Package 'tiledb'

September 5, 2021

Type Package

Version 0.9.6

Title Sparse and Dense Multidimensional Array Storage Engine for Data Science

Description The data science storage engine 'TileDB' introduces a powerful on-disk format for multi-dimensional arrays. It supports dense and sparse arrays, dataframes and key-values stores, cloud storage ('S3', 'GCS', 'Azure'), chunked arrays, multiple compression, encryption and checksum filters, uses a fully multi-threaded implementation, supports parallel I/O, data versioning ('time travel'), metadata and groups. It is implemented as an embeddable cross-platform C++ library with APIs from several languages, and integrations.

Copyright TileDB, Inc.

License MIT + file LICENSE

URL https://github.com/TileDB-Inc/TileDB-R

BugReports https://github.com/TileDB-Inc/TileDB-R/issues

SystemRequirements cmake (only when TileDB source build selected), git (only when TileDB source build selected); on x86_64 platforms pre-built TileDB Embedded libraries are available at GitHub and are used if no TileDB installation is detected, and no other option to build or download was specified by the user.

Imports methods, Rcpp, nanotime

LinkingTo Rcpp

Suggests tinytest, rmarkdown, knitr, minidown, curl, bit64, Matrix, palmerpenguins, nycflights13, data.table, tibble

VignetteBuilder knitr

RoxygenNote 7.1.1

Encoding UTF-8

NeedsCompilation yes

Author TileDB, Inc. [aut, cph],
Dirk Eddelbuettel [cre]

Maintainer Dirk Eddelbuettel <dirk@tiledb.com>

Repository CRAN

Date/Publication 2021-09-05 00:40:01 UTC

R topics documented:

`tiledb-package` *tiledb - Interface to the TileDB Storage Manager API*

Description

The efficient multi-dimensional array management system 'TileDB' introduces a novel on-disk format that can effectively store reads. It features excellent compression, an efficient parallel I/O system which also scales well, and bindings to multiple languages.

`allows_dups` *Returns logical value whether the array schema allows duplicate values or not. This is only valid for sparse arrays.*

Description

Returns logical value whether the array schema allows duplicate values or not. This is only valid for sparse arrays.

Usage

```
allows_dups(x)

## S4 method for signature 'tiledb_array_schema'
allows_dups(x)

tiledb_array_schema_get_allows_dups(x)
```

Arguments

x	tiledb_array_schema

Value

the logical value

`allows_dups<-` *Sets toggle whether the array schema allows duplicate values or not. This is only valid for sparse arrays.*

Description

Sets toggle whether the array schema allows duplicate values or not. This is only valid for sparse arrays.

Usage

```
allows_dups(x) <- value

## S4 replacement method for signature 'tiledb_array_schema'
allows_dups(x) <- value

tiledb_array_schema_set_allows_dups(x, value)
```

Arguments

x	tiledb_array_schema
value	logical value

Value

the tiledb_array_schema object

---

array_consolidate *Consolidate fragments of a TileDB Array*

Description

This function invokes a consolidation operation. Parameters affecting the operation can be set via an optional configuration object. Start and end timestamps can also be set directly.

Usage

```
array_consolidate(
  uri,
  cfg = NULL,
  start_time,
  end_time,
  ctx = tiledb_get_context()
)
```

Arguments

uri	A character value with the URI of a TileDB Array
cfg	An optional TileDB Configuration object
start_time	An optional timestamp value, if missing config default is used
end_time	An optional timestamp value, if missing config default is used
ctx	An option TileDB Context object

Value

NULL is returned invisibly

array_vacuum *After consolidation, remove consolidated fragments of a TileDB Array*

Description

This function can remove fragments following a consolidation step. Note that vacuuming should *not* be run if one intends to use the TileDB *time-traveling* feature of opening arrays at particular timestamps.

Usage

```
array_vacuum(uri, cfg = NULL, start_time, end_time, ctx = tiledb_get_context())
```

Arguments

uri	A character value with the URI of a TileDB Array
cfg	An optional TileDB Configuration object
start_time	An optional timestamp value, if missing config default is used
end_time	An optional timestamp value, if missing config default is used
ctx	An option TileDB Context object

Details

Parameters affecting the operation can be set via an optional configuration object. Start and end timestamps can also be set directly.

Value

NULL is returned invisibly

as.data.frame.tiledb_config *Convert a* tiledb_config *object to a R data.frame*

Description

Convert a tiledb_config object to a R data.frame

Usage

```
## S3 method for class 'tiledb_config'
as.data.frame(x, ...)
```

Arguments

x	tiledb_config object
...	Extra parameter for method signature, currently unused.

Value

a data.frame wth parameter, value columns

Examples

```
cfg <- tiledb_config()
as.data.frame(cfg)
```

---

as.vector.tiledb_config

Convert a tiledb_config *object to a R vector*

Description

Convert a tiledb_config object to a R vector

Usage

```
## S3 method for class 'tiledb_config'
as.vector(x, mode = "any")
```

Arguments

x	tiledb_config object
mode	Character value "any", currently unused

Value

a character vector of config parameter names, values

Examples

```
cfg <- tiledb_config()
as.vector(cfg)
```

as_data_frame *Construct a data.frame from query results*

Description

Converts a tiledb object to a data.frame object

Usage

```
as_data_frame(dom, data, extended = FALSE)
```

Arguments

dom	tiledb_domain object
data	tiledb object to be converted
extended	optional logical variable selected wider display with coordinates, defaults to false

Value

data.frame object constructed from data

attrs,tiledb_array,ANY-method *Retrieve attributes from* tiledb_array *object*

Description

By default, all attributes will be selected. But if a subset of attribute names is assigned to the internal slot attrs, then only those attributes will be queried. This methods accesses the slot.

Usage

```
## S4 method for signature 'tiledb_array,ANY'
attrs(object)
```

Arguments

object	A tiledb_array object

Value

An empty character vector if no attributes have been selected or else a vector with attributes.

---

attrs,tiledb_array_schema,ANY-method
Returns a list of all tiledb_attr *objects associated with the* tiledb_array_schema

---

Description

Returns a list of all tiledb_attr objects associated with the tiledb_array_schema

Usage

```
## S4 method for signature 'tiledb_array_schema,ANY'
attrs(object, idx, ...)
```

Arguments

object	tiledb_array_schema
idx	index argument, currently unused.
...	Extra parameter for method signature, currently unused.

Value

a list of tiledb_attr objects

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 10L), type = "INT32")))
sch <- tiledb_array_schema(dom, attrs = c(tiledb_attr("a1", type = "INT32"),
                                          tiledb_attr("a2", type = "FLOAT64")))
attrs(sch)

lapply(attrs(sch), datatype)
```

---

attrs,tiledb_array_schema,character-method
Returns a tiledb_attr *object associated with the* tiledb_array_schema *with a given name.*

Description

Returns a tiledb_attr object associated with the tiledb_array_schema with a given name.

Usage

```
## S4 method for signature 'tiledb_array_schema,character'
attrs(object, idx, ...)
```

Arguments

object	tiledb_array_schema
idx	attribute name string
...	Extra parameter for method signature, currently unused.

Value

a tiledb_attr object

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 10L), type = "INT32")))
sch <- tiledb_array_schema(dom, attrs = c(tiledb_attr("a1", type = "INT32"),
                                          tiledb_attr("a2", type = "FLOAT64")))
attrs(sch, "a2")
```

---

attrs,tiledb_array_schema,numeric-method

Returns a tiledb_attr *object associated with the* tiledb_array_schema *with a given index*

Description

The attribute index is defined by the order the attributes were defined in the schema

Usage

```
## S4 method for signature 'tiledb_array_schema,numeric'
attrs(object, idx, ...)
```

Arguments

object	tiledb_array_schema
idx	attribute index
...	Extra parameter for method signature, currently unused.

Value

a tiledb_attr object

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 10L), type = "INT32")))
sch <- tiledb_array_schema(dom, attrs = c(tiledb_attr("a1", type = "INT32"),
                                          tiledb_attr("a2", type = "FLOAT64")))
attrs(sch, 2)
```

`attrs,tiledb_dense,ANY-method`
Retrieve attributes from `tiledb_dense` *object*

Description

By default, all attributes will be selected. But if a subset of attribute names is assigned to the internal slot `attrs`, then only those attributes will be queried. This methods accesses the slot.

Usage

```
## S4 method for signature 'tiledb_dense,ANY'
attrs(object)
```

Arguments

`object`	A `tiledb_dense` array object

Value

An empty character vector if no attributes have been selected or else a vector with attributes.

`attrs,tiledb_sparse,ANY-method`
Retrieve attributes from `tiledb_sparse` *object*

Description

By default, all attributes will be selected. But if a subset of attribute names is assigned to the internal slot `attrs`, then only those attributes will be queried. This methods accesses the slot.

Usage

```
## S4 method for signature 'tiledb_sparse,ANY'
attrs(object)
```

Arguments

object	A tiledb_sparse array object

Value

An empty character vector if no attributes have been selected or else a vector with attributes.

---

attrs<- *Selects attributes for the given TileDB array*

Description

Selects attributes for the given TileDB array

Usage

```
attrs(x) <- value

## S4 replacement method for signature 'tiledb_dense'
attrs(x) <- value
```

Arguments

x	A tiledb_dense array object
value	A character vector with attributes

Value

The modified tiledb_dense array object

---

attrs<-,tiledb_array-method *Selects attributes for the given TileDB array*

Description

Selects attributes for the given TileDB array

Usage

```
## S4 replacement method for signature 'tiledb_array'
attrs(x) <- value
```

Arguments

x	A tiledb_array object
value	A character vector with attributes

Value

The modified tiledb_array object

---

attrs<-,tiledb_sparse-method *Selects attributes for the given TileDB array*

Description

Selects attributes for the given TileDB array

Usage

```
## S4 replacement method for signature 'tiledb_sparse'
attrs(x) <- value
```

Arguments

x	A tiledb_sparse array object
value	A character vector with attributes

Value

The modified tiledb_sparse array object

---

capacity *Retrieve schema capacity (for sparse fragments)*

Description

Returns the tiledb_array schema tile capacity for sparse fragments.

Usage

```
capacity(object)

## S4 method for signature 'tiledb_array_schema'
capacity(object)

tiledb_array_schema_get_capacity(object)
```

Arguments

object	An array_schema object

Value

The tile capacity value

---

capacity<- *Sets the schema capacity (for sparse fragments)*

Description

Sets the tiledb_array schema tile capacity for sparse fragments.

Usage

```
capacity(x) <- value

## S4 replacement method for signature 'tiledb_array_schema'
capacity(x) <- value

tiledb_array_schema_set_capacity(x, value)
```

Arguments

x	An array_schema object
value	An integer or numeric value for the new tile capacity

Value

The modified array_schema object

---

cell_order,tiledb_array_schema-method *Returns the cell layout string associated with the* tiledb_array_schema

---

Description

Returns the cell layout string associated with the tiledb_array_schema

Usage

```
## S4 method for signature 'tiledb_array_schema'
cell_order(object)
```

Arguments

object	tiledb object

---

cell_val_num *Return the number of scalar values per attribute cell*

Description

Return the number of scalar values per attribute cell

Usage

```
cell_val_num(object)

## S4 method for signature 'tiledb_attr'
cell_val_num(object)

tiledb_attribute_get_cell_val_num(object)
```

Arguments

object	tiledb_attr object

Value

integer number of cells

Examples

```
a1 <- tiledb_attr("a1", type = "FLOAT64", ncells = 1)
cell_val_num(a1)
```

---

cell_val_num<- *Set the number of scalar values per attribute cell*

Description

Set the number of scalar values per attribute cell

Usage

```
cell_val_num(x) <- value

## S4 replacement method for signature 'tiledb_attr'
cell_val_num(x) <- value

tiledb_attribute_set_cell_val_num(x, value)
```

Arguments

x	A TileDB Attribute object
value	An integer value of number of cells

Value

The modified attribute is returned

---

check *Check the schema for correctness*

Description

Returns the tiledb_array schema for correctness

Usage

```
check(object)

## S4 method for signature 'tiledb_array_schema'
check(object)

tiledb_array_schema_check(object)
```

Arguments

object	An array_schema object

Value

The boolean value TRUE is returned for a correct schema; for an incorrect schema an error condition is triggered.

---

config,tiledb_ctx-method *Retrieve the* tiledb_config *object from the* tiledb_ctx

---

Description

Retrieve the tiledb_config object from the tiledb_ctx

Usage

```
## S4 method for signature 'tiledb_ctx'
config(object = tiledb_get_context())
```

Arguments

object	tiledb_ctx object

Value

tiledb_config object associated with the tiledb_ctx instance

Examples

```
ctx <- tiledb_ctx(c("sm.tile_cache_size" = "10"))
cfg <- config(ctx)
cfg["sm.tile_cache_size"]
```

---

datatype,tiledb_attr-method

Return the tiledb_attr *datatype*

Description

Return the tiledb_attr datatype

Usage

```
## S4 method for signature 'tiledb_attr'
datatype(object)
```

Arguments

object	tiledb_attr object

Value

tiledb datatype string

Examples

```
a1 <- tiledb_attr("a1", type = "INT32")
datatype(a1)

a2 <- tiledb_attr("a1", type = "FLOAT64")
datatype(a2)
```

`datatype,tiledb_dim-method`
Return the `tiledb_dim` *datatype*

Description

Return the `tiledb_dim` datatype

Usage

```
## S4 method for signature 'tiledb_dim'
datatype(object)
```

Arguments

`object`	tiledb_dim object

Value

tiledb datatype string

Examples

```
d1 <- tiledb_dim("d1", domain = c(5L, 10L), tile = 2L, type = "INT32")
datatype(d1)
```

`datatype,tiledb_domain-method`
Returns the tiledb_domain TileDB type string

Description

Returns the tiledb_domain TileDB type string

Usage

```
## S4 method for signature 'tiledb_domain'
datatype(object)
```

Arguments

`object`	tiledb_domain

Value

tiledb_domain type string

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 100L), type = "INT32")))
datatype(dom)
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(0.5, 100.0), type = "FLOAT64")))
datatype(dom)
```

---

datetimes_as_int64 *Retrieve datetimes_as_int64 toggle*

Description

A `tiledb_array` object may contain date and datetime objects. While their internal representation is generally shielded from the user, it can useful to access them as the 'native' format which is an `integer64`. This function retrieves the current value of the selection variable, which has a default of `FALSE`.

Usage

```
datetimes_as_int64(object)

## S4 method for signature 'tiledb_array'
datetimes_as_int64(object)
```

Arguments

`object`	A `tiledb_array` object

Value

A logical value indicating whether `datetimes_as_int64` is selected

datetimes_as_int64<- *Set datetimes_as_int64 toggle*

Description

A tiledb_array object may contain date and datetime objects. While their internal representation is generally shielded from the user, it can useful to access them as the 'native' format which is an integer64. This function set the current value of the selection variable, which has a default of FALSE.

