

# Package ‘TileDBArray’

October 25, 2025

**Version** 1.19.1

**Date** 2025-06-19

**Title** Using TileDB as a DelayedArray Backend

**Description** Implements a DelayedArray backend for reading and writing dense or sparse arrays in the TileDB format. The resulting TileDBArrays are compatible with all Bioconductor pipelines that can accept DelayedArray instances.

**License** MIT + file LICENSE

**Depends** SparseArray (>= 1.5.20), DelayedArray (>= 0.31.7)

**Imports** methods, tiledb, S4Vectors

**Suggests** knitr, Matrix, rmarkdown, BiocStyle, BiocParallel, testthat

**biocViews** DataRepresentation, Infrastructure, Software

**VignetteBuilder** knitr

**BugReports** <https://github.com/LTLA/TileDBArray>

**URL** <https://github.com/LTLA/TileDBArray>

**RoxygenNote** 7.3.2

**Encoding** UTF-8

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

**git\_branch** devel

**git\_last\_commit** 5a6552a

**git\_last\_commit\_date** 2025-06-19

**Repository** Bioconductor 3.22

**Date/Publication** 2025-10-24

**Author** Aaron Lun [aut, cre],  
Genentech, Inc. [cph]

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

## Contents

|                                 |   |
|---------------------------------|---|
| TileDBArray . . . . .           | 2 |
| TileDBArray-globals . . . . .   | 3 |
| TileDBArray-pkg . . . . .       | 5 |
| TileDBRealizationSink . . . . . | 5 |

**Index****8**


---

|             |                              |
|-------------|------------------------------|
| TileDBArray | <i>Delayed TileDB arrays</i> |
|-------------|------------------------------|

---

**Description**

The TileDBArray class provides a [DelayedArray](#) backend for TileDB arrays (sparse and dense).

**Constructing a TileDBArray**

TileDBArray(x, attr) returns a TileDBArray object given:

- x, a string containing a URI to a TileDB backend, most typically a path to a directory.
- attr, a string specifying the attribute to represent in the array. Defaults to the first attribute.

Alternatively, x can be a TileDBArraySeed object, in which case attr is ignored.

TileDBArraySeed(x, attr) returns a TileDBArraySeed with the same arguments as described for TileDBArray. If x is already a TileDBArraySeed, it is returned directly without further modification.

[DelayedArray](#)(x) returns a TileDBArray object given x, a TileDBArraySeed.

In all cases, two-dimensional arrays will automatically generate a TileDBMatrix, a subclass of the TileDBArray.

**Available operations**

[extract\\_array](#)(x, index) will return an ordinary array containing values from the TileDBArraySeed x, subsetting to the indices specified in index. The latter should be a list of length equal to the number of dimensions in x, where each entry is an integer vector or NULL (in which case the entirety of the dimension is used).

[extract\\_sparse\\_array](#)(x, index) will return a [COO\\_SparseArray](#) representing the subset of x corresponding to the indices in index. The latter should be a list of the same structure as described for [extract\\_array](#).

[type](#)(x) will return a string containing the type of the TileDBArraySeed object x. Currently, only "integer", "logical" and "double"-precision is supported.

[is\\_sparse](#)(x) will return a logical scalar indicating whether the TileDBArraySeed x uses a sparse format in the TileDB backend.

[path](#)(x) will return a string containing the path to the TileDB backend directory.

[chunkdim](#)(x) will return an integer vector containing the tile extent in each dimension. This will be used as the chunk dimensions in methods like [chunkGrid](#).

All of the operations described above are also equally applicable to TileDBArray objects, as their methods simply delegate to those of the TileDBArraySeed.

All operations supported by [DelayedArray](#) objects are also available for TileDBArray objects.

**Author(s)**

Aaron Lun

**Examples**

```

data <- matrix(rpois(10000, 5), nrow=100, ncol=100)
B <- as(data, "TileDBArray")
B

# Apply typical DelayedArray operations:
as.matrix(B[1:10,1:10])
B %%% runif(ncol(B))

# This also works for sparse arrays:
sdata <- Matrix::rsparsematrix(nrow=100, ncol=100, density=0.1)
C <- as(sdata, "TileDBArray")
C

```

---

|                     |                                   |
|---------------------|-----------------------------------|
| TileDBArray-globals | <i>TileDBArray global options</i> |
|---------------------|-----------------------------------|

---

**Description**

Global options for writing TileDBArray backends, intended for parameters that cannot be automatically derived from the data.

