrieve `soma_joinids` as an [arrow::Array](#) for var.

Usage:

`SOMAXperimentAxisQuery$var_joinids()`

Method `X()`: Retrieves an X layer as a [SOMASparseNDArrayRead](#)

Usage:

`SOMAXperimentAxisQuery$X(layer_name)`

Arguments:

`layer_name` The name of the layer to retrieve.

Method `obsm()`: Retrieves an obsm layer as a [SOMASparseNDArrayRead](#)

Usage:

`SOMAXperimentAxisQuery$obsm(layer_name)`

Arguments:

`layer_name` The name of the layer to retrieve

Method `varm()`: Retrieves a varm layer as a [SOMASparseNDArrayRead](#)

Usage:

`SOMAXperimentAxisQuery$varm(layer_name)`

Arguments:

`layer_name` The name of the layer to retrieve

Method `obsp()`: Retrieves an obsp layer as a [SOMASparseNDArrayRead](#)

Usage:

`SOMAXperimentAxisQuery$obsp(layer_name)`

Arguments:

`layer_name` The name of the layer to retrieve

Method `varp()`: Retrieves a varp layer as a [SOMASparseNDArrayRead](#)

Usage:

`SOMAXperimentAxisQuery$varp(layer_name)`

Arguments:

`layer_name` The name of the layer to retrieve

Method `read()`: Reads the entire query result as a list of [arrow::Tables](#). This is a low-level routine intended to be used by loaders for other in-core formats, such as Seurat, which can be created from the resulting Tables.

Usage:

```
SOMAXperimentAxisQuery$read(
  X_layers = NULL,
  obs_column_names = NULL,
  var_column_names = NULL
)
```

Arguments:

`X_layers` The name(s) of the X layer(s) to read and return.

`obs_column_names`, `var_column_names` Specify which column names in var and obs dataframes to read and return.

Method `to_sparse_matrix()`: Retrieve a collection layer as a sparse matrix with named dimensions.

Load any layer from the X, obsm, varm, obsp, or varp collections as a [sparse matrix](#).

By default the matrix dimensions are named using the `soma_joinid` values in the specified layer's dimensions (e.g., `soma_dim_0`). However, dimensions can be named using values from any obs or var column that uniquely identifies each record by specifying the `obs_index` and `var_index` arguments.

For layers in obsm or varm, the column axis (the axis not indexed by “obs” or “var”) is set to the range of values present in “soma_dim_1”; this ensures that gaps in this axis are preserved (eg. when a query for “obs” that results in selecting entries that are all zero for a given PC)

Usage:

```
SOMAXperimentAxisQuery$to_sparse_matrix(
  collection,
  layer_name,
  obs_index = NULL,
  var_index = NULL
)
```

Arguments:

`collection` The [SOMACollection](#) containing the layer of interest, either: “X”, “obsm”, “varm”, “obsp”, or “varp”.

`layer_name` Name of the layer to retrieve from the collection.

`obs_index`, `var_index` Name of the column in obs or var (`var_index`) containing values that should be used as dimension labels in the resulting matrix. Whether the values are used as row or column labels depends on the selected collection:

Collection	obs_index	var_index
X	row names	column names
obsm	row names	ignored
varm	ignored	row names
obsp	row and column names	ignored
varp	ignored	row and column names

Returns: A [Matrix::sparseMatrix](#)

Method `to_seurat()`: Loads the query as a [Seurat](#) object

Usage:

```
SOMAXperimentAxisQuery$to_seurat(
  X_layers = c(counts = "counts", data = "logcounts"),
  obs_index = NULL,
  var_index = NULL,
  obs_column_names = NULL,
  var_column_names = NULL,
  obsm_layers = NULL,
  varm_layers = NULL,
  obsp_layers = NULL,
  drop_levels = FALSE,
  version = NULL
)
```

Arguments:

X_layers A named character of X layers to add to the Seurat assay where the names are the names of Seurat slots and the values are the names of layers within X; names should be one of:

- “counts” to add the layer as counts
- “data” to add the layer as data
- “scale.data” to add the layer as scale.data

At least one of “counts” or “data” is required

obs_index Name of column in obs to add as cell names; uses `paste0("cell", obs_joinids())` by default

var_index Name of column in var to add as feature names; uses `paste0("feature", var_joinids())` by default

obs_column_names Names of columns in obs to add as cell-level meta data; by default, loads all columns

var_column_names Names of columns in var to add as feature-level meta data; by default, loads all columns

obsm_layers Names of arrays in obsm to add as the cell embeddings; pass FALSE to suppress loading in any dimensional reductions; by default, loads all dimensional reduction information

varm_layers Named vector of arrays in varm to load in as the feature loadings; names must be names of arrays in obsm (eg. `varm_layers = c(X_pca = "PCs")`); pass FALSE to suppress loading in any feature loadings; will try to determine varm_layers from obsm_layers

obsp_layers Names of arrays in obsp to load in as [Graphs](#); by default, loads all graphs

drop_levels Drop unused levels from obs and var factor columns

version Assay version to read query in as; by default, will try to infer assay type from the measurement itself

Returns: A [Seurat](#) object

Method `to_seurat_assay()`: Loads the query as a Seurat [Assay](#)

Usage:

```
SOMAXperimentAxisQuery$to_seurat_assay(
  X_layers = c(counts = "counts", data = "logcounts"),
```

```

    obs_index = NULL,
    var_index = NULL,
    var_column_names = NULL,
    drop_levels = FALSE,
    version = NULL
)

```

Arguments:

X_layers A named character of X layers to add to the Seurat assay where the names are the names of Seurat slots and the values are the names of layers within X; names should be one of:

- “counts” to add the layer as counts
- “data” to add the layer as data
- “scale.data” to add the layer as scale.data

At least one of “counts” or “data” is required

obs_index Name of column in obs to add as cell names; uses `paste0("cell", obs_joinids())` by default

var_index Name of column in var to add as feature names; uses `paste0("feature", var_joinids())` by default

var_column_names Names of columns in var to add as feature-level meta data; by default, loads all columns

drop_levels Drop unused levels from var factor columns

version Assay version to read query in as; by default, will try to infer assay type from the measurement itself

Returns: An [Assay](#) object

Method `to_seurat_reduction()`: Loads the query as a Seurat [dimensional reduction](#)

Usage:

```

SOMAEperimentAxisQuery$to_seurat_reduction(
  obsm_layer,
  varm_layer = NULL,
  obs_index = NULL,
  var_index = NULL
)