Usage

```
datetimes_as_int64(x) <- value

## S4 replacement method for signature 'tiledb_array'
datetimes_as_int64(x) <- value
```

Arguments

x	A tiledb_array object
value	A logical value with the selection

Value

The modified tiledb_array array object

dim.tiledb_array_schema *Retrieve the dimension (domain extent) of the domain*

Description

Only valid for integral (integer) domains

Usage

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

Arguments

x	tiledb_array_schema

Value

a dimension vector

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 10L), type = "INT32")))
sch <- tiledb_array_schema(dom, attrs = c(tiledb_attr("a1", type = "INT32"),
                                          tiledb_attr("a2", type = "FLOAT64")))
dim(sch)
```

`dim.tiledb_dim` *Retrieves the dimension of the tiledb_dim domain*

Description

Retrieves the dimension of the tiledb_dim domain

Usage

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

Arguments

`x`	`tiledb_dim` object

Value

a vector of the tile_dim domain type, of the dim domain dimension (extent)

Examples

```
d1 <- tiledb_dim("d1", c(1L, 10L), 5L)
dim(d1)
```

`dim.tiledb_domain` *Retrieve the dimension (domain extent) of the domain*

Description

Only valid for integral (integer) domains

Usage

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

Arguments

x	tiledb_domain

Value

dimension vector

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 100L), type = "INT32"),
                              tiledb_dim("d2", c(1L, 100L), type = "INT32")))
dim(dom)
```

---

dimensions,tiledb_array_schema-method

Returns a list of `tiledb_dim` *objects associated with the* `tiledb_array_schema`

Description

Returns a list of `tiledb_dim` objects associated with the `tiledb_array_schema`

Usage

```
## S4 method for signature 'tiledb_array_schema'
dimensions(object)
```

Arguments

object	tiledb_array_schema

Value

a list of tiledb_dim objects

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 100L), type = "INT32"),
                                  tiledb_dim("d2", c(1L, 50L), type = "INT32")))
sch <- tiledb_array_schema(dom, attrs = c(tiledb_attr("a1", type = "INT32")))
dimensions(dom)

lapply(dimensions(dom), name)
```

dimensions,tiledb_domain-method *Returns a list of the tiledb_domain dimension objects*

Description

Returns a list of the tiledb_domain dimension objects

Usage

```
## S4 method for signature 'tiledb_domain'
dimensions(object)
```

Arguments

object	tiledb_domain

Value

a list of tiledb_dim

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 100L), type = "INT32"),
                              tiledb_dim("d2", c(1L, 50L), type = "INT32")))
dimensions(dom)

lapply(dimensions(dom), name)
```

domain *Generic Methods*

Description

Definition of generic methods

Usage

```
domain(object, ...)

dimensions(object, ...)

attrs(object, idx, ...)
```

```
cell_order(object, ...)

tile_order(object, ...)

filter_list(object, ...)

filter_list(x) <- value

is.sparse(object, ...)

tiledb_ndim(object, ...)

name(object)

datatype(object)

config(object, ...)

schema(object, ...)

tile(object)

is.integral(object)

nfilters(object)
```

Arguments

object	A TileDB object
...	Variable argument
idx	An index argument
x	A TileDB Object
value	A value to be assigned

`domain,tiledb_array_schema-method`
Returns the `tiledb_domain` *object associated with a given* `tiledb_array_schema`

Description

Returns the `tiledb_domain` object associated with a given `tiledb_array_schema`

Usage

```
## S4 method for signature 'tiledb_array_schema'
domain(object)
```

Arguments

object	tiledb_array_schema

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 10L), type = "INT32")))
sch <- tiledb_array_schema(dom, attrs = c(tiledb_attr("a1", type = "INT32")))
domain(sch)
```

---

domain,tiledb_dim-method

Return the tiledb_dim *domain*

Description

Return the tiledb_dim domain

Usage

```
## S4 method for signature 'tiledb_dim'
domain(object)
```

Arguments

object	tiledb_dim object

Value

a vector of (lb, ub) inclusive domain of the dimension

Examples

```
d1 <- tiledb_dim("d1", domain = c(5L, 10L))
domain(d1)
```

extended *Retrieve data.frame extended returns columns toggle*

Description

A tiledb_array object can be returned as data.frame. This methods returns the selection value for 'extended' format including row (and column, if present) indices.

Usage

```
extended(object)

## S4 method for signature 'tiledb_array'
extended(object)
```

Arguments

object	A tiledb_array object

Value

A logical value indicating whether an extended return is selected

extended<- *Set data.frame extended return columns toggle*

Description

A tiledb_array object can be returned as data.frame. This methods set the selection value for 'extended' format including row (and column, if present) indices.

Usage

```
extended(x) <- value

## S4 replacement method for signature 'tiledb_array'
extended(x) <- value
```

Arguments

x	A tiledb_array object
value	A logical value with the selection

Value

The modified tiledb_array array object

`filter_list,tiledb_array_schema-method`

Returns the offsets and coordinate filter_lists associated with the `tiledb_array_schema`

Description

Returns the offsets and coordinate filter_lists associated with the `tiledb_array_schema`

Usage

```
## S4 method for signature 'tiledb_array_schema'
filter_list(object)
```

Arguments

`object`	tiledb_array_schema

Value

a list of tiledb_filter_list objects

`filter_list,tiledb_attr-method`

Returns the TileDB Filter List object associated with the given TileDB Attribute

---

Description

Returns the TileDB Filter List object associated with the given TileDB Attribute

Usage

```
## S4 method for signature 'tiledb_attr'
filter_list(object)
```

Arguments

`object`	TileDB Attribute

Value

a tiledb_filter_list object

Examples

```
attr <- tiledb_attr(type = "INT32", filter_list=tiledb_filter_list(list(tiledb_filter("ZSTD"))))
filter_list(attr)
```

`filter_list,tiledb_dim-method`
Returns the TileDB Filter List object associated with the given TileDB Dimension

---

Description

Returns the TileDB Filter List object associated with the given TileDB Dimension

Usage

```
## S4 method for signature 'tiledb_dim'
filter_list(object)
```

Arguments

`object`	TileDB_Dimension

Value

A TileDB_filter_list object

`filter_list<-,tiledb_attr-method`
Sets the TileDB Filter List for the TileDB Attribute object

Description

Sets the TileDB Filter List for the TileDB Attribute object

Usage

```
## S4 replacement method for signature 'tiledb_attr'
filter_list(x) <- value
```

Arguments

`x`	TileDB Attribute
`value`	TileDB Filter List

Value

The modified TileDB Attribute object

`filter_list<-,tiledb_dim-method` *Sets the TileDB Filter List for the TileDB Dimension object*

Description

Sets the TileDB Filter List for the TileDB Dimension object

Usage

```
## S4 replacement method for signature 'tiledb_dim'
filter_list(x) <- value
```

Arguments

`x`	TileDB Dimension
`value`	TileDB Filter List

Value

The modified TileDB Dimension object

`fromDataFrame` *Create a TileDB dense or sparse array from a given* `data.frame` *Object*

Description

The supplied `data.frame` object is (currently) limited to integer, numeric, or character. In addition, three datetime columns are supported with the R representations of `Date`, `POSIXct` and `nanotime`.

Usage

```
fromDataFrame(
  obj,
  uri,
  col_index = NULL,
  sparse = FALSE,
  allows_dups = sparse,
  cell_order = "COL_MAJOR",
  tile_order = "COL_MAJOR",
  filter = "ZSTD",
```

```
  capacity = 10000L,
  tile_domain = NULL,
  tile_extent = NULL,
  debug = FALSE
)
```

Arguments

obj	A data.frame object.
uri	A character variable with an Array URI.
col_index	An optional column index, either numeric with a column index, or character with a column name, designating an index column; default is NULL implying an index column is added when the array is created
sparse	A logical switch to select sparse or dense (the default)
allows_dups	A logical switch to select if duplicate values are allowed or not, default is the same value as 'sparse'.
cell_order	A character variable with one of the TileDB cell order values, default is "COL_MAJOR".
tile_order	A character variable with one of the TileDB tile order values, default is "COL_MAJOR".
filter	A character variable vector, defaults to 'ZSTD', for one or more filters to be applied to each attribute;
capacity	A integer value with the schema capacity, default is 10000.
tile_domain	An integer vector or list or NULL. If an integer vector of size two it specifies the integer domain of the row dimension; if a list then a named element is used for the dimension of the same name; or if NULL the row dimension of the obj is used.
tile_extent	An integer value for the tile extent of the row dimensions; if NULL the row dimension of the obj is used. Note that the tile_extent cannot exceed the tile domain.
debug	Logical flag to select additional output

Details

The created (dense or sparse) array will have as many attributes as there are columns in the data.frame. Each attribute will be a single column. For a sparse array, one or more columns have to be designated as dimensions.

At present, factor variable are converted to character.

Value

Null, invisibly.

Examples

```
## Not run:
uri <- tempfile()
## turn factor into character
irisdf <- within(iris, Species <- as.character(Species))
fromDataFrame(irisdf, uri)
arr <- tiledb_array(uri, as.data.frame=TRUE, sparse=FALSE)
newdf <- arr[]
all.equal(iris, newdf)

## End(Not run)
```

`fromSparseMatrix` *Create (or return) a TileDB sparse array*

Description

The functions `fromSparseMatrix` and `toSparseMatrix` help in storing (and retrieving) sparse matrices using a TileDB backend.

Usage

```
fromSparseMatrix(
  obj,
  uri,
  cell_order = "ROW_MAJOR",
  tile_order = "ROW_MAJOR",
  filter = "ZSTD",
  capacity = 10000L
)

toSparseMatrix(uri)
```

Arguments

`obj`	A sparse matrix object.
`uri`	A character variable with an Array URI.
`cell_order`	A character variable with one of the TileDB cell order values, default is "COL_MAJOR".
`tile_order`	A character variable with one of the TileDB tile order values, default is "COL_MAJOR".
`filter`	A character variable vector, defaults to 'ZSTD', for one or more filters to be applied to each attribute;
`capacity`	A integer value with the schema capacity, default is 10000.

Value

Null, invisibly.

Examples

```
## Not run:
if (requireNamespace("Matrix", quietly=TRUE)) {
    library(Matrix)
    set.seed(123)      # just to fix it
    mat <- matrix(0, nrow=20, ncol=10)
    mat[sample(seq_len(200), 20)] <- seq(1, 20)
    spmat <- as(mat, "dgTMatrix")  # sparse matrix in dgTMatrix format
    uri <- "sparse_matrix"
    fromSparseMatrix(spmat, uri)   # now written
    chk <- toSparseMatrix(uri)     # and re-read
    print(chk)
    all.equal(spmat, chk)
}

## End(Not run)
```

---

has_attribute *Check a schema for a given attribute name*

Description

Check a schema for a given attribute name

Usage

```
has_attribute(schema, attr)
```

Arguments

schema	A schema for a TileDB Array
attr	A character variable with an attribute name

Value

A boolean value indicating if the attribute exists in the schema

is.anonymous *Returns TRUE if the tiledb_dim is anonymous*

Description

A TileDB attribute is anonymous if no name/label is defined

Usage

```
is.anonymous(object)

## S3 method for class 'tiledb_attr'
is.anonymous(object)
```

Arguments

object	tiledb_attr object

Value

TRUE or FALSE

Examples

```
a1 <- tiledb_attr("a1", type = "FLOAT64")
is.anonymous(a1)

a2 <- tiledb_attr("", type = "FLOAT64")
is.anonymous(a2)
```

is.anonymous.tiledb_dim *Returns TRUE if the tiledb_dim is anonymous*

Description

A TileDB dimension is anonymous if no name/label is defined

Usage

```
## S3 method for class 'tiledb_dim'
is.anonymous(object)
```

Arguments

object	tiledb_dim object

Value

TRUE or FALSE

Examples

```
d1 <- tiledb_dim("d1", c(1L, 10L), 10L)
is.anonymous(d1)

d2 <- tiledb_dim("", c(1L, 10L), 10L)
is.anonymous(d2)
```

---

is.integral,tiledb_domain-method

Returns TRUE is tiledb_domain is an integral (integer) domain

---

Description

Returns TRUE is tiledb_domain is an integral (integer) domain

Usage

```
## S4 method for signature 'tiledb_domain'
is.integral(object)
```

Arguments

object	tiledb_domain

Value

TRUE if the domain is an integral domain, else FALSE

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 100L), type = "INT32")))
is.integral(dom)
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(0.5, 100.0), type = "FLOAT64")))
is.integral(dom)
```

is.sparse,tiledb_array_schema-method

Returns TRUE if the tiledb_array_schema is sparse, else FALSE

Description

Returns TRUE if the tiledb_array_schema is sparse, else FALSE

Usage

```
## S4 method for signature 'tiledb_array_schema'
is.sparse(object)
```

Arguments

object	tiledb_array_schema

Value

TRUE if tiledb_array_schema is sparse

is.sparse,tiledb_dense-method

Returns true is if the array or array_schema is sparse

Description

Returns true is if the array or array_schema is sparse

Usage

```
## S4 method for signature 'tiledb_dense'
is.sparse(object)
```

Arguments

object	tiledb_dense

Value

FALSE

is.sparse,tiledb_sparse-method *Check if object is sparse*

Description

Check if object is sparse

Usage

```
## S4 method for signature 'tiledb_sparse'
is.sparse(object)
```

Arguments

object	TileDB object

Value

A logical value indicating whether the object is sparse

limitTileDBCores *Limit TileDB core use to a given number of cores*

Description

By default, TileDB will use all available cores on a given machine. In multi-user or multi-process settings, one may want to reduce the number of core. This function will take a given number, or default to smaller of the 'Ncpus' options value or the '"OMP_THREAD_LIMIT"' enviroment variable (or two as hard fallback).

Usage

```
limitTileDBCores(ncores, verbose = FALSE)
```

Arguments

ncores	Value of CPUs used, if missing the smaller of a fallback of two, the value of 'Ncpus' (if set) and the value of environment variable '"OMP_THREAD_LIMIT"' is used.
verbose	Optional logical toggle; if set, a short message is displayed informing the user about the value set.

Details

As this function returns a config object, its intended use is as argument to the context creating functions: `ctx <-tiledb_ctx(limitTileDBCores())`. To check that the values are set (or at a later point, still set) the config object should be retrieved via the corresponding method and this `ctx` object: `cfg <-config(ctx)`.

Value

The modified configuration object is returned invisibly.

