# SciDB-Py Documentation

***Release 14.3.0***

**SciDB-Py Developers**

July 31, 2014

Contents

SciDB-Py is a Python interface to the SciDB, the massively scalable array-oriented database. It provides an intuitive NumPy-like interface to SciDB, so that users can leverage powerful distributed, data-parallel scientific computing from the comfort of their Python interpreter.

CHAPTER 1

# Installing SciDB-Py

## 1.1 Software prerequisites

The scidbpy package requires at least:

1. An available SciDB installation
2. The Shim network interface to SciDB

We assume an existing installation of SciDB is available. Binary SciDB packages (for Ubuntu 12.04 and RHEL/CentOS 6) and source code are available from http://scidb.org. The examples in this tutorial assume that SciDB is running on a computer with host name "localhost," at port 8080. If SciDB is not running on localhost, adjust the name accordingly.

The scidbpy package requires installation of a simple HTTP network service called "shim" on the computer that SciDB coordinator is installed on. The network service only needs to be installed on the SciDB computer, not on client computers that connect to SciDB from Python. It's available in packaged binary form for supported SciDB operating systems, and as source code which can be compiled and deployed on any SciDB installation. See http://github.com/paradigm4/shim for source code and installation instructions.

## 1.2 Python Prerequisites

SciDB-Py requires Python 2.6-2.7 or 3.3, as well as NumPy and Requests. Some (optional) functionality requires SciPy and Pandas. Following are a description of these requirements:

**NumPy** tested with version 1.6-1.7.

**Requests** tested with version 1.2. Required for using the Shim interface to SciDB.

**Pandas (optional)** tested with version 0.10. Required only for importing/exporting SciDB arrays as Pandas Dataframe objects.

**SciPy (optional)** tested with versions 0.10-0.12. Required only for importing/exporting SciDB arrays as SciPy sparse matrices.

## 1.3 SciDB-Py Package Installation

The latest release of `scidb-py` can be installed from the Python package index:

```
pip install scidb-py
```

The development version can be found on github at http://github.com/paradigm4/scidb-py. Install the development package directly from Github with:

```
pip install git+http://github.com/paradigm4/scidb-py.git
```

or download the code and type:

```
python setup.py install
```

CHAPTER 2

# Getting Started

SciDB is an open-source database that organizes data in n-dimensional arrays. SciDB features include ACID transactions, parallel processing, distributed storage, efficient sparse array storage, and native parallel linear algebra operations. The ScidbPy package for Python defines a NumPy array-like interface for SciDB arrays in Python. The arrays mimic numpy arrays, but operations on them are performed by the SciDB engine. Data are materialized to Python only when requested. A basic set of array subsetting, arithmetic and utility operations are defined by the package. Additionally, a general query function provides a mechanism for performing arbitrary queries on scidbpy array objects.

## 2.1 Installation

For details on installation, see *Installing SciDB-Py*.

## 2.2 Loading the scidbpy package and connecting to SciDB

In order to use SciDB, the Python instance needs an interface to a SciDB server. This is accomplished through the `SciDBInterface` class. `SciDBInterface` is an abstract base class which is designed to be extended with various interface methods: currently the one implemented interface is `SciDBShimInterface`, and interface using the Shim HTTP protocol developed by Paradigm4.

The following example loads the package and defines an object named `sdb` that represents the SciDB interface. The example assumes that the SciDB coordinator is on the computer with host name 'localhost' – adjust the host name as required if SciDB is on a different computer:

```
>>> import numpy as np
>>> from scidbpy import SciDBQueryError, SciDBArray, connect
>>> sdb = connect('http://localhost:8080')
```

The following examples refer to an interface object named `sdb` similar to the illustration.

## 2.3 SciDB arrays

SciDB arrays are composed of *cells*. Each cell may contain one or more values referred to as *attributes*. The data types and number of attributes are consistent across all cells within one array. All the attribute values within a cell may be left undefined, in which case the cell is called empty. Arrays with empty cells are referred to as sparse arrays in the SciDB documentation.

Individual attribute values may also be explicitly marked missing with one of several possible SciDB null codes.

Cells are arranged by an integer coordinate system into n-dimensional arrays. SciDB uses signed 64-bit integers for coordinates. Each coordinate axis is typically referred to as a dimension in the SciDB documentation. SciDB is limited in theory to about 100 dimensions, but in practice that limit is typically much lower (up to say, 10 dimensions or so). While the default SciDB array origin is usually zero, SciDB arrays may use any signed 64-bit integer origin.

The `SciDBArray` class defines the primary method of interaction between Python and SciDB. `SciDBArray` objects are Python representations of SciDB arrays that mimic numpy arrays in may ways. `SciDBArray` array objects are limited to the following SciDB array attribute data types: `bool`, `float32`, `float64`, `int8`, `int16`, `int32`, `int64`, `uint8`, `unit16`, `uint32`, `uint64`, and single characters.

## 2.4 Creating SciDB array objects

The following sections illustrate a number of ways to create `SciDBArray` objects. The examples assume that an `sdb` interface object has already been set up.

### 2.4.1 From a numpy array

Perhaps the simplest approach to creating an arbitrary `SciDBArray` object is to upload a numpy array into SciDB with the `from_array()` function. Although this approach is very convenient, it is not really suitable for very big arrays (which might exceed memory availability in a single computer, for example). In such cases, consider other options described below.

The following example creates a `SciDBArray` object named `Xsdb` from a small 5x4 numpy array named `X`:

```
>>> X = np.random.random((5, 4))
>>> Xsdb = sdb.from_array(X)
```

The package takes care of naming the SciDB array in this example (use `Xsdb.name` to see the SciDB array name).

### 2.4.2 From a scipy sparse matrix

In a similar way, a `SciDBArray` can be created from a scipy sparse matrix. For example:

```
>>> from scipy.sparse import coo_matrix
>>> X = np.random.random((10, 10))
>>> X[X < 0.9] = 0  # make array sparse
>>> Xcoo = coo_matrix(X)
>>> Xsdb = sdb.from_sparse(Xcoo)
```

This operation is most efficient for matrices stored in coordinate form (`coo_matrix`). Other sparse formats will be internally converted to COO form in the process of transferring the data.