```

Arguments:

obsm_layer Name of array in obsm to load as the cell embeddings

varm_layer Name of the array in varm to load as the feature loadings; by default, will try to determine varm_layer from obsm_layer

obs_index Name of column in obs to add as cell names; uses `paste0("cell", obs_joinids())` by default

var_index Name of column in var to add as feature names; uses `paste0("feature", var_joinids())` by default

Returns: A [DimReduc](#) object

Method `to_seurat_graph()`: Loads the query as a Seurat [graph](#)

Usage:

SOMAXperimentAxisQuery\$to_seurat_graph(osp_layer, obs_index = NULL)

Arguments:

osp_layer Name of array in osp to load as the graph

obs_index Name of column in obs to add as cell names; uses paste0("cell", obs_joinids()) by default

Returns: A [Graph](#) object

Method to_single_cell_experiment(): Loads the query as a [SingleCellExperiment](#) object

Usage:

```
SOMAXperimentAxisQuery$to_single_cell_experiment(
  X_layers = NULL,
  obs_index = NULL,
  var_index = NULL,
  obs_column_names = NULL,
  var_column_names = NULL,
  obsm_layers = NULL,
  osp_layers = NULL,
  varp_layers = NULL,
  drop_levels = FALSE
)
```

Arguments:

X_layers A character vector of X layers to add as assays in the main experiment; may optionally be named to set the name of the resulting assay (eg. X_layers = c(counts = "raw") will load in X layer "raw" as assay "counts"); by default, loads in all X layers

obs_index Name of column in obs to add as cell names; uses paste0("cell", obs_joinids()) by default

var_index Name of column in var to add as feature names; uses paste0("feature", var_joinids()) by default

obs_column_names Names of columns in obs to add as colData; by default, loads all columns

var_column_names Names of columns in var to add as rowData; by default, loads all columns

obsm_layers Names of arrays in obsm to add as the reduced dimensions; pass FALSE to suppress loading in any reduced dimensions; by default, loads all reduced dimensions

osp_layers Names of arrays in osp to load in as [SelfHits](#); by default, loads all graphs

varp_layers Names of arrays in varp to load in as [SelfHits](#); by default, loads all networks

drop_levels Drop unused levels from obs and var factor columns

Returns: A [SingleCellExperiment](#) object

Method clone(): The objects of this class are cloneable with this method.

Usage:

SOMAXperimentAxisQuery\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

SOMAEperimentCreate *Create a SOMA Experiment*

Description

Factory function to create a [SOMA experiment](#) for writing (lifecycle: maturing).

Usage

```
SOMAEperimentCreate(
  uri,
  ingest_mode = c("write", "resume"),
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  tiledb_timestamp = NULL,
  context = NULL
)
```

Arguments

<code>uri</code>	URI for the TileDB object.
<code>ingest_mode</code>	Ingestion mode when creating the TileDB object; choose from: • “write”: create a new TileDB object and error if it already exists. • “resume”: attempt to create a new TileDB object; if it already exists, simply open it for writing.
<code>platform_config</code>	Optional platform configuration.
<code>tiledbsoma_ctx</code>	Optional (DEPRECATED) SOMATileDBContext.
<code>tiledb_timestamp</code>	Optional Datetime (POSIXct) for TileDB timestamp.
<code>context</code>	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

A new [SOMA experiment](#) stored at `uri` opened for writing.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-experiment")
obs <- data.frame(
  soma_joinid = bit64::seq.integer64(0L, 99L),
  obs_id = paste0("cell_", seq_len(100L))
)
sch <- arrow::infer_schema(obs)
```

```

(exp <- SOMAEperimentCreate(uri))
sdf <- exp$add_new_dataframe(
  "obs",
  sch,
  "soma_joinid",
  list(soma_joinid = c(0, 100))
)
sdf$write(arrow::as_arrow_table(obs, schema = sch))
sdf$close()
exp$close()

(exp <- SOMAEperimentOpen(uri))
exp$obs

```

SOMAEperimentOpen	<i>Open SOMA Experiment</i>
-------------------	-----------------------------

Description

Factory function to open a [SOMA experiment](#) for reading (lifecycle: maturing).

Usage

```

SOMAEperimentOpen(
  uri,
  mode = "READ",
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  tiledb_timestamp = NULL,
  context = NULL
)

```

Arguments

<code>uri</code>	URI for the TileDB object.
<code>mode</code>	One of “READ” or “WRITE”.
<code>platform_config</code>	Optional platform configuration.
<code>tiledbsoma_ctx</code>	Optional (DEPRECATED) SOMATileDBContext.
<code>tiledb_timestamp</code>	Optional Datetime (POSIXct) for TileDB timestamp; defaults to the current time. If not NULL, all members accessed through the collection inherit the timestamp.

context Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call `set_default_context` once before other SOMA operations to configure the default context.

Value

A [SOMA experiment](#) stored at `uri` opened in mode `mode`.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-experiment")
obs <- data.frame(
  soma_joinid = bit64::seq.integer64(0L, 99L),
  obs_id = paste0("cell_", seq_len(100L))
)
sch <- arrow::infer_schema(obs)

(exp <- SOMAExperimentCreate(uri))
sdf <- exp$add_new_dataframe(
  "obs",
  sch,
  "soma_joinid",
  list(soma_joinid = c(0, 100))
)
sdf$write(arrow::as_arrow_table(obs, schema = sch))
sdf$close()
exp$close()

(exp <- SOMAExperimentOpen(uri))
exp$obs
```

SOMAMeasurement

SOMA Measurement

Description

A `SOMAMeasurement` is a sub-element of a [SOMAExperiment](#), and is otherwise a specialized [SOMACollection](#) with pre-defined fields: `X`, `var`, `obsm/varm`, and `obsp/varp` (see *Active Bindings* below for details) (lifecycle: maturing).

Adding new objects to a collection

The `SOMAMeasurement` class provides a number of type-specific methods for adding new a object to the collection, such as `add_new_sparse_ndarray()` and `add_new_dataframe()`. These methods will create the new object and add it as member of the `SOMAMeasurement`. The new object will always inherit the parent context (see [SOMATileDBContext](#)) and, by default, its platform configuration (see [PlatformConfig](#)). However, the user can override the default platform configuration by passing a custom configuration to the `platform_config` argument.

Carrara (TileDB v3) behavior

When working with Carrara URIs (`tiledb://workspace/teamspace/...`), child objects created at a URI nested under a parent collection are **automatically added** as members of the parent. This means:

- You do not need to call `add_new_collection()` after creating a child at a nested URI—the child is already a member.
- For backward compatibility, calling `add_new_collection()` on an already-registered child is a **no-op** and will not cause an error.
- The member name must match the relative URI segment (e.g., creating at `parent_uri/child` automatically adds the child with key "child").