---

max_chunk_size *Returns the filter_list's max_chunk_size*

Description

Returns the filter_list's max_chunk_size

Usage

```
max_chunk_size(object)

## S4 method for signature 'tiledb_filter_list'
max_chunk_size(object)

tiledb_filter_list_get_max_chunk_size(object)
```

Arguments

`object`	tiledb_filter_list

Value

integer max_chunk_size

Examples

```
flt <- tiledb_filter("ZSTD")
tiledb_filter_set_option(flt, "COMPRESSION_LEVEL", 5)
filter_list <- tiledb_filter_list(c(flt))
max_chunk_size(filter_list)
```

---

name,tiledb_attr-method
Return the tiledb_attr *name*

Description

Return the tiledb_attr name

Usage

```
## S4 method for signature 'tiledb_attr'
name(object)
```

Arguments

object	tiledb_attr object

Value

string name, empty string if the attribute is anonymous

Examples

```
a1 <- tiledb_attr("a1", type = "INT32")
name(a1)

a2 <- tiledb_attr(type = "INT32")
name(a2)
```

---

name,tiledb_dim-method
Return the tiledb_dim *name*

Description

Return the tiledb_dim name

Usage

```
## S4 method for signature 'tiledb_dim'
name(object)
```

Arguments

object	tiledb_dim object

Value

string name, empty string if the dimension is anonymous

Examples

```
d1 <- tiledb_dim("d1", c(1L, 10L))
name(d1)

d2 <- tiledb_dim("", c(1L, 10L))
name(d2)
```

---

nfilters,tiledb_filter_list-method

Returns the filter_list's number of filters

Description

Returns the filter_list's number of filters

Usage

```
## S4 method for signature 'tiledb_filter_list'
nfilters(object)
```

Arguments

object	tiledb_filter_list

Value

integer number of filters

Examples

```
flt <- tiledb_filter("ZSTD")
tiledb_filter_set_option(flt, "COMPRESSION_LEVEL", 5)
filter_list <- tiledb_filter_list(c(flt))
nfilters(filter_list)
```

parse_query_condition *Create a 'tiledb_query_condition' object from an expression*

Description

The grammar for query conditions is at present constraint to six operators and three boolean types.

Usage

```
parse_query_condition(expr, debug = FALSE)
```

Arguments

expr	An expression that is understood by the TileDB grammar for query conditions.
debug	A boolean toogle to enable more verbose operations, defaults to 'FALSE'.

Value

A tiledb_query_condition object

print.tiledb_metadata *Print a TileDB Array Metadata object*

Description

Print a TileDB Array Metadata object

Usage

```
## S3 method for class 'tiledb_metadata'
print(x, width = NULL, ...)
```

Arguments

x	A TileDB array object
width	Optional display width, defaults to NULL
...	Optional method arguments, currently unused

Value

The array object, invisibly

query_condition *Retrieve query_condition value for the array*

Description

A tiledb_array object can have a corresponding query condition object. This methods returns it.

Usage

```
query_condition(object)

## S4 method for signature 'tiledb_array'
query_condition(object)
```

Arguments

object	A tiledb_array object

Value

A tiledb_query_condition object

query_condition<- *Set query_condition object for the array*

Description

A tiledb_array object can have an associated query condition object to set conditions on the read queries. This methods sets the 'query_condition' object.

Usage

```
query_condition(x) <- value

## S4 replacement method for signature 'tiledb_array'
query_condition(x) <- value
```

Arguments

x	A tiledb_array object
value	A tiledb_query_conditon_object

Value

The modified tiledb_array array object

query_layout *Retrieve query_layout values for the array*

Description

A tiledb_array object can have a corresponding query with a given layout given layout. This methods returns the selection value for 'query_layout' as a character value.

Usage

```
query_layout(object)

## S4 method for signature 'tiledb_array'
query_layout(object)
```

Arguments

object	A tiledb_array object

Value

A character value describing the query layout

query_layout<- *Set query_layout return values for the array*

Description

A tiledb_array object can have an associated query with a specific layout. This methods sets the selection value for 'query_layout' from a character value.

Usage

```
query_layout(x) <- value

## S4 replacement method for signature 'tiledb_array'
query_layout(x) <- value
```

Arguments

x	A tiledb_array object
value	A character variable for the query layout. Permitted values are "ROW_MAJOR", "COL_MAJOR", "GLOBAL_ORDER", or "UNORDERD".

Value

The modified tiledb_array array object

return.array *Retrieve array return toggle*

Description

A tiledb_array object can be returned as an array (or list of arrays), or, if select, as a data.frame or as a matrix. This methods returns the selection value for the array selection.

Usage

```
return.array(object, ...)

## S4 method for signature 'tiledb_array'
return.array(object)
```

Arguments

object	A tiledb_array object
...	Currently unused

Value

A logical value indicating whether array return is selected

return.array<- *Set array return toggle*

Description

A tiledb_array object can be returned as an array (or list of arrays), or, if select, as a data.frame or a matrix. This methods sets the selection value for a array.

Usage

```
return.array(x) <- value

## S4 replacement method for signature 'tiledb_array'
return.array(x) <- value
```

Arguments

x	A tiledb_array object
value	A logical value with the selection

Value

The modified tiledb_array array object

return.data.frame *Retrieve data.frame return toggle*

Description

A tiledb_dense object can be returned as an array (or list of arrays), or, if select, as a data.frame. This methods returns the selection value.

Usage

```
return.data.frame(object, ...)

## S4 method for signature 'tiledb_dense'
return.data.frame(object)
```

Arguments

object	A tiledb_dense array object
...	Currently unused

Value

A logical value indicating whether data.frame return is selected

return.data.frame,tiledb_array-method *Retrieve data.frame return toggle*

Description

A tiledb_array object can be returned as an array (or list of arrays), or, if select, as a data.frame. This methods returns the selection value.

Usage

```
## S4 method for signature 'tiledb_array'
return.data.frame(object)
```

Arguments

object	A tiledb_array object

Value

A logical value indicating whether data.frame return is selected

return.data.frame,tiledb_sparse-method *Retrieve data.frame return toggle*

Description

A `tiledb_sparse` object can be returned as an array (or list of arrays), or, if select, as a `data.frame`. This methods returns the selection value.

Usage

```
## S4 method for signature 'tiledb_sparse'
return.data.frame(object)
```

Arguments

`object`	A `tiledb_sparse` array object

Value

A logical value indicating whether `data.frame` return is selected

return.data.frame<- *Set data.frame return toggle*

Description

A `tiledb_dense` object can be returned as an array (or list of arrays), or, if select, as a `data.frame`. This methods sets the selection value.

Usage

```
return.data.frame(x) <- value

## S4 replacement method for signature 'tiledb_dense'
return.data.frame(x) <- value
```

Arguments

`x`	A `tiledb_dense` array object
`value`	A logical value with the selection

Value

The modified `tiledb_dense` array object

`return.data.frame<-,tiledb_array-method`
Set data.frame return toggle

Description

A `tiledb_array` object can be returned as an array (or list of arrays), or, if select, as a `data.frame`. This methods sets the selection value.

Usage

```
## S4 replacement method for signature 'tiledb_array'
return.data.frame(x) <- value
```

Arguments

`x`	A `tiledb_array` object
`value`	A logical value with the selection

Value

The modified `tiledb_array` array object

`return.data.frame<-,tiledb_sparse-method`
Set data.frame return toggle

Description

A `tiledb_sparse` object can be returned as an array (or list of arrays), or, if select, as a `data.frame`. This methods sets the selection value.

Usage

```
## S4 replacement method for signature 'tiledb_sparse'
return.data.frame(x) <- value
```

Arguments

`x`	A `tiledb_sparse` array object
`value`	A logical value with the selection

Value

The modified `tiledb_sparse` array object

return.matrix *Retrieve matrix return toggle*

Description

A tiledb_array object can be returned as an array (or list of arrays), or, if select, as a data.frame or as a matrix. This methods returns the selection value for the matrix selection.

Usage

```
return.matrix(object, ...)

## S4 method for signature 'tiledb_array'
return.matrix(object)
```

Arguments

object	A tiledb_array object
...	Currently unused

Value

A logical value indicating whether matrix return is selected

return.matrix<- *Set matrix return toggle*

Description

A tiledb_array object can be returned as an array (or list of arrays), or, if select, as a data.frame or a matrix. This methods sets the selection value for a matrix.

Usage

```
return.matrix(x) <- value

## S4 replacement method for signature 'tiledb_array'
return.matrix(x) <- value
```

Arguments

x	A tiledb_array object
value	A logical value with the selection

Value

The modified tiledb_array array object

return_as *Retrieve return_as conversion preference*

Description

A `tiledb_array` object can be returned as a 'list' (default), 'array', 'matrix', 'data.frame', 'data.table' or 'tibble'. This method permits to select a preference for the returned object. The default value of 'asis' means that no conversion is performed.

Usage

```
return_as(object, ...)

## S4 method for signature 'tiledb_array'
return_as(object)
```

Arguments

`object`	A `tiledb_array` object
`...`	Currently unused

Value

A character value indicating the preferred conversion where the value is one of 'asis' (the default), 'array', 'matrix','data.frame', 'data.table', or 'tibble'.

return_as<- *Retrieve return_as conversion preference*

Description

A `tiledb_array` object can be returned as a 'list' (default), 'array', 'matrix', 'data.frame', 'data.table' or 'tibble'. This method This methods permits to set a preference of returning a `list`, `array`, `matrix`, `data.frame`, a `data.table`, or a `tibble`. The default value of "asis" means that no conversion is performed and a `list` is returned.

Usage

```
return_as(x) <- value

## S4 replacement method for signature 'tiledb_array'
return_as(x) <- value
```

Arguments

x	A tiledb_array object
value	A character value with the selection

Value

The modified tiledb_array array object

---

r_to_tiledb_type *Look up TileDB type corresponding to the type of an R object*

Description

Look up TileDB type corresponding to the type of an R object

Usage

```
r_to_tiledb_type(x)
```

Arguments

x	an R array or list

Value

single character, e.g. INT32

---

save_return_as_preference *Store object conversion preference*

Description

Save (or load) 'return_as' conversion preference in an optional config file

Usage

```
save_return_as_preference(
  value = c("asis", "array", "matrix", "data.frame", "data.table", "tibble")
)

load_return_as_preference()

get_return_as_preference()

set_return_as_preference(
  value = c("asis", "array", "matrix", "data.frame", "data.table", "tibble")
)
```

Arguments

value	A character variable with one of the six permitted values

Details

The tiledb_array object can set a preference for conversion for each retrieved object. This preference can also be enconded in a configuration file as R (version 4.0.0 or later) allows a user- and package specific configuration files. These helper functions sets and retrieve the value, respectively, or retrieve the cached value from the package environment where is it set at package load.

Note that the value must be one of 'asis' (the default), 'array', 'matrix''data.frame', 'data.table' or 'tibble'. The latter two require the corresponding package to be installed.

Value

For the setter, TRUE is returned invisibly but the function is invoked for the side effect of storing the value. For either getter, the character value.

Note

This function requires R version 4.0.0 or later to utilise the per-user config directory accessor function. For older R versions, please set the attribute directly when creating the tiledb_array object, or via the return_as() method.

---

schema,character-method

Return a schema from a URI character value

Description

Return a schema from a URI character value

Usage

```
## S4 method for signature 'character'
schema(object, ...)
```

Arguments

object	A character variable with a URI
...	Extra parameters such as 'enckey', the encryption key

Value

The scheme for the object

---

schema,tiledb_array-method

Return a schema from a tiledb_array object

Description

Return a schema from a tiledb_array object

Usage

```
## S4 method for signature 'tiledb_array'
schema(object, ...)
```

Arguments

object	tiledb array object
...	Extra parameter for function signature, currently unused

Value

The scheme for the object

---

schema,tiledb_dense-method

Returns the tiledb_dense *array* tiledb_schema *object*

Description

Returns the tiledb_dense array tiledb_schema object

Usage

```
## S4 method for signature 'tiledb_dense'
schema(object, ...)
```

Arguments

object	tiledb_dense array object
...	Extra parameter for method signature, currently unused.

Value

tiledb_schema

schema,tiledb_sparse-method *Return a schema from a sparse array*

Description

Return a schema from a sparse array

Usage

```
## S4 method for signature 'tiledb_sparse'
schema(object, ...)
```

Arguments

object	sparse array object
...	Extra parameter for function signature, currently unused

Value

The scheme for the object

selected_ranges *Retrieve selected_ranges values for the array*

Description

A tiledb_array object can have a range selection for each dimension attribute. This methods returns the selection value for 'selected_ranges' and returns a list (with one element per dimension) of two-column matrices where each row describes one pair of minimum and maximum values. Alternatively, the list can be named with the names providing the match to the corresponding dimension.

Usage

```
selected_ranges(object)

## S4 method for signature 'tiledb_array'
selected_ranges(object)
```

Arguments

object	A tiledb_array object

Value

A list which can contain a matrix for each dimension

selected_ranges<- *Set selected_ranges return values for the array*

Description

A tiledb_array object can have a range selection for each dimension attribute. This methods sets the selection value for 'selected_ranges' which is a list (with one element per dimension) of two-column matrices where each row describes one pair of minimum and maximum values. Alternatively, the list can be named with the names providing the match to the corresponding dimension.

Usage

```
selected_ranges(x) <- value

## S4 replacement method for signature 'tiledb_array'
selected_ranges(x) <- value
```

Arguments

x	A tiledb_array object
value	A list of two-column matrices where each list element 'i' corresponds to the dimension attribute 'i'. The matrices can contain rows where each row contains the minimum and maximum value of a range.

Value

The modified tiledb_array array object

set_max_chunk_size *Set the filter_list's max_chunk_size*

Description

Set the filter_list's max_chunk_size

Usage

```
set_max_chunk_size(object, value)

## S4 method for signature 'tiledb_filter_list,numeric'
set_max_chunk_size(object, value)

tiledb_filter_list_set_max_chunk_size(object, value)
```

Arguments

object	tiledb_filter_list
value	string

Examples

```
flt <- tiledb_filter("ZSTD")
tiledb_filter_set_option(flt, "COMPRESSION_LEVEL", 5)
filter_list <- tiledb_filter_list(c(flt))
set_max_chunk_size(filter_list, 10)
```

---

show,tiledb_array-method

Prints a tiledb_array object

Description

Prints a tiledb_array object

Usage

```
## S4 method for signature 'tiledb_array'
show(object)
```

Arguments

object	A tiledb array object

---

show,tiledb_array_schema-method

Prints an array schema object

Description

Prints an array schema object

Usage

```
## S4 method for signature 'tiledb_array_schema'
show(object)
```

Arguments

object	An array_schema object

---

show,tiledb_attr-method
Prints an attribute object

Description

Prints an attribute object

Usage

```
## S4 method for signature 'tiledb_attr'
show(object)
```

Arguments

object	An attribute object

---

show,tiledb_config-method
Prints the config object to STDOUT

Description

Prints the config object to STDOUT

Usage

```
## S4 method for signature 'tiledb_config'
show(object)
```

Arguments

object	tiledb_config object

Examples

```
cfg <- tiledb_config()
show(cfg)
```

---

show,tiledb_dense-method
Prints a tiledb_dense array object

Description

Prints a tiledb_dense array object

Usage

```
## S4 method for signature 'tiledb_dense'
show(object)
```

Arguments

object	A tiledb_dense array object

---

show,tiledb_domain-method
Prints an domain object

Description

Prints an domain object

Usage

```
## S4 method for signature 'tiledb_domain'
show(object)
```

Arguments

object	An domain object

---

show,tiledb_sparse-method
Prints a tiledb_sparse array object

Description

Prints a tiledb_sparse array object

Usage

```
## S4 method for signature 'tiledb_sparse'
show(object)
```

Arguments

object	A tiledb_sparse array object

---

tile,tiledb_dim-method
Return the tiledb_dim *tile extent*

Description

Return the tiledb_dim tile extent

Usage

```
## S4 method for signature 'tiledb_dim'
tile(object)
```

Arguments

object	tiledb_dim object

Value

a scalar tile extent

Examples

```
d1 <- tiledb_dim("d1", domain = c(5L, 10L), tile = 2L)
tile(d1)
```

tiledb_array *Constructs a tiledb_array object backed by a persisted tiledb array uri*

Description

tiledb_array returns a new object. This class is experimental.