### 2.4.3 Convenience array creation functions

Many standard numpy functions for creating special arrays are supported. These include:

**`zeros()`** to create an array full of zeros:

```
>>> # Create a 10x10 array of double-precision zeros:
>>> A = sdb.zeros((10,10))
```

**`ones()`** to create an array full of ones:

```
>>> # Create a 10x10 array of 64-bit signed integer ones:
>>> A = sdb.ones((10,10), dtype='int64')
```

**random()** to create an array of uniformly distributed random floating-point values:

```
>>> # Create a 10x10 array of numbers between -1 and 2 (inclusive)
>>> #    sampled from a uniform random distribution.
>>> A = sdb.random((10,10), lower=-1, upper=2)
```

**randint()** to create an array of uniformly distributed random integers:

```
>>> # Create a 10x10 array of uniform random integers between 0 and 10
>>> #  (inclusive of 0, non-inclusive of 10)
>>> A = sdb.randint((10,10), lower=0, upper=10)
```

**arange()** to create and array with evenly-spaced values given a step size:

```
>>> # Create a vector of ten integers, counting up from zero
>>> A = sdb.arange(10)
```

**linspace()** to create an array with evenly spaced values between supplied bounds:

```
>>> # Create a vector of 5 equally spaced numbers between 1 and 10,
>>> # including the endpoints:
>>> A = sdb.linspace(1, 10, 5)
```

**identity()** to create a sparse or dense identity matrix:

```
>>> # Create a 10x10 sparse, double-precision-valued identity matrix:
>>> A = sdb.identity(10, dtype='double', sparse=True)
```

These functions should be familiar to anyone who has used NumPy, and the syntax of each function closely follows its numpy counterpart. In each case, the array is defined and created directly in the SciDB server, and the resulting Python object is simply a wrapper of the native SciDB array. Because of this, the functions outlined here and in the following sections can be more efficient ways to generate large SciDB arrays than copying data from a numpy array.

Note: SciDB does not yet have a way to set a random seed, prohibiting reproducible results involving the random number generator.

### 2.4.4 From an existing SciDB array

Finally, `SciDBArray` objects may be created from existing SciDB arrays, so long as the data type restrictions outlined above are met. (It usually makes sense to load large data sets into SciDB externally from the Python package, using the SciDB parallel bulk loader or similar facility.)

The following example uses the `query()` function to build and store a small 10x5 SciDB array named "A" independently of Python. We then create a `SciDBArray` object from the SciDB array with the `wrap_array()` function, passing the name of the array identifier on the SciDB server:

```
>>> # remove A if it already exists
>>> if "A" in sdb.list_arrays():
...     sdb.query("remove(A)")

>>> # create an array named 'A' on the server
>>> sdb.query("store(build(<v:double>[i=1:10,10,0,j=1:5,5,0],i+j),A)")

>>> # create a Python object pointing to this array
>>> A = sdb.wrap_array("A")
```

Note that there are some restrictions on the types of arrays which can be wrapped by `scidbpy`. The array data must be of a compatible type, and have integer indices. Also, arrays with indices that don't start at zero may not behave as expected for item access and slicing, discussed below.

Note also that many functions in the scidbpy package work on single-attribute arrays. When a `SciDBArray` object refers to a SciDB array with more than one attribute, only the first listed attribute is used.

### 2.4.5 Persistence of SciDBpy arrays

The `new_array()` function takes an argument named `persistent`. When `persistent` is set to True, arrays last in SciDB until explicitly removed by a `remove` (AFL) or `DROP` (AQL) query. If `persistent` is set to False, the arrays are removed when the `SciDBInterface.reap()` or `SciDBArray.reap()` methods are invoked. (Note that `SciDBInterface.reap()` is automatically invoked when Python exits)

Arrays defined from an existing SciDB array using the `wrap_array()` argument are always persistent, while all other array creation routines set `persistent=False` by default:

```
>>> X = sdb.random(10, persistent=False)  # default
>>> X.name in sdb.list_arrays()
True
>>> X.reap()
>>> X.name in sdb.list_arrays()
False
```

When `SciDBInterface` is used as a context manager, non-persistent arrays are reaped at the end of the context block:

```
>>> with SciDBShimInterface(url) as sdb:
>>>     X = sdb.random(10)
>>> print X.name
"__DELETED__"
```

## 2.5 Retrieving data from SciDB array objects

A central idea of the package is to program operations on SciDB arrays in a natural Python dialect, computing those operations in SciDB while minimizing data traffic between SciDB and Python. However, it is useful to materialize SciDB array data to Python, for example to obtain and plot results.

`SciDBArray` objects provide several functions that materialize array data to Python:

**`toarray()`** can be used to populate a `numpy` array from an *N*-dimensional array with any number of attributes:

```
>>> A = sdb.linspace(0, 10, 5)
>>> A.toarray()
array([  0. ,   2.5,   5. ,   7.5,  10. ])

>>> B = sdb.join(sdb.linspace(0, 8, 5), sdb.arange(5, dtype=int))
>>> B.toarray()
array([(0.0, 0), (2.0, 1), (4.0, 2), (6.0, 3), (8.0, 4)],
      dtype=[('f0', '<f8'), ('f0_2', '<i8')])
```

**`tosparse()`** can be used to populate a SciPy sparse matrix from a 2-dimensional array with a single attribute:

```
>>> I = sdb.identity(5, sparse=True)
>>> I.tosparse(sparse_fmt='dia')
<5x5 sparse matrix of type '<type 'numpy.float64'>'
        with 5 stored elements (1 diagonals) in DIAgonal format>
```

`tosparse()` will also work with 1-dimensional arrays or multi-dimensional arrays; in this case the result cannot be exported to a SciPy sparse format, but will be returned as a Numpy record array listing the indices and values.