Super classes

```
tiledsoma::SOMASObject -> tiledbsoma::SOMACollectionBase -> SOMAMeasurement
```

Active bindings

`var` A `SOMADataFrame` containing primary annotations on the variable axis, for variables in this measurement (i.e., annotates columns of `X`). The contents of the `soma_joinid` column define the variable index domain, `var_id`. All variables for this measurement must be defined in this data frame.

`X` A `SOMACollection` of `SOMASparseNDArrays`, each contains measured feature values indexed by `[obsid, varid]`.

`obsid` A `SOMACollection` of `SOMADenseNDArrays` containing annotations on the observation axis. Each array is indexed by `obsid` and has the same shape as `obs`.

`obsid` A `SOMACollection` of `SOMASparseNDArrays` containing pairwise annotations on the observation axis and indexed with `[obsid_1, obsid_2]`.

`varid` A `SOMACollection` of `SOMADenseNDArrays` containing annotations on the variable axis. Each array is indexed by `varid` and has the same shape as `var`.

`varid` A `SOMACollection` of `SOMASparseNDArrays` containing pairwise annotations on the variable axis and indexed with `[varid_1, varid_2]`.

Methods

Public methods:

- `SOMAMeasurement$clone()`

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
SOMAMeasurement$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```

uri <- withr::local_tempfile(pattern = "soma-measurement")
var <- data.frame(
  soma_joinid = bit64::seq.integer64(0L, 99L),
  var_id = paste0("feature_", seq_len(100L))
)
sch <- arrow::infer_schema(var)

(ms <- SOMAMeasurementCreate(uri))
sdf <- ms$add_new_dataframe(
  "var",
  sch,
  "soma_joinid",
  list(soma_joinid = c(0, 100))
)
sdf$write(arrow::as_arrow_table(var, schema = sch))
sdf$close()
ms$close()

(ms <- SOMAMeasurementOpen(uri))
ms$var

```

SOMAMeasurementCreate *Create a SOMA Measurement*

Description

Factory function to create a [SOMA measurement](#) for writing (lifecycle: maturing).

Usage

```

SOMAMeasurementCreate(
  uri,
  ingest_mode = c("write", "resume"),
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  tiledb_timestamp = NULL,
  context = NULL
)

```

Arguments

uri	URI for the TileDB object.
ingest_mode	Ingestion mode when creating the TileDB object; choose from: “write”: create a new TileDB object and error if it already exists.

- “resume”: attempt to create a new TileDB object; if it already exists, simply open it for writing.

platform_config Optional platform configuration.

tiledsoma_ctx Optional (DEPRECATED) SOMATileDBContext.

tiledb_timestamp Optional Datetime (POSIXct) for TileDB timestamp.

context Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call `set_default_context` once before other SOMA operations to configure the default context.

Value

A new [SOMA measurement](#) stored at `uri` opened for writing.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-measurement")
var <- data.frame(
  soma_joinid = bit64::seq.integer64(0L, 99L),
  var_id = paste0("feature_", seq_len(100L))
)
sch <- arrow::infer_schema(var)

(ms <- SOMAMeasurementCreate(uri))
sdf <- ms$add_new_dataframe(
  "var",
  sch,
  "soma_joinid",
  list(soma_joinid = c(0, 100))
)
sdf$write(arrow::as_arrow_table(var, schema = sch))
sdf$close()
ms$close()

(ms <- SOMAMeasurementOpen(uri))
ms$var
```

SOMAMeasurementOpen *Open SOMA Measurement*

Description

Factory function to open a [SOMA measurement](#) for reading (lifecycle: maturing).

Usage

```
SOMAMeasurementOpen(
  uri,
  mode = "READ",
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  tiledb_timestamp = NULL,
  context = NULL
)
```

Arguments

<code>uri</code>	URI for the TileDB object.
<code>mode</code>	One of “READ” or “WRITE”.
<code>platform_config</code>	Optional platform configuration.
<code>tiledbsoma_ctx</code>	Optional (DEPRECATED) SOMATileDBContext.
<code>tiledb_timestamp</code>	Optional Datetime (POSIXct) for TileDB timestamp; defaults to the current time. If not NULL, all members accessed through the collection inherit the timestamp.
<code>context</code>	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

A [SOMA measurement](#) stored at `uri` opened in mode `mode`.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-measurement")
var <- data.frame(
  soma_joinid = bit64::seq.integer64(0L, 99L),
  var_id = paste0("feature_", seq_len(100L))
)
sch <- arrow::infer_schema(var)

(ms <- SOMAMeasurementCreate(uri))
sdf <- ms$add_new_dataframe(
  "var",
  sch,
  "soma_joinid",
  list(soma_joinid = c(0, 100))
)
sdf$write(arrow::as_arrow_table(var, schema = sch))
sdf$close()
ms$close()
```

```
(ms <- SOMAMeasurementOpen(uri))
ms$var
```

SOMAOpen

*Open a SOMA Object***Description**

Utility function to open the corresponding SOMA object given a URI (lifecycle: maturing).

Usage

```
SOMAOpen(
  uri,
  mode = "READ",
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  context = NULL,
  tiledb_timestamp = NULL
)
```

Arguments

<code>uri</code>	URI for the TileDB object.
<code>mode</code>	One of “READ” or “WRITE”
<code>platform_config</code>	Optional platform configuration.
<code>tiledbsoma_ctx</code>	Optional (DEPRECATED) SOMATileDBContext.
<code>context</code>	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.
<code>tiledb_timestamp</code>	Optional Datetime (POSIXct) for TileDB timestamp; defaults to the current time. If not NULL, all members accessed through the collection inherit the timestamp.

Value

A SOMA object

Examples

```

dir <- withr::local_tempfile(pattern = "soma-open")
dir.create(dir, recursive = TRUE)

uri <- extract_dataset("soma-exp-pbmc-small", dir)
(exp <- SOMAOpen(uri))

uri <- extract_dataset("soma-dataframe-pbmc3k-processed-obs", dir)
(obs <- SOMAOpen(uri))

```

SOMASparseNDArray	<i>SOMA Sparse Nd-Array</i>
-------------------	-----------------------------

Description

SOMASparseNDArray is a sparse, N-dimensional array with offset (zero-based) integer indexing on each dimension. The SOMASparseNDArray has a user-defined schema, which includes:

- type: a primitive type, expressed as an Arrow type (e.g., [int64](#), [float32](#), etc), indicating the type of data contained within the array.
- shape: the shape of the array, i.e., number and length of each dimension. This is a soft limit which can be increased using `$resize()` up to the `maxshape`.
- maxshape: the hard limit up to which shape may be increased using `$resize()`.

All dimensions must have a positive, non-zero length.

Duplicate Writes

As duplicate index values are not allowed, index values already present in the object are overwritten and new index values are added (lifecycle: maturing).

Super classes