Usage

```
tiledb_array(
  uri,
  query_type = c("READ", "WRITE"),
  is.sparse = NA,
  as.data.frame = FALSE,
  attrs = character(),
  extended = TRUE,
  selected_ranges = list(),
  query_layout = character(),
  datetimes_as_int64 = FALSE,
  encryption_key = character(),
  timestamp = as.POSIXct(double(), origin = "1970-01-01"),
  as.matrix = FALSE,
  as.array = FALSE,
  query_condition = new("tiledb_query_condition"),
  timestamp_start = as.POSIXct(double(), origin = "1970-01-01"),
  timestamp_end = as.POSIXct(double(), origin = "1970-01-01"),
  return_as = get_return_as_preference(),
  ctx = tiledb_get_context()
)
```

Arguments

uri	uri path to the tiledb dense array
query_type	optionally loads the array in "READ" or "WRITE" only modes.
is.sparse	optional logical switch, defaults to "NA" letting array determine it
as.data.frame	optional logical switch, defaults to "FALSE"
attrs	optional character vector to select attributes, default is empty implying all are selected
extended	optional logical switch selecting wide 'data.frame' format, defaults to "TRUE"
selected_ranges	optional A list with matrices where each matrix i describes the (min,max) pair of ranges for dimension i
query_layout	optional A value for the TileDB query layout, defaults to an empty character variable indicating no special layout is set

datetimes_as_int64
: optional A logical value selecting date and datetime value representation as 'raw' integer64 and not as Date, POSIXct or nanotime objects.

encryption_key
: optional A character value with an AES-256 encryption key in case the array was written with encryption.

timestamp
: optional A POSIXct Datetime value determining where in time the array is to be openened. Deprecated, use 'timestamp_start' instead

as.matrix
: optional logical switch, defaults to "FALSE"; currently limited to dense matrices; in the case of multiple attributes in query a list of matrices is returned

as.array
: optional logical switch, defaults to "FALSE"; in the case of multiple attributes in query a list of arrays is returned

query_condition
: optional tiledb_query_condition object, by default uninitialized without a condition; this functionality requires TileDB 2.3.0 or later

timestamp_start
: optional A POSIXct Datetime value determining the inclusive time point at which the array is to be openened. No fragments written earlier will be considered.

timestamp_end
: optional A POSIXct Datetime value determining the inclusive time point until which the array is to be openened. No fragments written earlier later be considered.

return_as
: optional A character value with the desired tiledb_array conversion, permitted values are 'asis' (default, returning a list of columns), 'array', 'matrix','data.frame', 'data.table' or 'tibble'; the latter two require the respective packages installed. The existing as.* arguments take precedent over this.

ctx
: optional tiledb_ctx

Value

tiledb_array object

---

tiledb_array-class *An S4 class for a TileDB Array*

Description

This class aims to eventually replace tiledb_dense and tiledb_sparse provided equivalent functionality based on refactored implementation utilising newer TileDB features.

Slots

ctx A TileDB context object

uri A character despription

is.sparse A logical value

`as.data.frame` A logical value

`attrs` A character vector

`extended` A logical value

`selected_ranges` An optional list with matrices where each matrix i describes the (min,max) pair of ranges for dimension i

`query_layout` An optional character value

`datetimes_as_int64` A logical value

`encryption_key` A character value

`timestamp` A POSIXct datetime variable (deprecated, use timestamp_start)

`as.matrix` A logical value

`as.array` A logical value

`query_condition` A Query Condition object

`timestamp_start` A POSIXct datetime variable for the inclusive interval start

`timestamp_end` A POSIXct datetime variable for the inclusive interval start

`return_as` A character value with the desired `tiledb_array` conversion, permitted values are 'asis' (default, returning a list of columns), 'array', 'matrix','data.frame', 'data.table' or 'tibble'; the latter two require the respective packages installed

`ptr` External pointer to the underlying implementation

`tiledb_array_close` *Close a TileDB Array*

Description

Close a TileDB Array

Usage

```
tiledb_array_close(arr)
```

Arguments

`arr`	A TileDB Array object as for example returned by `tiledb_array()`

Value

The TileDB Array object but closed

tiledb_array_create *Creates a new TileDB array given an input schema.*

Description

Creates a new TileDB array given an input schema.

Usage

```
tiledb_array_create(uri, schema, encryption_key)
```

Arguments

uri	URI specifying path to create the TileDB array object
schema	tiledb_array_schema object
encryption_key	optional A character value with an AES-256 encryption key in case the array should be encryption.

Examples

```
## Not run:
pth <- tempdir()
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 10L), type = "INT32")))
sch <- tiledb_array_schema(dom, attrs = c(tiledb_attr("a1", type = "INT32")))
tiledb_array_create(pth, sch)
tiledb_object_type(pth)

## End(Not run)
```

tiledb_array_get_non_empty_domain_from_index *Get the non-empty domain from a TileDB Array by index*

Description

This functions works for both fixed- and variable-sized dimensions and switches internally.

Usage

```
tiledb_array_get_non_empty_domain_from_index(arr, idx)
```

Arguments

arr	A TileDB Array
idx	An integer index between one the number of dimensions

Value

A two-element object is returned describing the domain of selected dimension; it will either be a numeric vector in case of a fixed-size fixed-sized dimensions, or a characer vector for a variable-sized one.

---

tiledb_array_get_non_empty_domain_from_name

Get the non-empty domain from a TileDB Array by name

Description

This functions works for both fixed- and variable-sized dimensions and switches internally.

Usage

```
tiledb_array_get_non_empty_domain_from_name(arr, name)
```

Arguments

arr	A TileDB Array
name	An character variable with a dimension name

Value

A two-element object is returned describing the domain of selected dimension; it will either be a numeric vector in case of a fixed-size fixed-sized dimensions, or a characer vector for a variable-sized one.

---

tiledb_array_is_heterogeneous

Check for Heterogeneous Domain

Description

Check for Heterogeneous Domain

Usage

```
tiledb_array_is_heterogeneous(arr)
```

Arguments

arr	A TileDB Array object

Value

A boolean indicating if the array has heterogenous domains

tiledb_array_is_homogeneous *Check for Homogeneous Domain*

Description

Check for Homogeneous Domain

Usage

```
tiledb_array_is_homogeneous(arr)
```

Arguments

arr	A TileDB Array object

Value

A boolean indicating if the array has homogeneous domains

tiledb_array_open *Open a TileDB Array*

Description

Open a TileDB Array

Usage

```
tiledb_array_open(arr, type = c("READ", "WRITE"))
```

Arguments

arr	A TileDB Array object as for example returned by `tiledb_array()`
type	A character value that must be either 'READ' or 'WRITE'

Value

The TileDB Array object but opened for reading or writing

tiledb_array_open_at *Open a TileDB Array at Timestamp*

Description

Open a TileDB Array at Timestamp

Usage

```
tiledb_array_open_at(arr, type = c("READ", "WRITE"), timestamp)
```

Arguments

arr	A TileDB Array object as for example returned by `tiledb_array()`
type	A character value that must be either 'READ' or 'WRITE'
timestamp	A Datetime object that will be converted to millisecond granularity

Value

The TileDB Array object but opened for reading or writing

tiledb_array_schema *Constructs a* tiledb_array_schema *object*

Description

Constructs a `tiledb_array_schema` object

Usage

```
tiledb_array_schema(
  domain,
  attrs,
  cell_order = "COL_MAJOR",
  tile_order = "COL_MAJOR",
  sparse = FALSE,
  coords_filter_list = NULL,
  offsets_filter_list = NULL,
  capacity = 10000L,
  allows_dups = FALSE,
  ctx = tiledb_get_context()
)
```

Arguments

domain	tiledb_domain object
attrs	a list of one or more tiledb_attr objects
cell_order	(default "COL_MAJOR")
tile_order	(default "COL_MAJOR")
sparse	(default FALSE)
coords_filter_list	(optional)
offsets_filter_list	(optional)
capacity	(optional)
allows_dups	(optional, requires 'spars' to be TRUE)
ctx	tiledb_ctx object (optional)

Examples

```
schema <- tiledb_array_schema(
              dom = tiledb_domain(
                        dims = c(tiledb_dim("rows", c(1L, 4L), 4L, "INT32"),
                                 tiledb_dim("cols", c(1L, 4L), 4L, "INT32"))),
              attrs = c(tiledb_attr("a", type = "INT32")),
              cell_order = "COL_MAJOR",
              tile_order = "COL_MAJOR",
              sparse = FALSE)
schema
```

---

tiledb_array_schema-class

An S4 class for the TileDB array schema

Description

An S4 class for the TileDB array schema

Slots

ptr An external pointer to the underlying implementation

`tiledb_array_schema_set_coords_filter_list`
Set a Filter List for Coordinate of a TileDB Schema

Description

Set a Filter List for Coordinate of a TileDB Schema

Usage

```
tiledb_array_schema_set_coords_filter_list(sch, fl)
```

Arguments

`sch`	A TileDB Array Schema object
`fl`	A TileDB Filter List object

Value

The modified Array Schema object

`tiledb_array_schema_set_offsets_filter_list`
Set a Filter List for Variable-Sized Offsets of a TileDB Schema

---

Description

Set a Filter List for Variable-Sized Offsets of a TileDB Schema

Usage

```
tiledb_array_schema_set_offsets_filter_list(sch, fl)
```

Arguments

`sch`	A TileDB Array Schema object
`fl`	A TileDB Filter List object

Value

The modified Array Schema object

tiledb_arrow_array_ptr *Allocate (or Release) Arrow Array and Schema Pointers*

Description

These functions allocate (and free) appropriate pointer objects for, respectively, Arrow array and schema objects.

Usage

```
tiledb_arrow_array_ptr()

tiledb_arrow_schema_ptr()

tiledb_arrow_array_del(ptr)

tiledb_arrow_schema_del(ptr)
```

Arguments

ptr	A pointer object previously allocated with these functions

Value

The allocating functions return the requested pointer

tiledb_attr *Contructs a* tiledb_attr *object*

Description

Contructs a tiledb_attr object

Usage

```
tiledb_attr(
  name,
  type,
  filter_list = tiledb_filter_list(),
  ncells = 1,
  nullable = FALSE,
  ctx = tiledb_get_context()
)
```

Arguments

name	The dimension name / label string; if missing default "" is used.
type	The tiledb_attr TileDB datatype string; if missing the user is alerted that this is a *required* parameter.
filter_list	(default filter_list("NONE")) The tiledb_attr filter_list
ncells	(default 1) The number of cells, use NA to signal variable length
nullable	(default FALSE) A logical switch whether the attribute can have missing values
ctx	tiledb_ctx object (optional)

Value

tiledb_dim object

Examples

```
flt <- tiledb_filter_list(list(tiledb_filter("GZIP")))
attr <- tiledb_attr(name = "a1", type = "INT32",
                    filter_list = flt)
attr
```

---

tiledb_attr-class *An S4 class for a TileDB attribute*

Description

An S4 class for a TileDB attribute

Slots

ptr External pointer to the underlying implementation

---

tiledb_attribute_get_cell_size *Get the TileDB Attribute cell size*

Description

Get the TileDB Attribute cell size

Usage

```
tiledb_attribute_get_cell_size(attr)
```

Arguments

attr	A TileDB Attribute object

Value

A numeric value with the cell size

---

tiledb_attribute_get_fill_value

Get the fill value for a TileDB Attribute

Description

Get the fill value for a TileDB Attribute

Usage

```
tiledb_attribute_get_fill_value(attr)
```

Arguments

attr	A TileDB Attribute object

Value

The fill value for the attribute

---

tiledb_attribute_get_nullable

Get the TileDB Attribute Nullable flag value

Description

Get the TileDB Attribute Nullable flag value

Usage

```
tiledb_attribute_get_nullable(attr)
```

Arguments

attr	A TileDB Attribute object

Value

A boolean value with the 'Nullable' status

`tiledb_attribute_is_variable_sized`
Check whether TileDB Attribute is variable-sized

Description

Check whether TileDB Attribute is variable-sized

Usage

```
tiledb_attribute_is_variable_sized(attr)
```

Arguments

`attr`	A TileDB Attribute object

Value

A boolean value indicating variable-size or not

`tiledb_attribute_set_fill_value`
Set the fill value for a TileDB Attribute

Description

Set the fill value for a TileDB Attribute

Usage

```
tiledb_attribute_set_fill_value(attr, value)
```

Arguments

`attr`	A TileDB Attribute object
`value`	A fill value

Value

`NULL` is returned invisibly

tiledb_attribute_set_nullable *Set the TileDB Attribute Nullable flags*

Description

Set the TileDB Attribute Nullable flags

Usage

```
tiledb_attribute_set_nullable(attr, flag)
```

Arguments

attr	A TileDB Attribute object
flag	A boolean flag to turn 'Nullable' on or off

Value

Nothing is returned

tiledb_config *Creates a* tiledb_config *object*

Description

Note that for actually setting persistent values, the (altered) config object needs to used to create (or update) the tiledb_ctx object. Similarly, to check whether values are set, one should use the config method of the of the tiledb_ctx object. Examples for this are ctx <-tiledb_ctx(limitTileDBCores()) to use updated configuration values to create a context object, and cfg <-config(ctx) to retrieve it.