**`todataframe()`** can be used to populate a Pandas dataframe from a 1-dimensional array with any number of attributes:

```
>>> B = sdb.join(sdb.linspace(0, 8, 5, dtype='<A:double>'),
...              sdb.arange(1, 6, dtype='<B:int32>'),
...              sdb.ones(5, dtype='<C:float>'))
>>> B.todataframe()
   A  B  C
0  0  1  1
1  2  2  1
2  4  3  1
3  6  4  1
4  8  5  1
```

### 2.5.1 Tridiagonal Example

Let's consider a more complicated example. Here we'll use the advanced query syntax (discussed below) to efficiently create a 10x10 tridiagonal array, and import its data in both dense and sparse formats:

```
>>> tridiag = sdb.new_array((10, 10))
>>> sdb.query('store( \
...               build_sparse({A}, \
...                 iif({A.d0}={A.d1}, 2, -1), \
...                 {A.d0} <= {A.d1}+1 and {A.d0} >= {A.d1}-1), \
...               {A})',
...           A=tridiag)

>>> # Materialize SciDB array to Python as a numpy array:
>>> tridiag.toarray()
array([[ 2.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [-1.,  2.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0., -1.,  2.,  1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0., -1.,  2.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0., -1.,  2.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0., -1.,  2.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0., -1.,  2.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0., -1.,  2.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  2.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  2.]])

>>> # Materialize SciDB array to Python as a scipy.sparse array:
>>> tridiag.tosparse('csr')
<10x10 sparse matrix of type '<type 'numpy.float64'>'
        with 28 stored elements in Compressed Sparse Row format>
```

## 2.6 Operations on SciDB array objects

Operations on `SciDBArray` objects generally return new `SciDBArray` objects. The general idea is to promote function composition involving `SciDBArray` objects without moving data between SciDB and Python.

The `scidbpy` package provides quite a few common operations including subsetting, pointwise application of scalar functions, aggregations, and pointwise and matrix arithmetic.

Standard numpy attributes like `shape`, `ndim` and `size` are defined for `SciDBArray` objects:

```
>>> X = sdb.random((5, 10))
>>> X.shape
(5, 10)
>>> X.size
50
>>> X.ndim
2
```

Many SciDB-specific attributes are also defined, including `chunk_size`, `chunk_overlap`, and `sdbtype`,

```
>>> X.chunk_size
[1000, 1000]
>>> X.chunk_overlap
[0, 0]
>>> X.sdbtype
sdbtype('<f0:double>')
```

SciDBArrays also contain a `datashape` object, which encapsulates much of the interface between Python and SciDB data, including the full array schema:

```
>>> Xds = X.datashape
>>> Xds.schema
'<f0:double> [i0=0:4,1000,0,i1=0:9,1000,0]'
```

### 2.6.1 Element Access

Single elements of `SciDBArray` objects can be referenced with the standard numpy indexing syntax. These single elements are returned by value. Here we'll use the `tridiag` array created above:

```
>>> tridiag[1, 1]
2
>>> tridiag[0, 1]
-1
```

Note that element assignment (e.g. `tridiag[0, 0] = 4`)is not supported.

### 2.6.2 Subarrays

Rectilinear subarrays are also selected with standard numpy syntax. Subarrays of `SciDBArray` objects are new `SciDBArray` objects. Consider the example sparse tridiagonal array used previously:

```
>>> # Define a 3x10 subarray (returned as a new SciDBArray object)
>>> X = tridiag[2:5,:]
>>> X.toarray()
array([[ 0., -1.,  2.,  1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0., -1.,  2.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0., -1.,  2.,  1.,  0.,  0.,  0.,  0.]])
```

Tuple-based indexing (also known as *fancy indexing*) is not yet supported.

Note that subarray indexing of `SciDBArray` objects follows numpy convention. SciDB arrays with negative-valued coordinate indices should be translated to a coordinate system with a nonnegative origin.

### 2.6.3 Scalar functions of SciDBArray objects (aggregations)

The package exposes the following aggregations:

| Name | Description |
|---|---|
| `min()` | minimum value |
| `max()` | maximum value |
| `sum()` | sum of values |
| `var()` | variance of values |
| `stdev()` | standard deviation of values |
| `std()` | standard deviation of values |
| `avg()` | average/mean of values |
| `mean()` | average/mean of values |
| `count()` | count of nonempty cells |
| `approxdc()` | fast estimate of the number of distinct values |

**Examples: Minimum Aggregates**

Each operation can be computed across the entire array, or across specified dimensions by passing the index or indices of the desired dimensions. For example:

```
>>> np.random.seed(0)
>>> X = sdb.from_array(np.random.random((5, 3)))
>>> X.toarray()
array([[ 0.5488135 ,  0.71518937,  0.60276338],
       [ 0.54488318,  0.4236548 ,  0.64589411],
       [ 0.43758721,  0.891773  ,  0.96366276],
       [ 0.38344152,  0.79172504,  0.52889492],
       [ 0.56804456,  0.92559664,  0.07103606]])
```

Here we'll find the minimum of all values in the array. The returned result is a new SciDBArray, so we select the first element:

```
>>> X.min()[0]
0.071036058197886942
```

Like numpy, passing index 0 gives us the minimum within every column:

```
>>> X.min(0).toarray()
array([ 0.38344152,  0.4236548 ,  0.07103606])
```

Passing index 1 gives us the minimum within every row:

```
>>> X.min(1).toarray()
array([ 0.5488135 ,  0.4236548 ,  0.43758721,  0.38344152,  0.07103606])
```

Note that the convention for specifying aggregate indices here is designed to match numpy, and is *opposite the convention used within SciDB*. To recover SciDB-style aggregates, you can use the `scidb_syntax` flag:

```
>>> X.min(1, scidb_syntax=True).toarray()
array([ 0.38344152,  0.4236548 ,  0.07103606])
```

**Further Examples**

These operations return new `SciDBArray` objects consisting of scalar values. Here are a few examples that materialize their results to Python (using the `tridiag` array defined previously):

```
>>> tridiag.count()[0]
28
>>> tridiag.sum()[0]
```

```
20.0
>>> tridiag.var()[0]
1.6190476190476193
```

Note that a count of nonempty cells is also directly available from the `nonempty()` function:

```
>>> tridiag.nonempty()
28
```

A related function is `nonnull()`, which counts the number of nonempty cells which do not contain a null value. In this case, the result is the same as `nonempty()`:

```
>>> tridiag.nonnull()
28
```