```

tiledbsoma::SOMAObject -> tiledbsoma::SOMAArrayBase -> tiledbsoma::SOMANDArrayBase
-> SOMASparseNDArray

```

Methods**Public methods:**

- [SOMASparseNDArray\\$read\(\)](#)
- [SOMASparseNDArray\\$write\(\)](#)
- [SOMASparseNDArray\\$nnz\(\)](#)

- [SOMASparseNDArray\\$.write_coordinates\(\)](#)
- [SOMASparseNDArray\\$clone\(\)](#)

Method `read()`: Reads a user-defined slice of the `SOMASparseNDArray`.

Usage:

```
SOMASparseNDArray$read(
  coords = NULL,
  result_order = "auto",
  log_level = "auto"
)
```

Arguments:

`coords` Optional list of integer vectors, one for each dimension, with a length equal to the number of values to read. If `NULL`, all values are read. List elements can be named when specifying a subset of dimensions.

`result_order` Optional order of read results. This can be one of either `"ROW_MAJOR"`, `"COL_MAJOR"`, or `"auto"` (default).

`log_level` Optional logging level with default value of `"warn"`.

Returns: A [SOMASparseNDArrayRead](#).

Method `write()`: Write matrix-like data to the array (lifecycle: maturing).

Usage:

```
SOMASparseNDArray$write(values, bbox = NULL)
```

Arguments:

`values` Any matrix-like object coercible to a [TsparseMatrix](#). Character dimension names are ignored because `SOMANDArrays` use integer indexing.

`bbox` A vector of integers describing the upper bounds of each dimension of values. Generally should be `NULL`.

Returns: Invisibly returns `self`.

Method `nnz()`: Retrieve number of non-zero elements (lifecycle: maturing).

Usage:

```
SOMASparseNDArray$nnz()
```

Returns: A scalar with the number of non-zero elements.

Method `.write_coordinates()`: Write a COO table to the array.

Usage:

```
SOMASparseNDArray$.write_coordinates(values)
```

Arguments:

`values` A `data.frame` or [Arrow table](#) with data in COO format; must be named with the dimension and attribute labels of the array.

Returns: Invisibly returns `self`.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
SOMASparseNDArray$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

In TileDB this is an sparse array with N int64 dimensions of domain $[0, \text{maxInt64})$ and a single attribute.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-sparse-array")
mat <- Matrix::rsparsematrix(100L, 100L, 0.7, repr = "T")
mat[1:3, 1:5]

(arr <- SOMASparseNDArrayCreate(uri, arrow::float64(), shape = dim(mat)))
arr$write(mat)
arr$close()

(arr <- SOMASparseNDArrayOpen(uri))
m2 <- arr$read()$sparse_matrix()$concat()
m2[1:3, 1:5]
```

SOMASparseNDArrayCreate

Create a SOMA Sparse ND Array

Description

Factory function to create a [SOMA sparse ND array](#) for writing (lifecycle: maturing).

Usage

```
SOMASparseNDArrayCreate(
  uri,
  type,
  shape,
  ingest_mode = c("write", "resume"),
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  tiledb_timestamp = NULL,
  context = NULL
)
```

Arguments

uri	URI for the TileDB object.
type	An Arrow type defining the type of each element in the array.
shape	A vector of integers defining the shape of the array.
ingest_mode	Ingestion mode when creating the TileDB object; choose from:

- “write”: create a new TileDB object and error if it already exists.
- “resume”: attempt to create a new TileDB object; if it already exists, simply open it for writing.

`platform_config` Optional platform configuration.

`tiledb_soma_ctx` Optional (DEPRECATED) SOMATileDBContext.

`tiledb_timestamp` Optional Datetime (POSIXct) for TileDB timestamp.

`context` Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call `set_default_context` once before other SOMA operations to configure the default context.

Value

A new [SOMA sparse ND array](#) stored at `uri` opened for writing.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-sparse-array")
mat <- Matrix::rsparsematrix(100L, 100L, 0.7, repr = "T")
mat[1:3, 1:5]

(arr <- SOMASparseNDArrayCreate(uri, arrow::float64(), shape = dim(mat)))
arr$write(mat)
arr$close()

(arr <- SOMASparseNDArrayOpen(uri))
m2 <- arr$read()$sparse_matrix()$concat()
m2[1:3, 1:5]
```

SOMASparseNDArrayOpen *Open a SOMA Sparse ND Array*

Description

Factory function to open a [SOMA sparse ND array](#) for reading (lifecycle: maturing).

Usage

```
SOMASparseNDArrayOpen(
  uri,
  mode = "READ",
  platform_config = NULL,
  tiledb_soma_ctx = NULL,
  tiledb_timestamp = NULL,
  context = NULL
)
```

Arguments

<code>uri</code>	URI for the TileDB object.
<code>mode</code>	One of “READ” or “WRITE”.
<code>platform_config</code>	Optional platform configuration.
<code>tiledsoma_ctx</code>	Optional (DEPRECATED) SOMATileDBContext.
<code>tiledb_timestamp</code>	Optional Datetime (POSIXct) for TileDB timestamp; defaults to the current time.
<code>context</code>	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

A [SOMA sparse ND array](#) stored at `uri` opened in mode `mode`.

Examples

```
uri <- withr::local_tempfile(pattern = "soma-sparse-array")
mat <- Matrix::rsparsematrix(100L, 100L, 0.7, repr = "T")
mat[1:3, 1:5]

(arr <- SOMASparseNDArrayCreate(uri, arrow::float64(), shape = dim(mat)))
arr$write(mat)
arr$close()

(arr <- SOMASparseNDArrayOpen(uri))
m2 <- arr$read()$sparse_matrix()$concat()
m2[1:3, 1:5]
```

SOMATileDBContext

SOMA TileDB Context

Description

Context map for TileDB-backed SOMA objects

Super classes

```
tiledsoma::MappingBase -> tiledsoma::ScalarMap -> tiledsoma::SOMAContextBase ->
SOMATileDBContext
```

Methods

Public methods:

- [SOMATileDBContext\\$new\(\)](#)
- [SOMATileDBContext\\$keys\(\)](#)
- [SOMATileDBContext\\$items\(\)](#)
- [SOMATileDBContext\\$length\(\)](#)
- [SOMATileDBContext\\$get\(\)](#)
- [SOMATileDBContext\\$set\(\)](#)
- [SOMATileDBContext\\$to_tiledb_context\(\)](#)
- [SOMATileDBContext\\$context\(\)](#)
- [SOMATileDBContext\\$clone\(\)](#)

Method new():

Usage:

```
SOMATileDBContext$new(config = NULL, cached = TRUE)
```

Arguments:

config ...

cached Force new creation

Returns: An instantiated SOMATileDBContext object

Method keys():

Usage:

```
SOMATileDBContext$keys()
```

Returns: The keys of the map

Method items():

Usage:

```
SOMATileDBContext$items()
```

Returns: Return the items of the map as a list

Method length():

Usage:

```
SOMATileDBContext$length()
```

Returns: The number of items in the map

Method get():

Usage:

```
SOMATileDBContext$get(key, default = quote(expr = ))
```

Arguments:

key Key to fetch

default Default value to fetch if key is not found; defaults to NULL

Returns: The value of key in the map, or default if key is not found

Method set():*Usage:*

SOMATileDBContext\$set(key, value)

Arguments:

key Key to set

value Value to add for key, or NULL to remove the entry for key

Returns: [chainable] Invisibly returns self with value added as key**Method to_tiledb_context():***Usage:*

SOMATileDBContext\$to_tiledb_context()

Returns: A [tiledb_ctx](#) object, dynamically constructed. Most useful for the constructor of this class.**Method context():***Usage:*

SOMATileDBContext\$context()

Returns: A [tiledb_ctx](#) object, which is a stored (and long-lived) result from to_tiledb_context.**Method clone():** The objects of this class are cloneable with this method.*Usage:*

SOMATileDBContext\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

Examples

```
(ctx <- SOMATileDBContext$new())
ctx$get("sm.mem.reader.sparse_global_order.ratio_array_data")

ctx$to_tiledb_context()
```

SparseReadIter

SOMA Read Iterator Over Sparse Matrices

Description

SparseReadIter is a class that allows for iteration over a reads on [SOMASparseNDArray](#).

Super class

[tiledbsoma::ReadIter](#) -> SparseReadIter

Methods

Public methods:

- [SparseReadIter\\$new\(\)](#)
- [SparseReadIter\\$concat\(\)](#)
- [SparseReadIter\\$clone\(\)](#)

Method `new()`: Create (lifecycle: maturing).

Usage:

```
SparseReadIter$new(sr, shape, zero_based = FALSE)
```

Arguments:

`sr` Soma reader pointer.

`shape` Shape of the full matrix.

`zero_based` Logical, if TRUE will make iterator for `Matrix::dgTMatrix-class` otherwise [matrixZeroBasedView](#).

Method `concat()`: Concatenate remainder of iterator.

Usage:

```
SparseReadIter$concat()
```

Returns: [matrixZeroBasedView](#) of `Matrix::sparseMatrix`.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
SparseReadIter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
dir <- withr::local_tempfile(pattern = "matrix-iter")
dir.create(dir, recursive = TRUE)
(exp <- load_dataset("soma-exp-pbmc-small", dir))
qry <- exp$axis_query("RNA")
xqry <- qry$X("data")

iter <- xqry$sparse_matrix()
stopifnot(inherits(iter, "SparseReadIter"))

while (!iter$read_complete()) {
  block <- iter$read_next()
}
```

TableReadIter

*SOMA Read Iterator Over Arrow Tables***Description**

TableReadIter is a class that allows for iteration over a reads on [SOMASparseNDArray](#) and [SOMADataFrame](#). Iteration chunks are retrieved as an Arrow [Table](#).

Super class

[tiledbsoma::ReadIter](#) -> TableReadIter

Methods**Public methods:**

- [TableReadIter\\$concat\(\)](#)
- [TableReadIter\\$clone\(\)](#)

Method [concat\(\)](#): Concatenate remainder of iterator.

Usage:

[TableReadIter\\$concat\(\)](#)

Returns: An Arrow [Table](#).

Method [clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

[TableReadIter\\$clone\(deep = FALSE\)](#)

Arguments:

[deep](#) Whether to make a deep clone.

Examples

```
dir <- withr::local_tempfile(pattern = "table-iter")
dir.create(dir, recursive = TRUE)
(exp <- load_dataset("soma-exp-pbmc-small", dir))
qry <- exp$axis_query("RNA")
xqry <- qry$X("data")

iter <- xqry$tables()
stopifnot(inherits(iter, "TableReadIter"))

while (!iter$read_complete()) {
  block <- iter$read_next()
}
```

tiledb_soma_stats	<i>TileDB SOMA Statistics</i>
-------------------	-------------------------------

Description

These functions expose the TileDB Core functionality for performance measurements and statistics

Usage

```
tiledb_soma_stats_enable()
tiledb_soma_stats_disable()
tiledb_soma_stats_reset()
tiledb_soma_stats_dump()
tiledb_soma_stats_show()
```

Details

- `tiledb_soma_stats_enable()/tiledb_soma_stats_disable()`: Enable and disable TileDB's internal statistics
- `tiledb_soma_stats_reset()`: Reset all statistics to 0
- `tiledb_soma_stats_dump()`: Dump all statistics as a JSON string
- `tiledb_soma_stats_show()`: Pretty-print the JSON statistics

Value

`tiledb_soma_stats_show()`: a single-length character vector with the TileDB statistics encoded in JSON format

All other functions invisibly return NULL

write_soma	<i>Write a SOMA Object from an R Object</i>
------------	---

Description

Convert R objects to their appropriate SOMA counterpart function and methods can be written for it to provide a high-level $R \rightarrow$ SOMA interface.

Usage

```
write_soma(
  x,
  uri,
  ...,
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  context = NULL
)
```

Arguments

x	An object.
uri	URI for resulting SOMA object.
...	Arguments passed to other methods
platform_config	Optional platform configuration .
tiledbsoma_ctx	Optional (DEPRECATED) SOMATileDBContext .
context	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

The URI to the resulting [SOMAExperiment](#) generated from the data contained in x.

Known methods

- [Writing Seurat objects](#).
- [Writing SummarizedExperiment objects](#).
- [Writing SingleCellExperiment objects](#).