Usage

```
tiledb_config(config = NA_character_)
```

Arguments

config	(optional) character vector of config parameter names, values

Value

tiledb_config object

Examples

```
cfg <- tiledb_config()
cfg["sm.tile_cache_size"]

# set tile cache size to custom value
cfg <- tiledb_config(c("sm.tile_cache_size" = "100"))
cfg["sm.tile_cache_size"]
```

---

tiledb_config-class *An S4 class for a TileDB configuration*

Description

An S4 class for a TileDB configuration

Slots

`ptr` An external pointer to the underlying implementation

---

tiledb_config_load *Load a saved* `tiledb_config` *file from disk*

Description

Load a saved `tiledb_config` file from disk

Usage

```
tiledb_config_load(path)
```

Arguments

`path`	path to the config file

Examples

```
tmp <- tempfile()
cfg <- tiledb_config(c("sm.tile_cache_size" = "10"))
pth <- tiledb_config_save(cfg, tmp)
cfg <- tiledb_config_load(pth)
cfg["sm.tile_cache_size"]
```

tiledb_config_save *Save a* tiledb_config *object ot a local text file*

Description

Save a tiledb_config object ot a local text file

Usage

```
tiledb_config_save(config, path)
```

Arguments

config	The tiledb_config object
path	The path to config file to be created

Value

path to created config file

Examples

```
tmp <- tempfile()
cfg <- tiledb_config(c("sm.tile_cache_size" = "10"))
pth <- tiledb_config_save(cfg, tmp)

cat(readLines(pth), sep = "\n")
```

tiledb_config_unset *Unset a TileDB Config parameter to its default value*

Description

Unset a TileDB Config parameter to its default value

Usage

```
tiledb_config_unset(config, param)
```

Arguments

config	A TileDB Config object
param	A character variable with the parameter name

Value

The modified TileDB Config object

---

tiledb_ctx *Creates a* tiledb_ctx *object*

Description

Creates a tiledb_ctx object

Usage

```
tiledb_ctx(config = NULL, cached = TRUE)
```

Arguments

config	(optional) character vector of config parameter names, values
cached	(optional) logical switch to force new creation

Value

tiledb_ctx object

Examples

```
# default configuration
ctx <- tiledb_ctx()

# optionally set config parameters
ctx <- tiledb_ctx(c("sm.tile_cache_size" = "100"))
```

---

tiledb_ctx-class *An S4 class for a TileDB context*

Description

An S4 class for a TileDB context

Slots

ptr An external pointer to the underlying implementation

tiledb_ctx_set_default_tags *Sets default context tags*

Description

Sets default context tags

Usage

```
tiledb_ctx_set_default_tags(object)
```

Arguments

object	tiledb_ctx object

tiledb_ctx_set_tag *Sets a string:string "tag" on the Ctx*

Description

Sets a string:string "tag" on the Ctx

Usage

```
tiledb_ctx_set_tag(object, key, value)
```

Arguments

object	tiledb_ctx object
key	string
value	string

Examples

```
ctx <- tiledb_ctx(c("sm.tile_cache_size" = "10"))
cfg <- tiledb_ctx_set_tag(ctx, "tag", "value")
```

tiledb_delete_metadata *Delete a TileDB Array Metadata object given by key*

Description

Delete a TileDB Array Metadata object given by key

Usage

```
tiledb_delete_metadata(arr, key)
```

Arguments

arr	A TileDB Array object
key	A character value describing a metadata key

Value

A boolean indicating success

tiledb_dense *Constructs a tiledb_dense object backed by a persisted tiledb array uri*

Description

Constructs a tiledb_dense object backed by a persisted tiledb array uri

Usage

```
tiledb_dense(
  uri,
  query_type = c("READ", "WRITE"),
  as.data.frame = FALSE,
  attrs = character(),
  extended = FALSE,
  ctx = tiledb_get_context()
)
```

Arguments

uri	uri path to the tiledb dense array
query_type	optionally loads the array in "READ" or "WRITE" only modes.
as.data.frame	optional logical switch, defaults to "FALSE"
attrs	optional character vector to select attributes, default is empty implying all are selected
extended	optional logical switch selecting wide 'data.frame' format, defaults to "FALSE"
ctx	tiledb_ctx (optional)

Value

tiledb_dense array object

Planned Deprecation

We plan to deprecate the `tiledb_dense` array type in a future release. While exact timelines have not been finalised, it is advised to the `tiledb_array` for both *dense* and *sparse* arrays going forward.

`tiledb_dense-class` *An S4 class for a TileDB dense array*

Description

An S4 class for a TileDB dense array

Slots

`ctx` A TileDB context object

`uri` A character despription

`as.data.frame` A logical value

`attrs` A character vector

`extended` A logical value

`ptr` External pointer to the underlying implementation

Planned Deprecation

We plan to deprecate the `tiledb_dense` array type in a future release. While exact timelines have not been finalised, it is advised to the `tiledb_array` for both *dense* and *sparse* arrays going forward.

tiledb_dim *Contructs a* tiledb_dim *object*

Description

Contructs a tiledb_dim object

Usage

```
tiledb_dim(name, domain, tile, type, ctx = tiledb_get_context())
```

Arguments

name	The dimension name / label string. This argument is required.
domain	The dimension (inclusive) domain. The dimension's domain is defined by a (lower bound, upper bound) vector, and is usually either of type integer or double (i.e. numeric). For type, ASCII NULL is expected.
tile	The tile dimension tile extent. For type, ASCII NULL is expected.
type	The dimension TileDB datatype string
ctx	tiledb_ctx object (optional)

Value

tiledb_dim object

Examples

```
tiledb_dim(name = "d1", domain = c(1L, 10L), tile = 5L, type = "INT32")
```

tiledb_dim-class *An S4 class for a TileDB dimension object*

Description

An S4 class for a TileDB dimension object

Slots

ptr An external pointer to the underlying implementation

tiledb_domain *Constructs a* tiledb_domain *object*

Description

All tiledb_dim must be of the same TileDB type.

Usage

```
tiledb_domain(dims, ctx = tiledb_get_context())
```

Arguments

dims	list() of tiledb_dim objects
ctx	tiledb_ctx (optional)

Value

tiledb_domain

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 100L), type = "INT32"),
                              tiledb_dim("d2", c(1L, 50L), type = "INT32")))
```

tiledb_domain-class *An S4 class for a TileDB domain*

Description

An S4 class for a TileDB domain

Slots

ptr External pointer to the underlying implementation

`tiledb_domain_get_dimension_from_index` *Returns a Dimension indicated by index for the given TileDB Domain*

Description

Returns a Dimension indicated by index for the given TileDB Domain

Usage

```
tiledb_domain_get_dimension_from_index(domain, idx)
```

Arguments

`domain`	TileDB Domain object
`idx`	Integer index of the selected dimension

Value

TileDB Dimension object

`tiledb_domain_get_dimension_from_name` *Returns a Dimension indicated by name for the given TileDB Domain*

Description

Returns a Dimension indicated by name for the given TileDB Domain

Usage

```
tiledb_domain_get_dimension_from_name(domain, name)
```

Arguments

`domain`	TileDB Domain object
`name`	A character variable with a dimension name

Value

TileDB Dimension object

tiledb_domain_has_dimension *Check a domain for a given dimension name*

Description

Check a domain for a given dimension name

Usage

```
tiledb_domain_has_dimension(domain, name)
```

Arguments

domain	A domain of a TileDB Array schema
name	A character variable with a dimension name

Value

A boolean value indicating if the dimension exists in the domain

tiledb_filter *Constructs a* tiledb_filter *object*

Description

Available filters:

- "NONE"
- "GZIP"
- "ZSTD"
- "LZ4"
- "RLE"
- "BZIP2"
- "DOUBLE_DELTA"
- "BIT_WIDTH_REDUCTION"
- "BITSHUFFLE"
- "BYTESHUFFLE"
- "POSITIVE_DELTA"

Usage

```
tiledb_filter(name = "NONE", ctx = tiledb_get_context())
```

Arguments

name	(default "NONE") TileDB filter name string
ctx	tiledb_ctx object (optional)

Details

Valid compression options vary depending on the filter used, consult the TileDB docs for more information.

Value

tiledb_filter object

Examples

```
tiledb_filter("ZSTD")
```

---

tiledb_filter-class *An S4 class for a TileDB filter*

Description

An S4 class for a TileDB filter

Slots

ptr External pointer to the underlying implementation

---

tiledb_filter_get_option *Returns the filter's option*

Description

Returns the filter's option

Usage

```
tiledb_filter_get_option(object, option)
```

Arguments

object	tiledb_filter
option	string

Value

Integer value

Examples

```
c <- tiledb_filter("ZSTD")
tiledb_filter_set_option(c,"COMPRESSION_LEVEL", 5)
tiledb_filter_get_option(c, "COMPRESSION_LEVEL")
```

---

tiledb_filter_list *Constructs a* tiledb_filter_list *object*

Description

Constructs a tiledb_filter_list object

Usage

```
tiledb_filter_list(filters = c(), ctx = tiledb_get_context())
```

Arguments

filters	an optional list of one or more tiledb_filter_list objects
ctx	tiledb_ctx object (optional)

Value

tiledb_filter_list object

Examples

```
flt <- tiledb_filter("ZSTD")
tiledb_filter_set_option(flt, "COMPRESSION_LEVEL", 5)
filter_list <- tiledb_filter_list(c(flt))
filter_list
```

`tiledb_filter_list-class` *An S4 class for a TileDB filter list*

Description

An S4 class for a TileDB filter list

Slots

`ptr` An external pointer to the underlying implementation

`tiledb_filter_set_option` *Set the filter's option*

Description

Set the filter's option

Usage

```
tiledb_filter_set_option(object, option, value)
```

Arguments

`object`	tiledb_filter
`option`	string
`value`	int

Examples

```
c <- tiledb_filter("ZSTD")
tiledb_filter_set_option(c,"COMPRESSION_LEVEL", 5)
tiledb_filter_get_option(c, "COMPRESSION_LEVEL")
```

tiledb_filter_type *Returns the type of the filter used*

Description

Returns the type of the filter used

Usage

```
tiledb_filter_type(object)
```

Arguments

object	tiledb_filter

Value

TileDB filter type string

Examples

```
c <- tiledb_filter("ZSTD")
tiledb_filter_type(c)
```

tiledb_fragment_info *Constructs a* tiledb_fragment_info *object*

Description

Constructs a tiledb_fragment_info object

Usage

```
tiledb_fragment_info(uri, ctx = tiledb_get_context())
```

Arguments

uri	an character variable with the URI of the array for which fragment info is request
ctx	tiledb_ctx object (optional)

Value

tiledb_fragment_info object

tiledb_fragment_info-class

An S4 class for a TileDB fragment info object

Description

An S4 class for a TileDB fragment info object

Slots

`ptr` An external pointer to the underlying implementation

tiledb_fragment_info_dense

Return if a fragment info index is dense

Description

Return if a fragment info index is dense

Usage

```
tiledb_fragment_info_dense(object, fid)
```

Arguments

`object`	A TileDB fragment info object
`fid`	A fragment object index

Value

A logical value indicating if the fragment is dense

`tiledb_fragment_info_dump` *Dump the fragment info to console*

Description

Dump the fragment info to console

Usage

```
tiledb_fragment_info_dump(object)
```

Arguments

`object`	A TileDB fragment info object

Value

Nothing is returned, as a side effect the fragment info is displayed

`tiledb_fragment_info_get_cell_num` *Return a fragment info number of cells for a given fragment index*

Description

Return a fragment info number of cells for a given fragment index

Usage

```
tiledb_fragment_info_get_cell_num(object, fid)
```

Arguments

`object`	A TileDB fragment info object
`fid`	A fragment object index

Value

A numeric value with the number of cells

`tiledb_fragment_info_get_non_empty_domain_index`
Return a fragment info non-empty domain from index

Description

TODO: Rework with type information

Usage

```
tiledb_fragment_info_get_non_empty_domain_index(object, fid, did, typestr)
```

Arguments

`object`	A TileDB fragment info object
`fid`	A fragment object index
`did`	A domain index
`typestr`	An optional character variable describing the data type which will be accessed from the schema if missinh

Value

A TileDB Domain object

`tiledb_fragment_info_get_non_empty_domain_name`
Return a fragment info non-empty domain from name

Description

TODO: Rework with type information

Usage

```
tiledb_fragment_info_get_non_empty_domain_name(object, fid, dim_name, typestr)
```

Arguments

`object`	A TileDB fragment info object
`fid`	A fragment object index
`dim_name`	A character variable with the dimension name
`typestr`	An optional character variable describing the data type which will be accessed from the schema if missinh

Value

A TileDB Domain object

`tiledb_fragment_info_get_non_empty_domain_var_index`
Return a fragment info non-empty domain variable from index

Description

Return a fragment info non-empty domain variable from index

Usage

```
tiledb_fragment_info_get_non_empty_domain_var_index(object, fid, did)
```

Arguments

`object`	A TileDB fragment info object
`fid`	A fragment object index
`did`	A domain index

Value

A character vector with two elements

`tiledb_fragment_info_get_non_empty_domain_var_name`
Return a fragment info non-empty domain variable from name

Description

Return a fragment info non-empty domain variable from name

Usage

```
tiledb_fragment_info_get_non_empty_domain_var_name(object, fid, dim_name)
```

Arguments

`object`	A TileDB fragment info object
`fid`	A fragment object index
`dim_name`	A character variable with the dimension name

Value

A character vector with two elements

`tiledb_fragment_info_get_num` *Return a fragment info number of fragments*

Description

Return a fragment info number of fragments

Usage

```
tiledb_fragment_info_get_num(object)
```

Arguments

`object`	A TileDB fragment info object

Value

A numeric variable with the number of fragments

`tiledb_fragment_info_get_size` *Return a fragment info fragment size for a given fragment index*

Description

Return a fragment info fragment size for a given fragment index

Usage

```
tiledb_fragment_info_get_size(object, fid)
```

Arguments

`object`	A TileDB fragment info object
`fid`	A fragment object index

Value

A numeric variable with the number of fragments

`tiledb_fragment_info_get_timestamp_range` *Return a fragment info timestamp range for a given fragment index*

Description

Return a fragment info timestamp range for a given fragment index

Usage

```
tiledb_fragment_info_get_timestamp_range(object, fid)
```

Arguments

`object`	A TileDB fragment info object
`fid`	A fragment object index

Value

A Datetime vector with two elements for the range

`tiledb_fragment_info_get_to_vacuum_num` *Return the number of fragment info elements to be vacuumed*

Description

Return the number of fragment info elements to be vacuumed

Usage

```
tiledb_fragment_info_get_to_vacuum_num(object)
```

Arguments

`object`	A TileDB fragment info object

Value

A numeric value with the number of to be vacuumed fragments

tiledb_fragment_info_get_to_vacuum_uri

Return fragment info URI of the to be vacuumed index

Description

Return fragment info URI of the to be vacuumed index

Usage

```
tiledb_fragment_info_get_to_vacuum_uri(object, fid)
```

Arguments

object	A TileDB fragment info object
fid	A fragment object index

Value

A character variable with the URI of the be vacuumed index

tiledb_fragment_info_get_unconsolidated_metadata_num

Return fragment info number of unconsolidated metadata

Description

Return fragment info number of unconsolidated metadata

Usage

```
tiledb_fragment_info_get_unconsolidated_metadata_num(object)
```

Arguments

object	A TileDB fragment info object

Value

A numeric value with the number of unconsolidated metadata

tiledb_fragment_info_get_version

Return a fragment info version for a given fragment index

Description

Return a fragment info version for a given fragment index

Usage