### 2.6.4 Pointwise application of scalar functions

The package exposes SciDB scalar-valued scalar functions that can be applied element-wise to SciDB arrays:

| Function | Description |
|---|---|
| `sin()` | Trigonometric sine |
| `asin()` | Trigonometric arc-sine / inverse sine |
| `cos()` | Trigonometric cosine |
| `acos()` | Trigonometric arc-cosine / inverse cosine |
| `tan()` | Trigonometric tangent |
| `atan()` | Trigonometric arc-tangent / inverse tagent |
| `exp()` | Natural exponent |
| `log()` | Natural logarithm |
| `log10()` | Base-10 logarithm |

All trigonometric functions assume arguments are given in radians. Here is a simple example that compares a computation in SciDB with a local one (using the 'tridiag' array defined in the last examples):

```
>>> sin_tri = sdb.sin(tridiag)
>>> np.linalg.norm(sin_tri.toarray() - np.sin(tridiag.toarray()))
0.0
```

### 2.6.5 Shape and layout functions

Arrays may be transposed and their data re-arranged into new shapes with the usual `transpose()` and `reshape()` functions:

```
>>> tri_reshape = tridiag.reshape((20,5))
>>> tri_reshape.shape
(20, 5)
>>> tri_reshape.transpose().shape
(5, 20)
>>> tri_reshape.T.shape  # shortcut for transpose
(5, 20)
```

### 2.6.6 Arithmetic

The package defines elementwise operations on all arrays and linear algebra operations on matrices and vectors. Scalar multiplication is supported.

Element-wise sums and products:

```
>>> np.random.seed(1)
>>> X = sdb.from_array(np.random.random((10, 10)))
>>> Y = sdb.from_array(np.random.random((10, 10)))
>>> S = X + Y
>>> D = X - Y
>>> M = 2 * X
>>> (S + D - M).sum()[0]
-1.1102230246251565e-16
```

We can combine operations as well:

```
>>> Z = 0.5 * (X + X.T)
```

There are also linear algebra operations (matrix-matrix product, matrix-vector product) using the `dot()` function:

```
>>> XY = sdb.dot(X, Y)
>>> XY1 = sdb.dot(X, Y[:,1])
>>> XTX = sdb.dot(X.T, X)
```

### 2.6.7 Broadcasting

Numpy broadcasting conventions are generally followed in operations involving differently-sized `SciDBArray` objects. Consider the following example that centers a matrix by subtracting its column average from each column.

First we create a test array with 5 columns:

```
>>> np.random.seed(0)
>>> X = sdb.from_array(np.random.random((10, 5)))
```

Now create a vector of column means:

```
>>> xcolmean = X.mean(0)
>>> xcolmean.shape
(5,)
```

Subtract these means from the columns – this is a broadcasting operation:

```
>>> XC = X - xcolmean
```

To check that the columns are now centered, we compute the column mean of `XC`:

```
>>> XC.mean(1).toarray()
array([ -2.22044605e-17,   4.44089210e-17,  -1.11022302e-17,
         1.11022302e-16,  -3.33066907e-17])
```

The broadcasting operation which creates `XC` is implemented using a join operation along dimension 1.

## 2.7 Advanced Usage: the SciDB Query Interface

`scidbpy` provides python wrappers for many useful SciDB operations, but the SciDB AFL and AQL query languages can provide even more customization of operations (For more information on SciDB's AFL and AQL languages, see the SciDB Manual). The `query()` function provides a useful interface for generating raw queries by exploiting Python's String Formatting syntax. Through automatic insertion of the server-side identifiers of SciDB arrays, attributes, and dimensions, the query interface makes constructing complicated queries very convenient.

The general approach first creates a new `SciDBArray` object and then issues a query to populate data. For example, to build an array of zeros similar to the result of the `zeros()` function shown above, the query can be constructed in the following way:

```
>>> # first define an empty array to hold the result
>>> zeros = sdb.new_array(shape=(5, 5), dtype='double')
>>> # now execute a query to fill the array
>>> sdb.query('store(build({A}, 0), {A})', A=zeros)
>>> zeros.toarray()
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
```

The result is that `zeros` is a 10x10 array filled with zeros. Here the format statement `{A}` is replaced by the name of the desired array on the SciDB server.

We can use this interface to quickly build more complex arrays. For example, to create an identity matrix similar to the result of the `identity()` function shown above, we add a boolean check:

```
>>> ident = sdb.new_array((5, 5), dtype='double')
>>> sdb.query('store(build({A}, iif({A.d0}={A.d1}, 1, 0)), {A})',
...           A=ident)
>>> ident.toarray()
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.]])
```

Here the substitutions `{A.d0}` and `{A.d1}` are replaced by the first and second dimension names of the array referenced by `A`.

Things can become even more complicated. The following example creates a 5x5 sparse tridiagonal array, similar to the one used in the above examples:

```
>>> tridiag = sdb.new_array((5, 5))
>>> sdb.query('store( \
...               build_sparse({A}, \
...                 iif({A.d0}={A.d1}, 2, -1), \
...                 {A.d0} <= {A.d1}+1 and {A.d0} >= {A.d1}-1), \
...               {A})',
...           A=tridiag)
>>> tridiag.toarray()
array([[ 2., -1.,  0.,  0.,  0.],
       [-1.,  2., -1.,  0.,  0.],
       [ 0., -1.,  2., -1.,  0.],
       [ 0.,  0., -1.,  2., -1.],
       [ 0.,  0.,  0., -1.,  2.]])
```

The query builds a sparse tridiagonal array with 2 on the diagonal and -1 on the sub- and super-diagonals. This shows how the query-formatting syntax provided by the scidbpy package can be used to generate extremely powerful AFL queries.

The full replacement syntax is outlined in the documentation of the `query()` function. It is a useful way to help streamline the process of writing SciDB queries if and when it becomes necessary.

## 2.8 Example applications

### 2.8.1 Covariance and correlation matrices

We can use SciDB's distributed parallel linear algebra operations to compute covariance and correlation matrices without too much difficulty. The following example computes the covariance and correlation between the columns of the matrices X and Y. We break the example up into a few parts for clarity.