Examples

```
# Write a Bioconductor S4 DataFrame object to a SOMA
uri <- withr::local_tempfile(pattern = "s4-data-frame")
data("pbmc_small", package = "SeuratObject")
obs <- suppressWarnings(SeuratObject::UpdateSeuratObject(pbmc_small))[[[]]]
head(obs <- as(obs, "DataFrame"))

(sdf <- write_soma(obs, uri, soma_parent = NULL, relative = FALSE))

sdf$close()

# Write a Bioconductor SelfHits object to a SOMA
uri <- withr::local_tempfile(pattern = "hits")
(hits <- S4Vectors::SelfHits(
```

```

    c(2, 3, 3, 3, 3, 3, 4, 4, 4),
    c(4, 3, 2:4, 2, 2:3, 2),
    4,
    x = stats::rnorm(9L)
  ))

(arr <- write_soma(hits, uri, soma_parent = NULL, relative = FALSE))

arr$close()

# Write a character vector to a SOMA
uri <- withr::local_tempfile(pattern = "character")
(sdf <- write_soma(letters, uri, soma_parent = NULL, relative = FALSE))

sdf$close()

# Write a data.frame to a SOMA
uri <- withr::local_tempfile(pattern = "data-frame")
data("pbmc_small", package = "SeuratObject")
head(obs <- suppressWarnings(SeuratObject::UpdateSeuratObject(pbmc_small))[[[]])

(sdf <- write_soma(obs, uri, soma_parent = NULL, relative = FALSE))

sdf$close()

# Write a BPCells `IterableMatrix` to a SOMA

# Write a matrix to a SOMA
uri <- withr::local_tempfile(pattern = "matrix")
mat <- matrix(stats::rnorm(25L), nrow = 5L, ncol = 5L)
(arr <- write_soma(mat, uri, soma_parent = NULL, sparse = FALSE, relative = FALSE))

arr$close()

# Write a dense S4 Matrix to a SOMA
uri <- withr::local_tempfile(pattern = "s4-matrix")
mat <- Matrix::Matrix(stats::rnorm(25L), nrow = 5L, ncol = 5L)
(arr <- write_soma(mat, uri, soma_parent = NULL, sparse = FALSE, relative = FALSE))

arr$close()

# Write a TsparseMatrix to a SOMA
uri <- withr::local_tempfile(pattern = "tsparse-matrix")
mat <- Matrix::rsparsematrix(5L, 5L, 0.3, repr = "T")
(arr <- write_soma(mat, uri, soma_parent = NULL, relative = FALSE))

arr$close()

```

```

# Write a CsparseMatrix to a SOMA
uri <- withr::local_tempfile(pattern = "csparse-matrix")
mat <- Matrix::rsparsematrix(5L, 5L, 0.3, repr = "C")
(arr <- write_soma(mat, uri, soma_parent = NULL, relative = FALSE))

arr$close()

# Write an RsparseMatrix to a SOMA
uri <- withr::local_tempfile(pattern = "rsparse-matrix")
mat <- Matrix::rsparsematrix(5L, 5L, 0.3, repr = "R")
(arr <- write_soma(mat, uri, soma_parent = NULL, relative = FALSE))

arr$close()

```

write_soma.Seurat	Write a Seurat object to a SOMA
-------------------	---

Description

Write a [Seurat](#) object to a SOMA

Usage

```

## S3 method for class 'Seurat'
write_soma(
  x,
  uri,
  ...,
  ingest_mode = "write",
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  context = NULL
)

```

Arguments

x	A Seurat object.
uri	URI for resulting SOMA object.
...	Arguments passed to other methods
ingest_mode	Ingestion mode when creating the SOMA; choose from: “write”: create a new SOMA and error if it already exists. “resume”: attempt to create a new SOMA; if it already exists, simply open it for writing.
platform_config	Optional platform configuration .

tiledbsoma_ctx Optional (DEPRECATED) [SOMATileDBContext](#).
 context Optional [SOMAContext](#) object used for TileDB operations. If a context is not provided, then the default context will be used. Call `set_default_context` once before other SOMA operations to configure the default context.

Value

The URI to the resulting [SOMAExperiment](#) generated from the data contained in `x`.

Writing Cell-Level Metadata

Cell-level metadata is written out as a [data frame](#) called “obs” at the [experiment](#) level.

Writing v3 Assays

[Seurat Assay](#) objects are written out as individual [measurements](#):

- the “data” matrix is written out as a [sparse array](#) called “data” within the “X” group.
- the “counts” matrix, if not [empty](#), is written out as a [sparse array](#) called “counts” within the “X” group.
- the “scale.data” matrix, if not [empty](#), is written out as a [sparse array](#) called “scale_data” within the “X” group.
- feature-level metadata is written out as a [data frame](#) called “var”.

Expression matrices are transposed (cells as rows) prior to writing. All other slots, including results from extended assays (eg. SCTAssay, ChromatinAssay) are lost.

Performance Considerations: Ingestion of very large dense layers, such as `scale.data`, can be memory intensive. For better performance, users can remove these layers prior to ingestion and regenerate them after export, or ingest them separately as dense arrays for those who need to persist the exact matrix

```
# Using SeuratObject v5 syntax on a v3 `Assay`
# Cache the layer for separate ingestion, skip if planning to regenerate
mat <- object[["ASSAY"]][scale.data

# Remove the `scale.data` layer
object[["ASSAY"]][scale.data <- NULL

# Ingest the smaller object
uri <- write_soma(object, "/path/to/soma")

# Ingest the `scale.data` layer densely; needed only if persistence
# of the data is paramount
# Pad the `scale.data` layer so that its soma join IDs match the experiment
padded <- matrix(
  data = vector("numeric", length = prod(dim(object[["ASSAY"]]))),
  nrow = nrow(object[["ASSAY"]]),
  ncol = ncol(object[["ASSAY"]])
```

```

)
rowidx <- match(rownames(mat), rownames(object[["ASSAY"]]))
colidx <- match(colnames(mat), colnames(object[["ASSAY"]]))
padded[rowidx, colidx] <- mat

# Use `write_soma()` to ingest densely and register it within the `uns`
# collection; this may need to be created manually if the original
# object does not contain command logs
exp <- SOMAExperimentOpen(uri, "WRITE")
if (!match("uns", exp$names(), nomatch = 0L)) {
  # For `tilde://` URIs, set the URI for the new collection manually rather
  # than relying on `file.path()`
  uns <- SOMACollectionCreate(file.path(exp$uri, "uns"))
  exp$add_new_collection(uns, "uns")
}
arr <- write_soma(
  padded,
  "scale_data",
  soma_parent = exp$get("uns"),
  sparse = FALSE,
  key = "scale_data"
)
arr$close()
exp$close()

```

Please note that dense arrays cannot be read in using the [SOMAExperimentAxisQuery](#) mechanism; use [SOMADenseNDArray](#)\$read_dense_matrix, remembering to transpose before adding back to a Seurat object

Writing v5 Assays

Seurat v5 Assays are written out as individual [measurements](#):

- the layer matrices are written out as [sparse arrays](#) within the “X” group.
- feature-level metadata is written out as a [data frame](#) called “var”.