```
tiledb_fragment_info_get_version(object, fid)
```

Arguments

object	A TileDB fragment info object
fid	A fragment object index

Value

A integer value value with the version

tiledb_fragment_info_has_consolidated_metadata

Return if a fragment info index has consolidated metadata

Description

Return if a fragment info index has consolidated metadata

Usage

```
tiledb_fragment_info_has_consolidated_metadata(object, fid)
```

Arguments

object	A TileDB fragment info object
fid	A fragment object index

Value

A logical value indicating consolidated metadata

tiledb_fragment_info_sparse *Return if a fragment info index is sparse*

Description

Return if a fragment info index is sparse

Usage

```
tiledb_fragment_info_sparse(object, fid)
```

Arguments

object	A TileDB fragment info object
fid	A fragment object index

Value

A logical value indicating if the fragment is sparse

tiledb_fragment_info_uri *Return a fragment info URI given its index*

Description

Return a fragment info URI given its index

Usage

```
tiledb_fragment_info_uri(object, fid)
```

Arguments

object	A TileDB fragment info object
fid	A fragment object index

Value

A character variable with URI

tiledb_get_all_metadata *Return a TileDB Array Metadata object given by key*

Description

Return a TileDB Array Metadata object given by key

Usage

```
tiledb_get_all_metadata(arr)
```

Arguments

arr	A TileDB Array object, or a character URI describing one

Value

A object stored in the Metadata under the given key

tiledb_get_context *Retrieve a TileDB context object from the package cache*

Description

Retrieve a TileDB context object from the package cache

Usage

```
tiledb_get_context()
```

Value

A TileDB context object

tiledb_get_metadata *Return a TileDB Array Metadata object given by key*

Description

Return a TileDB Array Metadata object given by key

Usage

```
tiledb_get_metadata(arr, key)
```

Arguments

arr	A TileDB Array object, or a character URI describing one
key	A character value describing a metadata key

Value

A object stored in the Metadata under the given key, or 'NULL' if none found.

tiledb_get_query_status *Retrieve the cached status of the last finalized query*

Description

This function accesses the status of the last query without requiring the query object.

Usage

```
tiledb_get_query_status()
```

Value

The status of the last query

tiledb_get_vfs *Retrieve a TileDB VFS object from the package environment and cache*

Description

Retrieve a TileDB VFS object from the package environment and cache

Usage

```
tiledb_get_vfs()
```

Value

A TileDB VFS object

tiledb_group_create *Creates a TileDB group object at given uri path*

Description

Creates a TileDB group object at given uri path

Usage

```
tiledb_group_create(uri, ctx = tiledb_get_context())
```

Arguments

uri	path which to create group
ctx	tiledb_ctx object (optional)

Value

uri of created group

Examples

```
## Not run:
pth <- tempdir()
tiledb_group_create(pth)
tiledb_object_type(pth)

## End(Not run)
```

tiledb_has_metadata *Test if TileDB Array has Metadata*

Description

Test if TileDB Array has Metadata

Usage

```
tiledb_has_metadata(arr, key)
```

Arguments

arr	A TileDB Array object
key	A character value describing a metadata key

Value

A logical value indicating if the given key exists in the metdata of the given array

tiledb_is_supported_fs *Query if a TileDB backend is supported*

Description

The scheme corresponds to the URI scheme for TileDB resouces.

Usage

```
tiledb_is_supported_fs(scheme, object = tiledb_get_context())
```

Arguments

scheme	URI string scheme ("file", "hdfs", "s3")
object	tiledb_ctx object

Details

Ex:

- {file}:///path/to/file
- {hdfs}:///path/to/file
- {s3}://hostname:port/path/to/file

Value

TRUE if tiledb backend is supported, FALSE otherwise

Examples

```
tiledb_is_supported_fs("file")
tiledb_is_supported_fs("s3")
```

---

tiledb_ndim,tiledb_array_schema-method

Return the number of dimensions associated with the `tiledb_array_schema`

Description

Return the number of dimensions associated with the `tiledb_array_schema`

Usage

```
## S4 method for signature 'tiledb_array_schema'
tiledb_ndim(object)
```

Arguments

`object`	tiledb_array_schema

Value

integer number of dimensions

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(1L, 10L), type = "INT32")))
sch <- tiledb_array_schema(dom, attrs = c(tiledb_attr("a1", type = "INT32"),
                                          tiledb_attr("a2", type = "FLOAT64")))
tiledb_ndim(sch)
```

tiledb_ndim,tiledb_dim-method

Returns the number of dimensions for a tiledb domain object

Description

Returns the number of dimensions for a tiledb domain object

Usage

```
## S4 method for signature 'tiledb_dim'
tiledb_ndim(object)
```

Arguments

object	tiledb_ndim object

Value

1L

Examples

```
d1 <- tiledb_dim("d1", c(1L, 10L), 10L)
tiledb_ndim(d1)
```

tiledb_ndim,tiledb_domain-method

Returns the number of dimensions of the tiledb_domain

Description

Returns the number of dimensions of the tiledb_domain

Usage

```
## S4 method for signature 'tiledb_domain'
tiledb_ndim(object)
```

Arguments

object	tiledb_domain

Value

integer number of dimensions

Examples

```
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(0.5, 100.0), type = "FLOAT64")))
tiledb_ndim(dom)
dom <- tiledb_domain(dims = c(tiledb_dim("d1", c(0.5, 100.0), type = "FLOAT64"),
                                    tiledb_dim("d2", c(0.5, 100.0), type = "FLOAT64")))
tiledb_ndim(dom)
```

---

tiledb_num_metadata *Return count of TileDB Array Metadata objects*

Description

Return count of TileDB Array Metadata objects

Usage

```
tiledb_num_metadata(arr)
```

Arguments

arr	A TileDB Array object, or a character URI describing one

Value

A integer variable with the number of Metadata objects

---

tiledb_object_ls *List TileDB resources at a given root URI path*

Description

List TileDB resources at a given root URI path

Usage

```
tiledb_object_ls(uri, filter = NULL, ctx = tiledb_get_context())
```

Arguments

uri	uri path to walk
filter	optional filtering argument, default is "NULL", currently unused
ctx	tiledb_ctx object (optional)

Value

a dataframe with object type, object uri string columns

---

tiledb_object_mv *Move a TileDB resource to new uri path*

Description

Raises an error if either uri is invalid, or the old uri resource is not a tiledb object

Usage

```
tiledb_object_mv(old_uri, new_uri, ctx = tiledb_get_context())
```

Arguments

old_uri	old uri of existing tiledb resource
new_uri	new uri to move tiledb resource
ctx	tiledb_ctx object (optional)

Value

new uri of moved tiledb resource

---

tiledb_object_rm *Removes a TileDB resource*

Description

Raises an error if the uri is invalid, or the uri resource is not a tiledb object

Usage

```
tiledb_object_rm(uri, ctx = tiledb_get_context())
```

Arguments

uri	path which to create group
ctx	tiledb_ctx object (optional)

Value

uri of removed TileDB resource

`tiledb_object_type` *Return the TileDB object type string of a TileDB resource*

Description

Object types:

- `"ARRAY"`, dense or sparse TileDB array
- `"GROUP"`, TileDB group
- ‘"INVALID"“, not a TileDB resource

Usage

```
tiledb_object_type(uri, ctx = tiledb_get_context())
```

Arguments

`uri`	path to TileDB resource
`ctx`	tiledb_ctx object (optional)

Value

TileDB object type string

`tiledb_object_walk` *Recursively discover TileDB resources at a given root URI path*

Description

Recursively discover TileDB resources at a given root URI path

Usage

```
tiledb_object_walk(uri, order = "PREORDER", ctx = tiledb_get_context())
```

Arguments

`uri`	root uri path to walk
`order`	(default "PREORDER") specify "POSTORDER" for "POSTORDER" traversal
`ctx`	tiledb_ctx object (optional)

Value

a dataframe with object type, object uri string columns

tiledb_put_metadata *Store an object in TileDB Array Metadata under given key*

Description

Store an object in TileDB Array Metadata under given key

Usage

```
tiledb_put_metadata(arr, key, val)
```

Arguments

arr	A TileDB Array object, or a character URI describing one
key	A character value describing a metadata key
val	An object to be store

Value

A boolean value indicating success

tiledb_query *Creates a 'tiledb_query' object*

Description

Creates a 'tiledb_query' object

Usage

```
tiledb_query(array, type = c("READ", "WRITE"), ctx = tiledb_get_context())
```

Arguments

array	A TileDB Array object
type	A character value that must be one of 'READ' or 'WRITE'
ctx	(optional) A TileDB Ctx object

Value

'tiledb_query' object

tiledb_query-class *An S4 class for a TileDB Query object*

Description

An S4 class for a TileDB Query object

Slots

`ptr` An external pointer to the underlying implementation

tiledb_query_add_range *Set a range for a given query*

Description

Set a range for a given query

Usage

```
tiledb_query_add_range(query, schema, attr, lowval, highval, stride = NULL)
```

Arguments

`query`	A TileDB Query object
`schema`	A TileDB Schema object
`attr`	An character variable with a dimension name for which the range is set
`lowval`	The lower value of the range to be set
`highval`	The higher value of the range to be set
`stride`	An optional stride value for the range to be set

Value

The query object, invisibly

`tiledb_query_add_range_with_type`
Set a range for a given query, also supplying type

Description

Set a range for a given query, also supplying type

Usage

```
tiledb_query_add_range_with_type(
  query,
  idx,
  datatype,
  lowval,
  highval,
  stride = NULL
)
```

Arguments

`query`	A TileDB Query object
`idx`	An integer index, zero based, of the dimensions
`datatype`	A character value containing the data type
`lowval`	The lower value of the range to be set
`highval`	The highre value of the range to be set
`stride`	An optional stride value for the range to be set

Value

The query object, invisibly

`tiledb_query_alloc_buffer_ptr_char`
Allocate a Query buffer for reading a character attribute

Description

Allocate a Query buffer for reading a character attribute

Usage

```
tiledb_query_alloc_buffer_ptr_char(sizeoffsets, sizedata)
```

Arguments

sizeoffsets	An optional value of the size of the offsets vector
sizedata	An optional value of the size of the data string

Value

An external pointer to the allocated buffer object

---

tiledb_query_alloc_buffer_ptr_char_subarray

Allocate a Query buffer for reading a character attribute using a subarray

---

Description

Note that this uses an API part that may be deprecated in the future.

Usage

```
tiledb_query_alloc_buffer_ptr_char_subarray(
  array,
  attr,
  subarray = NULL,
  sizeoffsets = 0,
  sizedata = 0
)
```

Arguments

array	A TileDB Array object
attr	A character value containing the attribute
subarray	A vector of length four describing the subarray required for dense arrays
sizeoffsets	An optional value of the size of the offsets vector
sizedata	An optional value of the size of the data string

Value

An external pointer to the allocated buffer object

tiledb_query_buffer_alloc_ptr

Allocate a Query buffer for a given type

Description

This function allocates a query buffer for the given data type.

Usage

```
tiledb_query_buffer_alloc_ptr(query, datatype, ncells)
```

Arguments

query	A TileDB Query object
datatype	A character value containing the data type
ncells	A number of elements (not bytes)

Value

An external pointer to the allocated buffer object

tiledb_query_condition

Creates a 'tiledb_query_condition' object

Description

Creates a 'tiledb_query_condition' object

Usage

```
tiledb_query_condition(ctx = tiledb_get_context())
```

Arguments

ctx	(optional) A TileDB Ctx object; if not supplied the default context object is retrieved

Value

A 'tiledb_query_condition' object

tiledb_query_condition-class

An S4 class for a TileDB QueryCondition object

Description

An S4 class for a TileDB QueryCondition object

Slots

`ptr` An external pointer to the underlying implementation

`init` A logical variable tracking if the query condition object has been initialized

tiledb_query_condition_combine

Combine two 'tiledb_query_condition' objects

Description

Combines two query condition object using a relatiional operator. Note that at present only 'AND' is supported.

Usage

```
tiledb_query_condition_combine(lhs, rhs, op)
```

Arguments

`lhs`	A 'tiledb_query_condition' object on the left-hand side of the relation
`rhs`	A 'tiledb_query_condition' object on the left-hand side of the relation
`op`	A character value with then relation, this must be one of 'AND', 'OR' or 'NOT'.

Value

The combined 'tiledb_query_condition' object

`tiledb_query_condition_init` *Initialize a 'tiledb_query_condition' object*

Description

Initializes (and possibly allocates) a query condition object using a triplet of attribute name, comparison value, and operator. Six types of conditions are supported, they all take a single scalar comparison argument and attribute to compare against. At present only integer or numeric attribute comparisons are implemented.

Usage

```
tiledb_query_condition_init(
  attr,
  value,
  dtype,
  op,
  qc = tiledb_query_condition()
)
```

Arguments

`attr`	A character value with the scheme attribute name
`value`	A scalar value that the attribute is compared against
`dtype`	A character value with the TileDB data type of the attribute column, for example 'FLOAT64' or 'INT32'
`op`	A character value with then comparison operation, this must be one of 'LT', 'LE', 'GT', 'GE', 'EQ', 'NE'.
`qc`	(optional) A 'tiledb_query_condition' object to be initialized by this call, if none is given a new one is allocated.

Value

The initialized 'tiledb_query_condition' object

`tiledb_query_create_buffer_ptr` *Allocate and populate a Query buffer for a given object of a given data type.*

Description

This function allocates a query buffer for the given data object of the given type and assigns the object content to the buffer.

Usage

```
tiledb_query_create_buffer_ptr(query, datatype, object)
```

Arguments

query	A TileDB Query object
datatype	A character value containing the data type
object	A vector object of the given type

Value

An external pointer to the allocated buffer object

---

tiledb_query_create_buffer_ptr_char

Allocate and populate a Query buffer for writing the given char vector

---

Description

Allocate and populate a Query buffer for writing the given char vector

Usage

```
tiledb_query_create_buffer_ptr_char(query, varvec)
```

Arguments

query	A TileDB Query object
varvec	A vector of strings

Value

An external pointer to the allocated buffer object

tiledb_query_export_buffer *Export Query Buffer to Pair of Arrow IO Pointers*

Description

This function exports the named buffer from a 'READ' query to two Arrow C pointers.

Usage

```
tiledb_query_export_buffer(query, name, ctx = tiledb_get_context())
```

Arguments

query	A TileDB Query object
name	A character variable identifying the buffer
ctx	tiledb_ctx object (optional)

Value

A two-element numeric vector where the two elements are pointers to the Arrow array and schema

tiledb_query_finalize *Finalize TileDB Query*

Description

Finalize TileDB Query

Usage

```
tiledb_query_finalize(query)
```

Arguments

query	A TileDB Query object

Value

A character value, either 'READ' or 'WRITE'

tiledb_query_get_buffer_char

Retrieve content from a Query character buffer

Description

This function uses a query buffer for a character attribute or dimension and returns its content.