Part 1, set up some example matrices:

```
# Two small arrays, each with 1000 rows, and with 5 and 3 columns
>>> x = np.random.random((1000, 5))
>>> y = np.column_stack((x[:, 0] * 2, x[:, 1] + x[:, 0] / 2., x[:, 4]))

>>> X = sdb.from_array(x)
>>> Y = sdb.from_array(y)
```

Part 2, center the example matrices:

```
>>> # Subtract the column means from X using broadcasting:
>>> XC = X - X.mean(0)

>>> # Similarly subtract the column means from Y:
>>> YC = Y - Y.mean(0)
```

Part 3, compute the covariance matrix:

```
>>> COV = sdb.dot(XC.T, YC) / (X.shape[0] - 1)
```

Part 4, compute the correlation matrix:

```
>>> # Column vector with column standard deviations of X matrix:
>>> xsd = X.std(0).reshape((5, 1))

>>> # Row vector with column standard deviations of Y matrix:
>>> ysd = Y.std(0).reshape((1, 3))

>>> # Their outer product:
>>> outersd = sdb.dot(xsd, ysd)

>>> COR = COV / outersd
>>> COR.toarray()
array([[ 1.        ,  0.46877777,  0.0063839 ],
       [ 0.02479068,  0.89466598,  0.00688237],
       [-0.01463645, -0.04804887,  0.03429709],
       [-0.01133627,  0.01364739,  0.05206071],
       [ 0.0063839 ,  0.00893397,  1.        ]])
```

The overhead of working interactively with SciDB can make these examples run somewhat slowly for small problems. But the same code shown here can be applied to arbitrarily large matrices, and those computations can run in parallel across a cluster.

## 2.9 Working with the Array Functional Language

The syntax for using `SciDBArray` instances resembles working with NumPy arrays. SciDB provides another interface for working with arrays, called the Array Functional Language, or AFL. The AFL consists of approximately 100 functions to perform array analysis. You can access these operators through the `afl` attribute of a SciDB instance:

```
>>> X = sdb.ones(100)
>>> sdb.afl.sum(X)
SciDB Expression: <sum(py1100988436438_00001)>
>>> sdb.afl.sum(X).eval()  # returns a SciDB array
SciDBArray('not empty py1100988436438_00002<f0_sum:double NULL DEFAULT null> [i=0:0,1,0]')
>>> sdb.afl.sum(X).toarray()  # converts to a NumPy array
array([ 100.])
```

Note that AFL queries can be nested inside each other. The following code computes a (50, 50, 50) array, sums over the second 2 dimensions, and finds the maximum value of those 50 numbers:

```
>>> X = sdb.random((50, 50, 50))
>>> afl = sdb.afl
>>> afl.max(afl.sum(X, 'f0', 'i0')).toarray()
array([ 29.65605829])
```

Working with AFL has a few advantages:

- **AFL functions map directly onto database queries, giving you** more control over query building
- **AFL expressions are evaluated lazily – no database communication** occurs until you call the `eval` or `toarray` methods on an expression. This means you can compose complex queries, without unnecessary communication with the server. Likewise, passing complex AFL expressions makes it easier for SciDB to perform query optimization.

CHAPTER 3

# Code Reference

This is the list of classes and functions available in SciDB-py.

## 3.1 SciDB Array Class

**class** scidbpy.**SciDBArray**(*datashape*, *interface*, *name*, *persistent=False*)

SciDBArray class

It is not recommended to instantiate this class directly; use a convenience routine from SciDBInterface.

### Attributes

| | |
|---|---|
| T | Permute the dimensions of an array. |
| afl | An alias to the AFL namespace |
| chunk_overlap | |
| chunk_size | |
| datashape | |
| dim_names | |
| dtype | |
| ndim | |
| sdbtype | |
| shape | |
| size | |

### Methods

| | |
|---|---|
| alias([name]) | Return an alias of the array, optionally with a new name |
| approxdc([index, scidb_syntax]) | Return the number of distinct values of the array or along an axis. |
| att(a) | Return the attribute name of the array. |
| attribute(a) | Return the attribute name of the array. |
| avg([index, scidb_syntax]) | Return the average of the array or the average along an axis. |
| contains_nulls([attr]) | Return True if the array contains null values. |
| contents(**kwargs) | Return a string representation of the array contents |
| copy([new_name, persistent]) | Make a copy of the array in the database |
| count([index, scidb_syntax]) | Return the count of the array or the count along an axis. |

Continued on next page

Table 3.2 – continued from previous page

| | |
|---|---|
| dimension(d) | Return the dimension name of the array |
| issparse() | Check whether array is sparse. |
| max([index, scidb_syntax]) | Return the maximum of the array or the maximum along an axis. |
| mean([index, scidb_syntax]) | Return the average of the array or the average along an axis. |
| min([index, scidb_syntax]) | Return the minimum of the array or the minimum along an axis. |
| nonempty() | Return the number of nonempty elements in the array. |
| nonnull([attr]) | Return the number of non-empty and non-null values. |
| reap([ignore]) | Delete this object from the database if it isn't persistent. |
| regrid(size[, aggregate]) | Regrid the array using the specified aggregate |
| rename(new_name[, persistent]) | Rename the array in the database, optionally making the new array persistent. |
| reshape(shape, **kwargs) | Reshape data into a new array |
| std([index, scidb_syntax]) | Return the standard deviation of the array or along an axis. |
| stdev([index, scidb_syntax]) | Return the standard deviation of the array or along an axis. |
| substitute(value) | Reshape data into a new array, substituting a default for any nulls. |
| sum([index, scidb_syntax]) | Return the sum of the array or the sum along an axis. |
| toarray([transfer_bytes]) | Transfer data from database and store in a numpy array. |
| todataframe([transfer_bytes]) | Transfer array from database and store in a local Pandas dataframe |
| tosparse([sparse_fmt, transfer_bytes]) | Transfer array from database and store in a local sparse array. |
| transpose(*axes) | Permute the dimensions of an array. |
| var([index, scidb_syntax]) | Return the variance of the array or the variance along an axis. |

**T**

Permute the dimensions of an array.

**Parameters** **axes** : None, tuple of ints, or *n* ints

- None or no argument: reverses the order of the axes.
- tuple of ints: *i* in the *j*-th place in the tuple means *a*'s *i*-th axis becomes *a.transpose()*'s *j*-th axis.
- *n* ints: same as an n-tuple of the same ints (this form is intended simply as a "convenience" alternative to the tuple form)

**Returns** **out** : ndarray

Copy of *a*, with axes suitably permuted.