Expression matrices are transposed (cells as rows) prior to writing. All other slots, including results from extended assays (eg. SCTAssay, ChromatinAssay) are lost.

The following bits of metadata are written in various parts of the measurement

- “soma_ecosystem_seurat_assay_version”: written at the measurement level; indicates the Seurat assay version. Set to “v5”.
- “soma_ecosystem_seurat_v5_default_layers”: written at the “X” group level; indicates the [default layers](#).
- “soma_ecosystem_seurat_v5_ragged”: written at the “X/<layer>” array level; with a value of “ragged”, indicates whether or not the layer is ragged.
- “soma_r_type_hint”: written at the “X/<layer>” array level; indicates the R class and defining package (for S4 classes) of the original layer.

Writing DimReduces

Seurat DimReduce objects are written out to the “obsm” and “varm” groups of a **measurement**:

- cell embeddings are written out as a **sparse matrix** in the “obsm” group.
- feature loadings, if not **empty**, are written out as a **sparse matrix** in the “varm” groups; loadings are padded with NAs to include all features.

Dimensional reduction names are translated to AnnData-style names (eg. “pca” becomes X_pca for embeddings and “PCs” for loadings). All other slots, including projected feature loadings and jackstraw information, are lost.

Writing Graphs

Seurat Graph objects are written out as **sparse matrices** to the “obsp” group of a **measurement**.

Writing SeuratCommands

Seurat command logs are written out as **data frames** to the “seurat_commands” group of a **collection**.

Examples

```
uri <- withr::local_tempfile(pattern = "pbmc-small")

data("pbmc_small", package = "SeuratObject")
suppressWarnings(pbmc_small <- SeuratObject::UpdateSeuratObject(pbmc_small))

uri <- write_soma(pbmc_small, uri)

(exp <- SOMAExperimentOpen(uri))
exp$obs
exp$get("uns")$get("seurat_commands")$names()
(ms <- exp$ms$get("RNA"))
ms$var
ms$X$names()
ms$obsm$names()
ms$varm$names()
ms$obsp$names()

exp$close()
```

```
write_soma.SingleCellExperiment
```

Write a [SingleCellExperiment](#) object to a SOMA

Description

Write a [SingleCellExperiment](#) object to a SOMA

Usage

```
## S3 method for class 'SingleCellExperiment'
write_soma(
  x,
  uri,
  ms_name = NULL,
  ...,
  ingest_mode = "write",
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  context = NULL
)
```

Arguments

<code>x</code>	An object.
<code>uri</code>	URI for resulting SOMA object.
<code>ms_name</code>	Name for resulting measurement; defaults to <code>mainExpName(x)</code> .
<code>...</code>	Arguments passed to other methods
<code>ingest_mode</code>	Ingestion mode when creating the SOMA; choose from: • “write”: create a new SOMA and error if it already exists. • “resume”: attempt to create a new SOMA; if it already exists, simply open it for writing.
<code>platform_config</code>	Optional platform configuration .
<code>tiledbsoma_ctx</code>	Optional (DEPRECATED) SOMATileDBContext .
<code>context</code>	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

The URI to the resulting [SOMAExperiment](#) generated from the data contained in `x`.

Writing Reduced Dimensions

Reduced dimensions are written out as [sparse matrices](#) within the obsm group of [measurement](#) named ms_name.

Writing Column Pairs

Column-wise relationship matrices are written out as [sparse matrices](#) within the obsp group of [measurement](#) named ms_name.

Writing Row Pairs

Row-wise relationship matrices are written out as [sparse matrices](#) within the varp group of [measurement](#) named ms_name.

Writing colData

colData is written out as a [data frame](#) called “obs” at the [experiment](#) level.

Writing Assay Matrices

Each [assay matrix](#) is written out as a [sparse matrix](#) within the X group of [measurement](#) named ms_name. Names for assay matrices within X are taken from the [assay names](#). Assay matrices are transposed (samples as rows) prior to writing.

Writing rowData

rowData is written out as a [data frame](#) called “var” at the [measurement](#) level.

Examples

```
uri <- withr::local_tempfile(pattern = "single-cell-experiment")

mat <- abs(Matrix::rsparsematrix(
  230L,
  80L,
  0.3,
  dimnames = list(paste0("feature_", seq_len(230)), paste0("cell_", seq_len(80)))
))
(sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(counts = mat, logcounts = log2(mat + 1L)),
  reducedDims = list(
    pca = matrix(stats::runif(80 * 5L), nrow = 80),
    tsne = matrix(stats::rnorm(80 * 2L), nrow = 80)
  ),
  mainExpName = "RNA"
))

uri <- write_soma(sce, uri)

(exp <- SOMAExperimentOpen(uri))
```

```
exp$close()
```

```
write_soma.SummarizedExperiment
```

Write a [SummarizedExperiment](#) object to a SOMA

Description

Write a [SummarizedExperiment](#) object to a SOMA

Usage

```
## S3 method for class 'SummarizedExperiment'
write_soma(
  x,
  uri,
  ms_name,
  ...,
  ingest_mode = "write",
  platform_config = NULL,
  tiledbsoma_ctx = NULL,
  context = NULL
)
```

Arguments

<code>x</code>	An object.
<code>uri</code>	URI for resulting SOMA object.
<code>ms_name</code>	Name for resulting measurement.
<code>...</code>	Arguments passed to other methods
<code>ingest_mode</code>	Ingestion mode when creating the SOMA; choose from: “write”: create a new SOMA and error if it already exists. “resume”: attempt to create a new SOMA; if it already exists, simply open it for writing.
<code>platform_config</code>	Optional platform configuration .
<code>tiledbsoma_ctx</code>	Optional (DEPRECATED) SOMATileDBContext .
<code>context</code>	Optional SOMAContext object used for TileDB operations. If a context is not provided, then the default context will be used. Call <code>set_default_context</code> once before other SOMA operations to configure the default context.

Value

The URI to the resulting [SOMAExperiment](#) generated from the data contained in `x`.

Writing colData

colData is written out as a [data frame](#) called “obs” at the [experiment](#) level.

Writing Assay Matrices

Each [assay matrix](#) is written out as a [sparse matrix](#) within the X group of [measurement](#) named ms_name. Names for assay matrices within X are taken from the [assay names](#). Assay matrices are transposed (samples as rows) prior to writing.

Writing rowData

rowData is written out as a [data frame](#) called “var” at the [measurement](#) level.