Usage

```
tiledb_query_get_buffer_char(bufptr, sizeoffsets = 0, sizestring = 0)
```

Arguments

bufptr	An external pointer with a query buffer
sizeoffsets	An optional argument for the length of the internal offsets vector
sizestring	An optional argument for the length of the internal string

Value

An R object as resulting from the query

tiledb_query_get_buffer_ptr

Retrieve content from a Query buffer

Description

This function uses a query buffer and returns its content.

Usage

```
tiledb_query_get_buffer_ptr(bufptr)
```

Arguments

bufptr	An external pointer with a query buffer

Value

An R object as resulting from the query

tiledb_query_get_est_result_size

Retrieve the estimated result size for a query and attribute

Description

When reading from sparse arrays, one cannot know beforehand how big the result will be (unless one actually executes the query). This function offers a way to get the estimated result size for the given attribute. As TileDB does not actually execute the query, getting the estimated result is very fast.

Usage

```
tiledb_query_get_est_result_size(query, name)
```

Arguments

query	A TileDB Query object
name	A variable with an attribute name

Value

An estimate of the query result size

tiledb_query_get_est_result_size_var

Retrieve the estimated result size for a query and variable-sized attribute

Description

When reading variable-length attributes from either dense or sparse arrays, one cannot know beforehand how big the result will be (unless one actually executes the query). This function offers a way to get the estimated result size for the given attribute. As TileDB does not actually execute the query, getting the estimated result is very fast.

Usage

```
tiledb_query_get_est_result_size_var(query, name)
```

Arguments

query	A TileDB Query object
name	A variable with an attribute name

Value

An estimate of the query result size

`tiledb_query_get_fragment_num`
Retrieve the Number of Fragments for Query

Description

This function is only applicable to 'WRITE' queries.

Usage

```
tiledb_query_get_fragment_num(query)
```

Arguments

`query`	A TileDB Query object

Value

An integer with the number of fragments for the given query

`tiledb_query_get_fragment_timestamp_range`
Retrieve the timestamp range for a given Query Fragment

Description

This function is only applicable to 'WRITE' queries. The time resolution in TileDB is millseconds since the epoch so an R `Datetime` vector is returned.

Usage

```
tiledb_query_get_fragment_timestamp_range(query, idx)
```

Arguments

`query`	A TileDB Query object
`idx`	An integer (or numeric) index ranging from zero to the number of fragments minus 1

Value

A two-element datetime vector with the start and end time of the fragment write.

`tiledb_query_get_fragment_uri`
Retrieve the URI for a given Query Fragment

Description

This function is only applicable to 'WRITE' queries.

Usage

```
tiledb_query_get_fragment_uri(query, idx)
```

Arguments

`query`	A TileDB Query object
`idx`	An integer (or numeric) index ranging from zero to the number of fragments minus 1

Value

An character value with the fragment URI

`tiledb_query_get_layout`
Get TileDB Query layout

Description

Get TileDB Query layout

Usage

```
tiledb_query_get_layout(query)
```

Arguments

`query`	A TileDB Query object

Value

The TileDB Query layout as a string

tiledb_query_get_range

Retrieve the query range for a query dimension and range index

Description

Retrieve the query range for a query dimension and range index

Retrieve the query range for a variable-sized query dimension and range index

Usage

```
tiledb_query_get_range(query, dimidx, rngidx)

tiledb_query_get_range(query, dimidx, rngidx)
```

Arguments

query	A TileDB Query object
dimidx	An integer index selecting the dimension
rngidx	An integer index selection the given range for the dimension

Value

An integer vector with elements start, end and stride for the query range for the given dimension and range index

An string vector with elements start and end for the query range for the given dimension and range index

tiledb_query_get_range_num

Retrieve the number of ranges for a query dimension

Description

Retrieve the number of ranges for a query dimension

Usage

```
tiledb_query_get_range_num(query, idx)
```

Arguments

query	A TileDB Query object
idx	An integer index selecting the dimension

Value

An integer with the number of query range for the given dimensions

---

`tiledb_query_import_buffer`

Import to Query Buffer from Pair of Arrow IO Pointers

Description

This function imports to the named buffer for a 'WRITE' query from two Arrow C pointers.

Usage

```
tiledb_query_import_buffer(
  query,
  name,
  arrowpointers,
  ctx = tiledb_get_context()
)
```

Arguments

`query`	A TileDB Query object
`name`	A character variable identifying the buffer
`arrowpointers`	A two-element numeric vector with two pointers to an Arrow Array and Schema, respectively
`ctx`	tiledb_ctx object (optional)

Value

The update Query external pointer is returned

`tiledb_query_result_buffer_elements`

Get TileDB Query result buffer element size

Description

The underlying library functions returns a pair of values as a vector of length two. The first number is the number of element offsets for variable size attributes (and always zero for fixed-sized attributes and coordinates). The second is the number of elements in the data buffer. For variable-sized attributes the first number is the number of cells read (and hence the number of offsets), the second number is the number of elements in the data buffer.

Usage

```
tiledb_query_result_buffer_elements(query, attr)
```

Arguments

query	A TileDB Query object
attr	A character value containing the attribute

Details

As this function was first made available when only a scalar (corresponding to the second result) was returned, we still return that value.

Value

A integer with the number of elements in the results buffer for the given attribute

See Also

tiledb_query_result_buffer_elements_vec

---

tiledb_query_result_buffer_elements_vec

Get TileDB Query result buffer element size pair as vector

Description

The underlying library functions returns a pair of values as a vector of length two. The first number is the number of element offsets for variable size attributes (and always zero for fixed-sized attributes and coordinates). The second is the number of elements in the data buffer. For variable-sized attributes the first number is the number of cells read (and hence the number of offsets), the second number is the number of elements in the data buffer. In the case of a nullable attribute, a third element is returned with the size of the validity buffer.

Usage

```
tiledb_query_result_buffer_elements_vec(query, attr, nullable = FALSE)
```

Arguments

query	A TileDB Query object
attr	A character value containing the attribute
nullable	A logical variable that is 'TRUE' to signal that the attribute is nullable, and 'FALSE' otherwise

Value

A vector with the number of elements in the offsets buffer (and zero for fixed-size attribute or dimensions), the number elements in the results buffer for the given attribute, and (if nullable) a third element with the validity buffer size.

See Also

tiledb_query_result_buffer_elements

`tiledb_query_set_buffer`

Set TileDB Query buffer

Description

This function allocates query buffers directly from R vectors in case the types match: `integer`, `double`, `logical`. For more general types see `tiledb_query_buffer_alloc_ptr` and `tiledb_query_buffer_assign_ptr`

Usage

```
tiledb_query_set_buffer(query, attr, buffer)
```

Arguments

`query`	A TileDB Query object
`attr`	A character value containing the attribute
`buffer`	A vector providing the query buffer

Value

The modified query object, invisisibly

`tiledb_query_set_buffer_ptr`

Assigns to a Query buffer for a given attribute

Description

This function assigns a given query buffer to a query.

Usage

```
tiledb_query_set_buffer_ptr(query, attr, bufptr)
```

Arguments

query	A TileDB Query object
attr	A character value containing the attribute
bufptr	An external pointer with a query buffer

Value

The modified query object, invisibly

---

tiledb_query_set_buffer_ptr_char

Assign a buffer to a Query attribute

Description

Assign a buffer to a Query attribute

Usage

```
tiledb_query_set_buffer_ptr_char(query, attr, bufptr)
```

Arguments

query	A TileDB Query object
attr	A character value containing the attribute
bufptr	An external pointer with a query buffer

Value

The modified query object, invisibly

---

tiledb_query_set_condition

Set a query combination object for a query

Description

Set a query combination object for a query

Usage

```
tiledb_query_set_condition(query, qc)
```

Arguments

query	A TileDB Query object
qc	A TileDB Query Combination object

Value

The modified query object, invisibly

---

tiledb_query_set_layout

Set TileDB Query layout

Description

Set TileDB Query layout

Usage

```
tiledb_query_set_layout(
  query,
  layout = c("COL_MAJOR", "ROW_MAJOR", "GLOBAL_ORDER", "UNORDERED")
)
```

Arguments

query	A TileDB Query object
layout	A character variable with the layout; must be one of "COL_MAJOR", "ROW_MAJOR", "GLOBAL_ORDER", "UNORDERED")

Value

The modified query object, invisibly

---

tiledb_query_set_subarray

Set subarray for TileDB Query object

Description

Set subarray for TileDB Query object

Usage

```
tiledb_query_set_subarray(query, subarray, type)
```

Arguments

query	A TileDB Query object
subarray	A subarry vector object
type	An optional type as a character, if missing type is inferred from the vector.

Value

The modified query object, invisibly

---

tiledb_query_status *Get TileDB Query status*

Description

Get TileDB Query status

Usage

```
tiledb_query_status(query)
```

Arguments

query	A TileDB Query object

Value

A character value describing the query status

---

tiledb_query_submit *Submit TileDB Query*

Description

Note that the query object may need to be finalized via `tiledb_query_finalize`.

Usage

```
tiledb_query_submit(query)
```

Arguments

query	A TileDB Query object

Value

The modified query object, invisibly

tiledb_query_submit_async

Submit TileDB Query asynchronously without a callback returning immediately

Description

Note that the query object may need to be finalized via `tiledb_query_finalize`.

Usage

```
tiledb_query_submit_async(query)
```

Arguments

`query`	A TileDB Query object

Value

The modified query object, invisibly

tiledb_query_type

Return TileDB Query type

Description

Return TileDB Query type

Usage

```
tiledb_query_type(query)
```

Arguments

`query`	A TileDB Query object

Value

A character value, either 'READ' or 'WRITE'

tiledb_schema_get_names

Get all Dimension and Attribute Names

Description

Get all Dimension and Attribute Names

Usage

```
tiledb_schema_get_names(sch)
```

Arguments

sch	A TileDB Schema object

Value

A character vector of dimension and attribute names

tiledb_schema_get_types

Get all Dimension and Attribute Types

Description

Get all Dimension and Attribute Types

Usage

```
tiledb_schema_get_types(sch)
```

Arguments

sch	A TileDB Schema object

Value

A character vector of dimension and attribute data types

tiledb_set_context *Store a TileDB context object in the package cache*

Description

Store a TileDB context object in the package cache

Usage

```
tiledb_set_context(ctx)
```

Arguments

ctx	A TileDB context object

Value

NULL, invisibly. The function is invoked for the side-effect of storing the VFS object.

tiledb_set_vfs *Store a TileDB VFS object in the package environment*

Description

Store a TileDB VFS object in the package environment

Usage

```
tiledb_set_vfs(vfs)
```

Arguments

vfs	A TileDB VFS object

Value

NULL, invisibly. The function is invoked for the side-effect of storing the VFS object.

`tiledb_sparse` *Constructs a tiledb_sparse object backed by a persisted tiledb array uri*

Description

tiledb_sparse returns a list of coordinates and attributes vectors for reads

Usage

```
tiledb_sparse(
  uri,
  query_type = c("READ", "WRITE"),
  as.data.frame = FALSE,
  attrs = character(),
  extended = TRUE,
  ctx = tiledb_get_context()
)
```

Arguments

`uri`	uri path to the tiledb dense array
`query_type`	optionally loads the array in "READ" or "WRITE" only modes.
`as.data.frame`	optional logical switch, defaults to "FALSE"
`attrs`	optional character vector to select attributes, default is empty implying all are selected
`extended`	optional logical switch selecting wide 'data.frame' format, defaults to "TRUE"
`ctx`	tiledb_ctx (optional)

Value

tiledb_sparse array object

Planned Deprecation

We plan to deprecate the `tiledb_sparse` array type in a future release. While exact timelines have not been finalised, it is advised to the `tiledb_array` for both *dense* and *sparse* arrays going forward.

tiledb_sparse-class *An S4 class for a TileDB sparse array*

Description

An S4 class for a TileDB sparse array

Slots

`ctx` A TileDB context object

`uri` A character despription

`as.data.frame` A logical value

`attrs` A character vector

`extended` A logical value

`ptr` External pointer to the underlying implementation

Planned Deprecation

We plan to deprecate the `tiledb_sparse` array type in a future release. While exact timelines have not been finalised, it is advised to the `tiledb_array` for both *dense* and *sparse* arrays going forward.

tiledb_stats_disable *Disable internal TileDB statistics counters*

Description

This function ends the collection of internal statistics.

Usage

```
tiledb_stats_disable()
```

tiledb_stats_dump *Dumps internal TileDB statistics to file*

Description

Dumps internal TileDB statistics to file

Usage

```
tiledb_stats_dump(path)
```

Arguments

path	Character variable with path to stats file; if the empty string is passed then the result is displayed on stdout.

Examples

```
pth <- tempfile()
tiledb_stats_dump(pth)
cat(readLines(pth)[1:10], sep = "\n")
```

tiledb_stats_enable *Enable internal TileDB statistics counters*

Description

This function starts the collection of internal statistics.

Usage

```
tiledb_stats_enable()
```

tiledb_stats_print *Print internal TileDB statistics*

Description

This function is a convenience wrapper for `tiledb_stats_dump`.

Usage

```
tiledb_stats_print()
```

tiledb_stats_raw_dump *Dumps internal TileDB statistics as JSON to file*

Description

This function requires TileDB Embedded 2.0.3 or later.

Usage

```
tiledb_stats_raw_dump(path)
```

Arguments

path	Character variable with path to stats file; if the empty string is passed then the result is displayed on stdout.

Examples

```
if (tiledb_version(TRUE) >= "2.0.3") {
  pth <- tempfile()
  tiledb_stats_raw_dump(pth)
  cat(readLines(pth)[1:10], sep = "\n")
}
```

tiledb_stats_raw_get *Gets internal TileDB statistics as JSON string*

Description

This function is a convenience wrapper for tiledb_stats_raw_dump and returns the result as a JSON string. It required TileDB Embedded 2.0.3 or later.

Usage

```
tiledb_stats_raw_get()
```

tiledb_stats_raw_print *Print internal TileDB statistics as JSON*

Description

This function is a convenience wrapper for `tiledb_stats_raw_dump`. It required TileDB Embedded 2.0.3 or later.

Usage

```
tiledb_stats_raw_print()
```

tiledb_stats_reset *Reset internal TileDB statistics counters*

Description

This function resets the counters for internal statistics.