**afl**

An alias to the AFL namespace

**alias**(*name=None*)

Return an alias of the array, optionally with a new name

**approxdc**(*index=None*, *scidb_syntax=False*)

Return the number of distinct values of the array or along an axis.

The distinct count is an estimate only.

**Parameters** **index** : int, optional

Axis along which to operate. By default, flattened input is used.

**scidb_syntax** : bool, optional (default=False)

If False, index follows the numpy convention (i.e., the array is collapsed over the index'th axis). If True, index follows the SciDB convention (i.e., the array is collapsed over all axes *except* index)

**Returns** A SciDB array

**att**(*a*)
Return the attribute name of the array.

**Parameters** **a** : int

Index of the attribute to lookup

**attribute**(*a*)
Return the attribute name of the array.

**Parameters** **a** : int

Index of the attribute to lookup

**avg**(*index=None*, *scidb_syntax=False*)
Return the average of the array or the average along an axis.

**Parameters** **index** : int, optional

Axis along which to operate. By default, flattened input is used.

**scidb_syntax** : bool, optional (default=False)

If False, index follows the numpy convention (i.e., the array is collapsed over the index'th axis). If True, index follows the SciDB convention (i.e., the array is collapsed over all axes *except* index)

**Returns** A SciDB array

**contains_nulls**(*attr=None*)
Return True if the array contains null values.

**Parameters** **attr** : None, int, or array_like

the attribute index/indices to check. If None, then check all.

**Returns** **contains_nulls** : boolean

**contents**(***kwargs*)
Return a string representation of the array contents

**copy**(*new_name=None*, *persistent=False*)
Make a copy of the array in the database

**Parameters** **new_name** : string (optional)

if specifiedmust be a valid array name which does not already exist in the database.

**persistent** : boolean (optional)

specify whether the new array is persistent (default=False)

**Returns** **copy** : SciDBArray

return a copy of the original array

**count**(*index=None*, *scidb_syntax=False*)
Return the count of the array or the count along an axis.

The count is equal to the number of nonnull elements.

**Parameters** **index** : int, optional

Axis along which to operate. By default, flattened input is used.

**scidb_syntax** : bool, optional (default=False)

If False, index follows the numpy convention (i.e., the array is collapsed over the index'th axis). If True, index follows the SciDB convention (i.e., the array is collapsed over all axes *except* index)

**Returns** A SciDB array

**dimension**(*d*)
Return the dimension name of the array

**Parameters** **d** : int

The index of the dimension to lookup

**issparse**()
Check whether array is sparse.

**max**(*index=None*, *scidb_syntax=False*)
Return the maximum of the array or the maximum along an axis.

**Parameters** **index** : int, optional

Axis along which to operate. By default, flattened input is used.

**scidb_syntax** : bool, optional (default=False)

If False, index follows the numpy convention (i.e., the array is collapsed over the index'th axis). If True, index follows the SciDB convention (i.e., the array is collapsed over all axes *except* index)

**Returns** A SciDB array

**mean**(*index=None*, *scidb_syntax=False*)
Return the average of the array or the average along an axis.

**Parameters** **index** : int, optional

Axis along which to operate. By default, flattened input is used.

**scidb_syntax** : bool, optional (default=False)

If False, index follows the numpy convention (i.e., the array is collapsed over the index'th axis). If True, index follows the SciDB convention (i.e., the array is collapsed over all axes *except* index)

**Returns** A SciDB array

#### Notes

Identical to `SciDBArray.avg()`

**min**(*index=None*, *scidb_syntax=False*)
Return the minimum of the array or the minimum along an axis.

**Parameters** **index** : int, optional

Axis along which to operate. By default, flattened input is used.

**scidb_syntax** : bool, optional (default=False)

If False, index follows the numpy convention (i.e., the array is collapsed over the index'th axis). If True, index follows the SciDB convention (i.e., the array is collapsed over all axes *except* index)

**Returns** A SciDB array

**nonempty**()
Return the number of nonempty elements in the array.

Nonempty refers to the sparsity of an array, and thus includes in the count elements with values which are set to NULL.

**See also:**

nonnull

**nonnull**(*attr=0*)
Return the number of non-empty and non-null values.

This query must be done for each attribute: the default is the first attribute.

**Parameters** **attr** : None, int or array_like

the attribute or attributes to query. If None, then query all attributes.

**Returns** **nonnull** : array_like

the nonnull count for each attribute. The returned value is the same shape as the input `attr`.

**See also:**

nonempty

**reap**(*ignore=False*)
Delete this object from the database if it isn't persistent.

**Parameters** **ignore** : bool (default False)

If False and the array is persistent, then reap raises an error If True and the array is persistent, reap does nothing

**Raises** **SciDBForbidden if ‘‘persistent=True‘‘ and ‘‘ignore=False‘**

**regrid**(*size*, *aggregate='avg'*)
Regrid the array using the specified aggregate

**Parameters** **size** : int or tuple of ints

Specify the size of the regridding along each dimension. If a single integer, then use the same regridding along each dimension.

**aggregate** : string

specify the aggregation function to use when creating the new grid. Default is ‘avg’. Possible values are: [’avg’, ‘sum’, ‘min’, ‘max’, ‘count’, ‘stdev’, ‘var’, ‘approxdc’]

**Returns** **A** : scidbarray

The re-gridded version of the array. The size of dimension i is ceil(self.shape[i] / size[i])

**rename**(*new_name*, *persistent=False*)
Rename the array in the database, optionally making the new array persistent.

**Parameters** **new_name** : string

must be a valid array name which does not already exist in the database.

**persistent** : boolean (optional)

specify whether the new array is persistent (default=False)

**Returns** **self** : SciDBArray

return a pointer to self

**reshape**(*shape*, ***kwargs*)
Reshape data into a new array

**Parameters** **shape** : tuple or int

The shape of the new array. Must be compatible with the current shape

****kwargs :**

additional keyword arguments will be passed to SciDBDatashape

**Returns** **arr** : SciDBArray

new array of the specified shape

**std**(*index=None*, *scidb_syntax=False*)
Return the standard deviation of the array or along an axis.

**Parameters** **index** : int, optional

Axis along which to operate. By default, flattened input is used.