**Usage**

```

getTileDBPath()

setTileDBPath(path = NULL)

getTileDBAttr()

setTileDBAttr(attr = NULL)

getTileDBDimType()

setTileDBDimType(dimtype = NULL)

getTileDBExtent()

setTileDBExtent(extent = NULL)

getTileDBContext()

setTileDBContext(context = NULL)

getTileDBCellOrder()

setTileDBCellOrder(cellorder = NULL)

getTileDBTileOrder()

```

```
setTileDBTileOrder(tileorder = NULL)
```

```
getTileDBCcapacity()
```

```
setTileDBCcapacity(capacity = NULL)
```

### Arguments

|           |                                                                                                                                                                                                                          |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path      | String containing a path to a TileDB backend.                                                                                                                                                                            |
| attr      | String containing the name of a TileDB attribute.                                                                                                                                                                        |
| dimtype   | String specifying the TileDB datatype to use for the dimensions.                                                                                                                                                         |
| extent    | Integer scalar specifying the tile extent for all dimensions. Alternatively, an integer vector of length equal to the number of dimensions, specifying a different extent for each dimension in the array to be created. |
| context   | A TileDB context object, see <a href="#">tiledb_ctx</a> for an example.                                                                                                                                                  |
| cellorder | String specifying the desired cell order.                                                                                                                                                                                |
| tileorder | String specifying the desired tile order.                                                                                                                                                                                |
| capacity  | Integer scalar specifying the data tile capacity for sparse arrays.                                                                                                                                                      |

### Value

All of the getter functions return the current global value, or a default value if the former is NULL:

- path defaults to a temporary file in [tempdir](#).
- attr defaults to "x".
- dimtype defaults to "INT32".
- extent defaults to 100L.
- cellorder defaults to "COL\_MAJOR".
- tileorder defaults to "COL\_MAJOR".
- capacity defaults to 10000L.
- context defaults to the value of [tiledb\\_ctx\(\)](#).

All setter functions change the global value and return NULL invisibly.

### Author(s)

Aaron Lun

### See Also

[writeTileDBArray](#), where these functions are most often used.

### Examples

```
setTileDBPath("my_local_dir")
getTileDBPath()
```

---

TileDBArray-pkg

The **TileDBArray** package

---

## Description

Implements the TileDB framework as a [DelayedArray](#) backend, with read and write functionality for both dense and sparse arrays. Currently only integer, logical and double-precision values are supported.

## Author(s)

Aaron Lun

---

TileDBRealizationSink    *Write arrays to TileDB*


---

## Description

Write array data to a TileDB backend via **DelayedArray**'s [RealizationSink](#) machinery.

## Writing a TileDBArray

```
TileDBRealizationSink(
  dim,
  dimnames=NULL,
  type="double",
  path=getTileDBPath(),
  attr=getTileDBAttr(),
  storagetype=NULL,
  dimtype=getTileDBDimType(),
  sparse=FALSE,
  extent=getTileDBExtent(),
  offset=1L,
  cellorder=getTileDBCellOrder(),
  tileorder=getTileDBTileOrder(),
  capacity=getTileDBCapacity(),
  context=getTileDBContext()
)
```

returns a TileDBRealizationSink object that can be used to write content to a TileDB backend. It accepts the following arguments:

- `dim`, an integer vector (usually of length 2) to specify the array dimensions.
- `dimnames`, a list of length equal to `dim`, containing character vectors with names for each dimension. Defaults to `NULL`, i.e., no `dimnames`.
- `type`, a string specifying the R data type for the newly written array. Currently only "double", "integer" and "logical" arrays are supported.
- `path`, a string containing the location of the new TileDB backend.
- `attr`, a string specifying the name of the attribute to store.