Usage

```
tiledb_stats_reset()
```

tiledb_subarray *Query a array using a subarray vector*

Description

tiledb_subarray returns a results of query

Usage

```
tiledb_subarray(A, subarray_vector, attrs = c())
```

Arguments

`A`	tiledb_sparse or tiledb_dense
`subarray_vector`	subarray to query
`attrs`	list of attributes to query

Value

list of attributes being returned with query results

tiledb_version *The version of the libtiledb library*

Description

The version of the libtiledb library

Usage

```
tiledb_version(compact = FALSE)
```

Arguments

compact	Logical value indicating wheter a compact package_version object should be returned

Value

An named int vector c(major, minor, patch), or if select, a package_version object

Examples

```
tiledb_version()
tiledb_version(compact = TRUE)
```

tiledb_vfs *Creates a* tiledb_vfs *object*

Description

Creates a tiledb_vfs object

Usage

```
tiledb_vfs(config = NULL, ctx = tiledb_get_context())
```

Arguments

config	(optional) character vector of config parameter names, values
ctx	(optional) A TileDB Ctx object

Value

The tiledb_vfs object

Examples

```
# default configuration
vfs <- tiledb_vfs()
```

`tiledb_vfs-class` *An S4 class for a TileDB VFS object*

Description

An S4 class for a TileDB VFS object

Slots

`ptr` An external pointer to the underlying implementation

`tiledb_vfs_close` *Close a TileDB VFS Filehandle*

Description

Close a TileDB VFS Filehandle

Usage

```
tiledb_vfs_close(fh, ctx = tiledb_get_context())
```

Arguments

`fh`	A TileDB VFS Filehandle external pointer as returned from `tiledb_vfs_open`
`ctx`	(optional) A TileDB Ctx object

Value

The result of the close operation is returned.

tiledb_vfs_create_bucket *Create a VFS Bucket*

Description

Create a VFS Bucket

Usage

```
tiledb_vfs_create_bucket(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a cloud bucket
vfs	A TileDB VFS object; default is to use a cached value.

Value

The uri value

tiledb_vfs_create_dir *Create a VFS Directory*

Description

Create a VFS Directory

Usage

```
tiledb_vfs_create_dir(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a diretory path
vfs	A TileDB VFS object; default is to use a cached value.

Value

The uri value of the created directory

tiledb_vfs_dir_size *Return VFS Directory Size*

Description

Return VFS Directory Size

Usage

```
tiledb_vfs_dir_size(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a file path
vfs	A TileDB VFS object; default is to use a cached value.

Value

The size of the directory

tiledb_vfs_empty_bucket *Empty a VFS Bucket*

Description

Empty a VFS Bucket

Usage

```
tiledb_vfs_empty_bucket(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a cloud bucket
vfs	A TileDB VFS object; default is to use a cached value.

Value

The URI value that was emptied

tiledb_vfs_file_size *Return VFS File Size*

Description

Return VFS File Size

Usage

```
tiledb_vfs_file_size(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a file path
vfs	A TileDB VFS object; default is to use a cached value.

Value

The size of the file

tiledb_vfs_is_bucket *Check for VFS Bucket*

Description

Check for VFS Bucket

Usage

```
tiledb_vfs_is_bucket(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a cloud bucket
vfs	A TileDB VFS object; default is to use a cached value.

Value

A boolean value indicating if it is a valid bucket

Examples

```
## Not run:
cfg <- tiledb_config()
cfg["vfs.s3.region"] <- "us-west-1"
ctx <- tiledb_ctx(cfg)
vfs <- tiledb_vfs()
tiledb_vfs_is_bucket(vfs, "s3://tiledb-public-us-west-1/test-array-4x4")

## End(Not run)
```

---

tiledb_vfs_is_dir *Test for VFS Directory*

Description

Test for VFS Directory

Usage

```
tiledb_vfs_is_dir(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a diretory path
vfs	A TileDB VFS object; default is to use a cached value.

Value

A boolean value indicating if it is a directory

---

tiledb_vfs_is_empty_bucket *Check for empty VFS Bucket*

Description

Check for empty VFS Bucket

Usage

```
tiledb_vfs_is_empty_bucket(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a cloud bucket
vfs	A TileDB VFS object; default is to use a cached value.

Value

A boolean value indicating if it is an empty bucket

Examples

```
## Not run:
cfg <- tiledb_config()
cfg["vfs.s3.region"] <- "us-west-1"
ctx <- tiledb_ctx(cfg)
vfs <- tiledb_vfs()
tiledb_vfs_is_empty_bucket(vfs, "s3://tiledb-public-us-west-1/test-array-4x4")

## End(Not run)
```

---

tiledb_vfs_is_file *Test for VFS File*

Description

Test for VFS File

Usage

```
tiledb_vfs_is_file(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a file path
vfs	A TileDB VFS object; default is to use a cached value.

Value

A boolean value indicating if it is a file

---

tiledb_vfs_ls *Return VFS Directory Listing*

Description

Return VFS Directory Listing

Usage

```
tiledb_vfs_ls(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a file path
vfs	A TileDB VFS object; default is to use a cached value.

Value

The content of the directory, non-recursive

---

tiledb_vfs_move_dir *Move (or rename) a VFS Directory*

Description

Move (or rename) a VFS Directory

Usage

```
tiledb_vfs_move_dir(olduri, newuri, vfs = tiledb_get_vfs())
```

Arguments

olduri	Character variable with an existing URI describing a directory path
newuri	Character variable with a new desired URI directory path
vfs	A TileDB VFS object; default is to use a cached value.

Value

The newuri value of the moved directory

---

tiledb_vfs_move_file *Move (or rename) a VFS File*

Description

Move (or rename) a VFS File

Usage

```
tiledb_vfs_move_file(olduri, newuri, vfs = tiledb_get_vfs())
```

Arguments

olduri	Character variable with an existing URI describing a file path
newuri	Character variable with a new desired URI file path
vfs	A TileDB VFS object; default is to use a cached value.

Value

The newuri value of the moved file

`tiledb_vfs_open` *Open a TileDB VFS Filehandle for reading or writing*

Description

Open a TileDB VFS Filehandle for reading or writing

Usage

```
tiledb_vfs_open(
  binfile,
  mode = c("READ", "WRITE", "APPEND"),
  vfs = tiledb_get_vfs(),
  ctx = tiledb_get_context()
)
```

Arguments

`binfile`	A character variable describing the (binary) file to be opened
`mode`	A character variable with value 'READ', 'WRITE' or 'APPEND'
`vfs`	A TileDB VFS object; default is to use a cached value.
`ctx`	(optional) A TileDB Ctx object

Value

A TileDB VFS Filehandle object (as an external pointer)

`tiledb_vfs_read` *Read from a TileDB VFS Filehandle*

Description

This interface currently defaults to reading an integer vector. This is suitable for R objects as a raw vector used for (de)serialization can be mapped easily to an integer vector. It is also possible to `memcpy` to the contiguous memory of an integer vector should other (non-R) data be transferred.

Usage

```
tiledb_vfs_read(fh, offset, nbytes, ctx = tiledb_get_context())
```

Arguments

fh	A TileDB VFS Filehandle external pointer as returned from tiledb_vfs_open
offset	A scalar integer64 value with the byte offset from the beginning of the file with a of zero.
nbytes	A scalar integer64 value with the number of bytes to be read.
ctx	(optional) A TileDB Ctx object

Value

The binary file content is returned as an integer vector.

---

tiledb_vfs_remove_bucket *Remove a VFS Bucket*

Description

Remove a VFS Bucket

Usage

```
tiledb_vfs_remove_bucket(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a cloud bucket
vfs	A TileDB VFS object; default is to use a cached value.

Value

The uri value

---

tiledb_vfs_remove_dir *Remove a VFS Directory*

Description

Remove a VFS Directory

Usage

```
tiledb_vfs_remove_dir(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a diretory path
vfs	A TileDB VFS object; default is to use a cached value.

Value

The uri value of the removed directory

---

tiledb_vfs_remove_file *Remove a VFS File*

Description

Remove a VFS File

Usage

```
tiledb_vfs_remove_file(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a file path
vfs	A TileDB VFS object; default is to use a cached value.

Value

The uri value of the removed file

---

tiledb_vfs_sync *Sync a TileDB VFS Filehandle*

Description

Sync a TileDB VFS Filehandle

Usage

```
tiledb_vfs_sync(fh, ctx = tiledb_get_context())
```

Arguments

fh	A TileDB VFS Filehandle external pointer as returned from `tiledb_vfs_open`
ctx	(optional) A TileDB Ctx object

Value

The result of the sync operation is returned.

tiledb_vfs_touch *Touch a VFS URI Resource*

Description

Touch a VFS URI Resource

Usage

```
tiledb_vfs_touch(uri, vfs = tiledb_get_vfs())
```

Arguments

uri	Character variable with a URI describing a bucket, file or directory
vfs	A TileDB VFS object; default is to use a cached value.

Value

The uri value

tiledb_vfs_write *Write to a TileDB VFS Filehandle*

Description

This interface currently defaults to using an integer vector. This is suitable for R objects as the raw vector result from serialization can be mapped easily to an integer vector. It is also possible to memcpy to the contiguous memory of an integer vector should other (non-R) data be transferred.

Usage

```
tiledb_vfs_write(fh, vec, ctx = tiledb_get_context())
```

Arguments

fh	A TileDB VFS Filehandle external pointer as returned from tiledb_vfs_open
vec	An integer vector of content to be written
ctx	(optional) A TileDB Ctx object

Value

The result of the write operation is returned.

`tile_order,tiledb_array_schema-method`
Returns the tile layout string associated with the `tiledb_array_schema`

Description

Returns the tile layout string associated with the `tiledb_array_schema`

Usage

```
## S4 method for signature 'tiledb_array_schema'
tile_order(object)
```

Arguments

`object`	tiledb object

`[,tiledb_array,ANY-method`
Returns a TileDB array, allowing for specific subset ranges.

Description

Heterogenous domains are supported, including timestamps and characters.

Usage

```
## S4 method for signature 'tiledb_array,ANY'
x[i, j, ..., drop = FALSE]
```

Arguments

`x`	tiledb_array object
`i`	optional row index expression which can be a list in which case minimum and maximum of each list element determine a range; multiple list elements can be used to supply multiple ranges.
`j`	optional column index expression which can be a list in which case minimum and maximum of each list element determine a range; multiple list elements can be used to supply multiple ranges.
`...`	Extra parameters for method signature, currently unused.
`drop`	Optional logical switch to drop dimensions, default FALSE, currently unused.

Details

This function may still still change; the current implementation should be considered as an initial draft.

Value

The resulting elements in the selected format

[,tiledb_config,ANY-method

Gets a config parameter value

Description

Gets a config parameter value

Usage

```
## S4 method for signature 'tiledb_config,ANY'
x[i, j, ..., drop = FALSE]
```

Arguments

x	tiledb_config object
i	parameter key string
j	parameter key string, currently unused.
...	Extra parameter for method signature, currently unused.
drop	Optional logical switch to drop dimensions, default FALSE, currently unused.

Value

a config string value if parameter exists, else NA

Examples

```
cfg <- tiledb_config()
cfg["sm.tile_cache_size"]
cfg["does_not_exist"]
```

`[,tiledb_dense,ANY-method`

Gets a dense array value

Description

Gets a dense array value

Usage

```
## S4 method for signature 'tiledb_dense,ANY'
x[i, j, ..., drop = FALSE]
```

Arguments

x	dense array object
i	parameter key string
j	parameter key string, currently unused.
...	Extra parameter for method signature, currently unused.
drop	Optional logical switch to drop dimensions, default FALSE, currently unused.

Value

An element from a dense array

`[,tiledb_filter_list,ANY-method`

Returns the filter at given index

Description

Returns the filter at given index

Usage

```
## S4 method for signature 'tiledb_filter_list,ANY'
x[i, j, ..., drop = FALSE]
```

Arguments

x	`tiledb_config` object
i	parameter key string
j	parameter key string, currently unused.
...	Extra parameter for method signature, currently unused.
drop	Optional logical switch to drop dimensions, default false.

Value

object tiledb_filter

Examples

```
flt <- tiledb_filter("ZSTD")
tiledb_filter_set_option(flt, "COMPRESSION_LEVEL", 5)
filter_list <- tiledb_filter_list(c(flt))
filter_list[0]
```

[,tiledb_sparse,ANY-method

Gets a sparse array value

Description

Gets a sparse array value

Usage

```
## S4 method for signature 'tiledb_sparse,ANY'
x[i, j, ..., drop = FALSE]
```

Arguments

x	sparse array object
i	parameter key string
j	parameter key string, currently unused.
...	Extra parameter for method signature, currently unused.
drop	Optional logical switch to drop dimensions, default FALSE, currently unused.

Value

An element from the sparse array

`[<-,tiledb_array,ANY,ANY,ANY-method`

Sets a tiledb array value or value range

Description

This function assigns a right-hand side object, typically a data.frame or something that can be coerced to a data.frame, to a tiledb array.

Usage

```
## S4 replacement method for signature 'tiledb_array,ANY,ANY,ANY'
x[i, j, ...] <- value
```

Arguments

x	sparse or dense TileDB array object
i	parameter row index
j	parameter column index
...	Extra parameter for method signature, currently unused.
value	The value being assigned

Details

For sparse matrices, row and column indices can either be supplied as part of the left-hand side object, or as part of the data.frame provided approrpiate column names.

This function may still still change; the current implementation should be considered as an initial draft.

Value

The modified object

Examples

```
## Not run:
uri <- "quickstart_sparse"      ## as created by the other example
arr <- tiledb_array(uri)        ## open array
df <- arr[]                     ## read current content
## First approach: matching data.frame with appriate row and column
newdf <- data.frame(rows=c(1,2,2), cols=c(1,3,4), a=df$a+100)
## Second approach: supply indices explicitly
arr[c(1,2), c(1,3)] <- c(42,43) ## two values
arr[2, 4] <- 88                 ## or just one

## End(Not run)
```

[<-,tiledb_config,ANY,ANY,ANY-method

Sets a config parameter value

Description

Sets a config parameter value

Usage

```
## S4 replacement method for signature 'tiledb_config,ANY,ANY,ANY'
x[i, j] <- value
```

Arguments

x	tiledb_config object
i	parameter key string
j	parameter key string
value	value to set, will be converted into a stringa

Value

updated tiledb_config object

Examples

```
cfg <- tiledb_config()
cfg["sm.tile_cache_size"]

# set tile cache size to custom value
cfg["sm.tile_cache_size"] <- 100
cfg["sm.tile_cache_size"]
```

[<-,tiledb_dense,ANY,ANY,ANY-method

Sets a dense array value

Description

Sets a dense array value

Usage

```
## S4 replacement method for signature 'tiledb_dense,ANY,ANY,ANY'
x[i, j, ...] <- value
```

Arguments

x	dense array object
i	parameter key string
j	parameter key string, currently unused.
...	Extra parameter for method signature, currently unused.
value	The value being assigned

Value

The modified object

[<-,tiledb_sparse,ANY,ANY,ANY-method

Sets a sparse array value

Description

Sets a sparse array value

Usage

```
## S4 replacement method for signature 'tiledb_sparse,ANY,ANY,ANY'
x[i, j, ...] <- value
```

Arguments

x	sparse array object
i	parameter key string
j	parameter key string, currently unused.
...	Extra parameter for method signature, currently unused.
value	The value being assigned

Value

The modified object

Index