**scidb_syntax** : bool, optional (default=False)

If False, index follows the numpy convention (i.e., the array is collapsed over the index'th axis). If True, index follows the SciDB convention (i.e., the array is collapsed over all axes *except* index)

**Returns** A SciDB array

#### Notes

Identical to `SciDBArray.stdev()`

**stdev**(*index=None*, *scidb_syntax=False*)
Return the standard deviation of the array or along an axis.

**Parameters** **index** : int, optional

Axis along which to operate. By default, flattened input is used.

**scidb_syntax** : bool, optional (default=False)

If False, index follows the numpy convention (i.e., the array is collapsed over the index'th axis). If True, index follows the SciDB convention (i.e., the array is collapsed over all axes *except* index)

**Returns** A SciDB array

**substitute**(*value*)
Reshape data into a new array, substituting a default for any nulls.

**Parameters** **value** : value to replace nulls (required)

**Returns** **arr** : SciDBArray

new non-nullable array

**sum**(*index=None*, *scidb_syntax=False*)
Return the sum of the array or the sum along an axis.

**Parameters** **index** : int, optional

Axis along which to operate. By default, flattened input is used.

**scidb_syntax** : bool, optional (default=False)

If False, index follows the numpy convention (i.e., the array is collapsed over the index'th axis). If True, index follows the SciDB convention (i.e., the array is collapsed over all axes *except* index)

**Returns** A SciDB array

**toarray**(*transfer_bytes=True*)
Transfer data from database and store in a numpy array.

**Parameters** **transfer_bytes** : boolean

if True (default), then transfer data as bytes rather than as ASCII.

**Returns** **arr** : np.ndarray

The dense array containing the data.

**todataframe**(*transfer_bytes=True*)
Transfer array from database and store in a local Pandas dataframe

This is valid only for a one-dimensional array.

**Parameters** **transfer_bytes** : boolean

if True (default), then transfer data as bytes rather than as ASCII.

**Returns** **arr** : pd.DataFrame

The dataframe object containing the data in the array.

**tosparse**(*sparse_fmt='recarray'*, *transfer_bytes=True*)
Transfer array from database and store in a local sparse array.

**Parameters** **transfer_bytes** : boolean

if True (default), then transfer data as bytes rather than as ASCII. This is more accurate, but requires two passes over the data (one for indices, one for values).

**sparse_format** : string or None

Specify the sparse format to use. Available formats are: - 'recarray' : a record array containing the indices and

values for each data point. This is valid for arrays of any dimension and with any number of attributes.

- ['coo'|'csc'|'csr'|'dok'|'lil'] : a scipy sparse matrix. These are valid only for 2-dimensional arrays with a single attribute.

**Returns** **arr** : ndarray or sparse matrix

The sparse representation of the data

**transpose**(**axes*)
Permute the dimensions of an array.

**Parameters** **axes** : None, tuple of ints, or *n* ints

- None or no argument: reverses the order of the axes.
- tuple of ints: *i* in the *j*-th place in the tuple means *a*'s *i*-th axis becomes *a.transpose()*'s *j*-th axis.

- *n* ints: same as an n-tuple of the same ints (this form is intended simply as a "convenience" alternative to the tuple form)

**Returns** **out** : ndarray

Copy of *a*, with axes suitably permuted.

**var** (*index=None*, *scidb_syntax=False*)
Return the variance of the array or the variance along an axis.

**Parameters** **index** : int, optional

Axis along which to operate. By default, flattened input is used.

**scidb_syntax** : bool, optional (default=False)

If False, index follows the numpy convention (i.e., the array is collapsed over the index'th axis). If True, index follows the SciDB convention (i.e., the array is collapsed over all axes *except* index)

**Returns** A SciDB array

## 3.2 SciDB Interface

### 3.2.1 Base Class

**class** scidbpy.**SciDBInterface**

#### Attributes

| |
|---|
| afl |

#### Methods

| | |
|---|---|
| acos(A) | Element-wise trigonometric inverse cosine |
| approxdc(A[, index, scidb_syntax]) | Array or axis unique element estimate. |
| arange([start,] stop[, step,][, dtype]) | Return evenly spaced values within a given interval. |
| asin(A) | Element-wise trigonometric inverse sine |
| atan(A) | Element-wise trigonometric inverse tangent |
| avg(A[, index, scidb_syntax]) | Array or axis average. |
| cos(A) | Element-wise trigonometric cosine |
| count(A[, index, scidb_syntax]) | Array or axis count. |
| cross_join(A, B, *dims) | Perform a cross-join on arrays A and B. |
| dot(A, B) | Compute the matrix product of A and B |
| exp(A) | Element-wise natural exponent |
| from_array(A[, instance_id]) | Initialize a scidb array from a numpy array |
| from_dataframe(A[, instance_id]) | Initialize a scidb array from a pandas dataframe |
| from_sparse(A[, instance_id]) | Initialize a scidb array from a sparse array |
| identity(n[, dtype, sparse]) | Return a 2-dimensional square identity matrix of size n |
| join(*args) | Perform a series of array joins on the arguments and return the result. |
| linspace(start, stop[, num, endpoint, retstep]) | Return evenly spaced numbers over a specified interval. |
| list_arrays([parsed, n]) | List the arrays currently in the database |

Table 3.4 – continued from previous page

| | |
|---|---|
| log(A) | Element-wise natural logarithm |
| log10(A) | Element-wise base-10 logarithm |
| max(A[, index, scidb_syntax]) | Array or axis maximum. |
| mean(A[, index, scidb_syntax]) | Array or axis mean. |
| merge(A, B) | Merge two arrays |
| min(A[, index, scidb_syntax]) | Array or axis minimum. |
| new_array([shape, dtype, persistent]) | Create a new array, either instantiating it in SciDB or simply reserving the name |
| ones(shape[, dtype]) | Return an array of ones |
| query(query, *args, **kwargs) | Perform a query on the database. |
| randint(shape[, dtype, lower, upper, persistent]) | Return an array of random integers between lower and upper |
| random(shape[, dtype, lower, upper, persistent]) | Return an array of random floats between lower and upper |
| reap() | Reap all arrays created via new_array |
| sin(A) | Element-wise trigonometric sine |
| std(A[, index, scidb_syntax]) | Array or axis standard deviation. |
| stdev(A[, index, scidb_syntax]) | Array or axis standard deviation. |
| substitute(A, value) | Replace null values in an array |
| sum(A[, index, scidb_syntax]) | Array or axis sum. |
| svd(A[, return_U, return_S, return_VT]) | Compute the Singular Value Decomposition of the array A: |
| tan(A) | Element-wise trigonometric tangent |
| toarray(A[, transfer_bytes]) | Convert a SciDB array to a numpy array |
| todataframe(A[, transfer_bytes]) | Convert a SciDB array to a pandas dataframe |
| tosparse(A[, sparse_fmt, transfer_bytes]) | Convert a SciDB array to a sparse representation |
| var(A[, index, scidb_syntax]) | Array or axis variance. |
| wrap_array(scidbname[, persistent]) | Create a new SciDBArray object that references an existing SciDB |
| zeros(shape[, dtype]) | Return an array of zeros |