- `storage_type`, a string specifying the TileDB data type for the attribute, e.g., "UINT8", "FLOAT32". If NULL, this is automatically determined from type using `r_to_tiledb_type`.
- `dim_type`, a string specifying the TileDB data type for the dimension.
- `sparse`, a logical scalar indicating whether the array should be stored in sparse form.
- `extent`, an integer scalar (or vector of length equal to `dim`) specifying the tile extent for each dimension. Larger values improve compression at the cost of unnecessary data extraction during reads.
- `offset`, an integer scalar (or vector of length equal to `dim`) specifying the starting offset for each dimension's domain.
- `cell_order`, a string specifying the ordering of cells within each tile.
- `tile_order`, a string specifying the ordering of tiles across the array.
- `capacity`, an integer scalar specifying the size of each data tile in the sparse case.
- `context` is the TileDB context, defaulting to the output of `tiledb_ctx()`.

`writeTileDBArray(x, sparse=is_sparse(x), ...)` writes the matrix-like object `x` to a TileDB backend, returning a `TileDBArray` object referring to that backend. Appropriate values for `dim`, `dimnames` and `type` are determined automatically from `x` itself. All other arguments described for `TileDBRealizationSink` can be passed into `...` to configure the representation.

### Coercing to a TileDBArray

`as(x, "TileDBArray")` will coerce a matrix-like object `x` to a `TileDBArray` object.  
`as(x, "TileDBArraySeed")` will coerce a matrix-like object `x` to a `TileDBArraySeed` object.  
`as(x, "TileDBMatrix")` will coerce a matrix-like object `x` to a `TileDBArraySeed` object.  
`as(x, "TileDBArray")` will coerce a `TileDBRealizationSink` `x` to a `TileDBArray` object.  
`as(x, "TileDBArraySeed")` will coerce a `TileDBRealizationSink` `x` to a `TileDBArraySeed` object.  
`as(x, "DelayedArray")` will coerce a `TileDBRealizationSink` `x` to a `TileDBArray` object.

### Sink internals

`write_block(sink, viewport, block)` will write the subarray `block` to the `TileDBRealizationSink` `sink` at the specified `viewport`, returning `sink` upon completion. See `write_block` in **DelayedArray** for more details.

`type(x)` will return a string specifying the type of the `TileDBRealizationSink` `x`.

### Examples

```
X <- matrix(rnorm(100000), ncol=200)
path <- tempfile()
out <- writeTileDBArray(X, path=path)

# Works for integer matrices.
Xi <- matrix(rpois(100000, 2), ncol=200)
pathi <- tempfile()
outi <- writeTileDBArray(Xi, path=pathi)

# Works for logical matrices.
Xl <- matrix(rpois(100000, 0.5) > 0, ncol=200)
pathl <- tempfile()
outl <- writeTileDBArray(Xl, path=pathl)
```

```
# Works for sparse numeric matrices.
Y <- Matrix::rsparsematrix(1000, 1000, density=0.01)
path2 <- tempfile()
out2 <- writeTileDBArray(Y, path=path2)

# And for sparse logical matrices.
path2l <- tempfile()
out2l <- writeTileDBArray(Y > 0, path=path2l)

# Works for dimnames.
rownames(X) <- sprintf("GENE_%i", seq_len(nrow(X)))
path3 <- tempfile()
out3 <- writeTileDBArray(X, path=path3)
```

# Index

[chunkdim](#), [2](#)  
[chunkdim](#), [TileDBArraySeed](#)-method  
 ([TileDBArray](#)), [2](#)  
[chunkGrid](#), [2](#)  
[coerce](#), [ANY](#), [TileDBArray](#)-method  
 ([TileDBRealizationSink](#)), [5](#)  
[coerce](#), [ANY](#), [TileDBMatrix](#)-method  
 ([TileDBRealizationSink](#)), [5](#)  
[coerce](#), [ANY](#), [TileDBRealizationSink](#)-method  
 ([TileDBRealizationSink](#)), [5](#)  
[coerce](#), [TileDBRealizationSink](#), [DelayedArray](#)-method  
 ([TileDBRealizationSink](#)), [5](#)  
[coerce](#), [TileDBRealizationSink](#), [TileDBArray](#)-method  
 ([TileDBRealizationSink](#)), [5](#)  
[coerce](#), [TileDBRealizationSink](#), [TileDBMatrix](#)-method  
 ([TileDBRealizationSink](#)), [5](#)  
[COO\\_SparseArray](#), [2](#)  
  