Examples

```
uri <- withr::local_tempfile(pattern = "summarized-experiment")

mat <- abs(Matrix::rsparsematrix(
  230L,
  80L,
  0.3,
  dimnames = list(paste0("feature_", seq_len(230)), paste0("cell_", seq_len(80)))
))
(se <- SummarizedExperiment::SummarizedExperiment(list(counts = mat, logcounts = log2(mat + 1L))))

uri <- write_soma(se, uri, ms_name = "RNA")

(exp <- SOMAExperimentOpen(uri))
exp$obs
(ms <- exp$ms$get("RNA"))
ms$var
ms$X$names()

exp$close()
```

Index

`.NotYetImplemented`, [22](#)

`Arrow` record batch, [21](#), [33](#)
`Arrow` schema, [20](#)
`Arrow` table, [21](#), [28](#), [33](#), [50](#)
`Arrow` type, [30](#), [51](#)
`arrow::Array`, [36](#)
`arrow::float32`, [27](#)
`arrow::Table`, [14](#), [35](#), [36](#)
`Assay`, [38](#), [39](#), [62](#)
assay matrix, [66](#), [68](#)
assay names, [66](#), [68](#)
`Assays`, [63](#)

`collection`, [64](#)
command logs, [64](#)
`ConfigList`, [3](#), [9](#), [10](#)
configuration, [10](#)

`data frame`, [62](#), [63](#), [66](#), [68](#)
data frames, [64](#)
default layers, [63](#)
`dgTMatrix`-class, [56](#)
dimensional reduction, [39](#)
`DimReduc`, [39](#), [64](#)

`empty`, [62](#), [64](#)
example-datasets, [4](#)
experiment, [62](#), [66](#), [68](#)
`extract_dataset` (example-datasets), [4](#)

`float32`, [27](#), [49](#)

`get_default_context`, [5](#)
`Graph`, [38](#), [40](#), [64](#)
`graph`, [39](#)

`int64`, [27](#), [49](#)
`IntIndexer`, [6](#)

`list_datasets` (example-datasets), [4](#)

`load_dataset` (example-datasets), [4](#)

`mainExpName`, [65](#)
`map`, [3](#), [10](#)
`Matrix::sparseMatrix`, [7](#), [37](#)
`matrixZeroBasedView`, [7](#), [56](#)
measurement, [64](#), [66](#), [68](#)
measurements, [62](#), [63](#)

platform configuration, [20](#), [59](#), [61](#), [65](#), [67](#)
`PlatformConfig`, [9](#), [15](#), [32](#), [43](#)

`ScalarMap`, [3](#), [4](#), [9](#), [10](#)
`SelfHits`, [40](#)
`set_default_context`, [6](#), [11](#)
`set_log_level`, [12](#)
`Seurat`, [37](#), [38](#), [61](#)
`SeuratCommand`, [64](#)
`show_package_versions`, [12](#)
`SingleCellExperiment`, [40](#), [65](#)
`SOMA collection`, [16–18](#)
`SOMA data frame`, [24–26](#)
`SOMA dataframe`, [25](#)
`SOMA dense ND array`, [29–31](#)
`SOMA experiment`, [41–43](#)
`SOMA measurement`, [45–47](#)
`SOMA sparse ND array`, [51–53](#)
`SOMAAxisIndexer`, [34](#)
`SOMAAxisQuery`, [13](#), [33–35](#)
`SOMAAxisQueryResult`, [14](#)
`SOMACollection`, [15](#), [15](#), [31](#), [32](#), [37](#), [43](#), [44](#)
`SOMACollectionCreate`, [16](#)
`SOMACollectionOpen`, [17](#)
`SOMAContext`, [5](#), [11](#), [18](#)
`SOMADataFrame`, [13](#), [15](#), [19](#), [32](#), [34](#), [44](#), [57](#)
`SOMADataFrame$update()`, [33](#)
`SOMADataFrameCreate`, [20](#), [24](#)
`SOMADataFrameOpen`, [26](#)
`SOMADenseNDArray`, [15](#), [27](#), [34](#), [44](#), [63](#)
`SOMADenseNDArrayCreate`, [29](#)

SOMADenseNDArrayOpen, 30
 SOMAExperiment, 15, 31, 32, 34, 35, 43, 59, 62, 65, 67
 SOMAExperimentAxisQuery, 13, 14, 33, 34, 63
 SOMAExperimentCreate, 41
 SOMAExperimentOpen, 42
 SOMAMeasurement, 32–34, 43, 43
 SOMAMeasurementCreate, 45
 SOMAMeasurementOpen, 46
 SOMAOpen, 48
 SOMASparseNDArray, 15, 19, 34, 44, 49, 55, 57
 SOMASparseNDArrayCreate, 51
 SOMASparseNDArrayOpen, 52
 SOMASparseNDArrayRead, 36, 50
 SOMATileDBContext, 15, 32, 43, 53, 59, 62, 65, 67
 sparse array, 62
 sparse arrays, 63
 sparse matrices, 64, 66
 sparse matrix, 37, 64, 66, 68
 sparseMatrix, 56
 SparseReadIter, 55
 SummarizedExperiment, 67

 Table, 57
 TableReadIter, 21, 35, 57
 tempdir, 5
 tiledb::parse_query_condition, 21
 tiledb::parse_query_condition(), 13
 tiledb_ctx, 55
 tiledbsoma::MappingBase, 3, 9, 53
 tiledbsoma::ReadIter, 55, 57
 tiledbsoma::ScalarMap, 53
 tiledbsoma::SOMAArrayBase, 19, 27, 49
 tiledbsoma::SOMACollectionBase, 15, 32, 44
 tiledbsoma::SOMAContextBase, 53
 tiledbsoma::SOMANDArrayBase, 27, 49
 tiledbsoma::SOMAObject, 15, 19, 27, 32, 44, 49
 tiledbsoma_stats, 58
 tiledbsoma_stats_disable
 (tiledbsoma_stats), 58
 tiledbsoma_stats_dump
 (tiledbsoma_stats), 58
 tiledbsoma_stats_enable
 (tiledbsoma_stats), 58

 tiledbsoma_stats_reset
 (tiledbsoma_stats), 58
 tiledbsoma_stats_show
 (tiledbsoma_stats), 58
 TsparseMatrix, 50

 write_soma, 58
 write_soma.Seurat, 61
 write_soma.SingleCellExperiment, 65
 write_soma.SummarizedExperiment, 67
 Writing Seurat objects, 59
 Writing SingleCellExperiment objects, 59
 Writing SummarizedExperiment objects, 59