[DelayedArray](#), [2](#), [5](#)  
[DelayedArray](#), [TileDBArraySeed](#)-method  
 ([TileDBArray](#)), [2](#)  
  
[extract\\_array](#), [2](#)  
[extract\\_array](#), [TileDBArraySeed](#)-method  
 ([TileDBArray](#)), [2](#)  
[extract\\_sparse\\_array](#), [2](#)  
[extract\\_sparse\\_array](#), [TileDBArraySeed](#)-method  
 ([TileDBArray](#)), [2](#)  
  
[getTileDBAttr](#) ([TileDBArray-globals](#)), [3](#)  
[getTileDBCcapacity](#)  
 ([TileDBArray-globals](#)), [3](#)  
[getTileDBCcellOrder](#)  
 ([TileDBArray-globals](#)), [3](#)  
[getTileDBContext](#) ([TileDBArray-globals](#)),  
[3](#)  
[getTileDBDimType](#) ([TileDBArray-globals](#)),  
[3](#)  
[getTileDBExtent](#) ([TileDBArray-globals](#)), [3](#)  
[getTileDBPath](#) ([TileDBArray-globals](#)), [3](#)  
[getTileDBTileOrder](#)  
 ([TileDBArray-globals](#)), [3](#)  
  
[is\\_sparse](#), [2](#)

[is\\_sparse](#), [TileDBArraySeed](#)-method  
 ([TileDBArray](#)), [2](#)  
  
[matrixClass](#), [TileDBArray](#)-method  
 ([TileDBArray](#)), [2](#)  
  
[path](#), [2](#)  
[path](#), [TileDBArraySeed](#)-method  
 ([TileDBArray](#)), [2](#)  
  
[r\\_to\\_tiledb\\_type](#), [6](#)  
[RealizationSink](#), [5](#)  
  
[setTileDBAttr](#) ([TileDBArray-globals](#)), [3](#)  
[setTileDBCcapacity](#)  
 ([TileDBArray-globals](#)), [3](#)  
[setTileDBCcellOrder](#)  
 ([TileDBArray-globals](#)), [3](#)  
[setTileDBContext](#) ([TileDBArray-globals](#)),  
[3](#)  
[setTileDBDimType](#) ([TileDBArray-globals](#)),  
[3](#)  
[setTileDBExtent](#) ([TileDBArray-globals](#)), [3](#)  
[setTileDBPath](#) ([TileDBArray-globals](#)), [3](#)  
[setTileDBTileOrder](#)  
 ([TileDBArray-globals](#)), [3](#)  
[show](#), [TileDBArraySeed](#)-method  
 ([TileDBArray](#)), [2](#)  
  
[tempdir](#), [4](#)  
[tiledb\\_ctx](#), [4](#), [6](#)  
[TileDBArray](#), [2](#), [6](#)  
[TileDBArray-class](#) ([TileDBArray](#)), [2](#)  
[TileDBArray-globals](#), [3](#)  
[TileDBArray-pkg](#), [5](#)  
[TileDBArraySeed](#) ([TileDBArray](#)), [2](#)  
[TileDBArraySeed-class](#) ([TileDBArray](#)), [2](#)  
[TileDBMatrix](#) ([TileDBArray](#)), [2](#)  
[TileDBMatrix-class](#) ([TileDBArray](#)), [2](#)  
[TileDBRealizationSink](#), [5](#)  
[TileDBRealizationSink-class](#)  
 ([TileDBRealizationSink](#)), [5](#)  
  
[type](#), [2](#)  
[type](#), [TileDBArraySeed](#)-method  
 ([TileDBArray](#)), [2](#)



type,TileDBRealizationSink-method  
    (TileDBRealizationSink), [5](#)

write\_block, [6](#)

write\_block,TileDBRealizationSink-method  
    (TileDBRealizationSink), [5](#)

writeTileDBArray, [4](#)

writeTileDBArray  
    (TileDBRealizationSink), [5](#)