**acos**(*A*)
Element-wise trigonometric inverse cosine

**approxdc**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis unique element estimate.

see `SciDBArray.approxdc()`

**arange**([*start*], *stop*[, *step*], *dtype=None*, ***kwargs*)
Return evenly spaced values within a given interval.

Values are generated within the half-open interval `[start, stop)` (in other words, the interval including *start* but excluding *stop*). For integer arguments the behavior is equivalent to the Python range function, but returns an ndarray rather than a list.

When using a non-integer step, such as 0.1, the results will often not be consistent. It is better to use `linspace` for these cases.

**Parameters** **start** : number, optional

Start of interval. The interval includes this value. The default start value is 0.

**stop** : number

End of interval. The interval does not include this value, except in some cases where *step* is not an integer and floating point round-off affects the length of *out*.

**step** : number, optional

Spacing between values. For any output *out*, this is the distance between two adjacent values, `out[i+1] - out[i]`. The default step size is 1. If *step* is specified, *start* must also be given.

**dtype** : dtype

The type of the output array. If *dtype* is not given, it is inferred from the type of the input arguments.

****kwargs :**

Additional arguments are passed to SciDBDatashape when creating the output array.

**Returns** **arange** : SciDBArray

Array of evenly spaced values.

For floating point arguments, the length of the result is `ceil((stop - start)/step)`. Because of floating point overflow, this rule may result in the last element of *out* being greater than *stop*.

**asin**(*A*)
Element-wise trigonometric inverse sine

**atan**(*A*)
Element-wise trigonometric inverse tangent

**avg**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis average.

see `SciDBArray.avg()`

**cos**(*A*)
Element-wise trigonometric cosine

**count**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis count.

see `SciDBArray.count()`

**cross_join**(*A*, *B*, **dims*)
Perform a cross-join on arrays A and B.

**Parameters** **A, B** : SciDBArray

***dims** : tuples

The remaining arguments are tuples of dimension indices which should be joined.

**dot**(*A*, *B*)
Compute the matrix product of A and B

**Parameters** **A** : SciDBArray

A must be a two-dimensional matrix of shape (n, p)

**B** : SciDBArray

B must be a two-dimensional matrix of shape (p, m)

**Returns** **C** : SciDBArray

The wrapper of the SciDB Array, of shape (n, m), consisting of the matrix product of A and B

**exp**(*A*)
Element-wise natural exponent

**from_array**(*A*, *instance_id=0*, ***kwargs*)
Initialize a scidb array from a numpy array

**Parameters** **A** : array_like (numpy array or sparse array)

input array from which the scidb array will be created

**instance_id** : integer

the instance ID used in loading (default=0; see SciDB documentation)

****kwargs :**

Additional keyword arguments are passed to new_array()

**Returns** **arr** : SciDBArray

SciDB Array object built from the input array

**from_dataframe**(*A*, *instance_id=0*, ***kwargs*)
Initialize a scidb array from a pandas dataframe

**Parameters** **A** : pandas dataframe

data from which the scidb array will be created.

**instance_id** : integer

the instance ID used in loading (default=0; see SciDB documentation)

****kwargs :**

Additional keyword arguments are passed to new_array()

**Returns** **arr** : SciDBArray

SciDB Array object built from the input array

**from_sparse**(*A*, *instance_id=0*, ***kwargs*)
Initialize a scidb array from a sparse array

**Parameters** **A** : sparse array

sparse input array from which the scidb array will be created. Note that this array will internally be converted to COO format.

**instance_id** : integer

the instance ID used in loading (default=0; see SciDB documentation)

****kwargs :**

Additional keyword arguments are passed to new_array()

**Returns** **arr** : SciDBArray

SciDB Array object built from the input array

**identity**(*n*, *dtype='double'*, *sparse=False*, ***kwargs*)
Return a 2-dimensional square identity matrix of size n

**Parameters** **n** : integer

the number of rows and columns in the matrix

**dtype** : string or list

The data type of the array

**sparse** : boolean

specify whether to create a sparse array (default=False)

****kwargs :**

Additional keyword arguments are passed to SciDBDataShape.

**Returns arr** : SciDBArray

A SciDBArray containint an [n x n] identity matrix

**join** (**args*)

Perform a series of array joins on the arguments and return the result.

**linspace** (*start*, *stop*, *num=50*, *endpoint=True*, *retstep=False*, ***kwargs*)

Return evenly spaced numbers over a specified interval.

Returns *num* evenly spaced samples, calculated over the interval [*start*, *stop* ].

The endpoint of the interval can optionally be excluded.

**Parameters start** : scalar

The starting value of the sequence.

**stop** : scalar

The end value of the sequence, unless *endpoint* is set to False. In that case, the sequence consists of all but the last of `num + 1` evenly spaced samples, so that *stop* is excluded. Note that the step size changes when *endpoint* is False.

**num** : int, optional

Number of samples to generate. Default is 50.

**endpoint** : bool, optional

If True, *stop* is the last sample. Otherwise, it is not included. Default is True.

**retstep** : bool, optional

If True, return (*samples*, *step*), where *step* is the spacing between samples.

****kwargs :**

additional keyword arguments are passed to SciDBDataShape

**Returns samples** : SciDBArray

There are *num* equally spaced samples in the closed interval `[start, stop]` or the half-open interval `[start, stop)` (depending on whether *endpoint* is True or False).

**step** : float (only if *retstep* is True)

Size of spacing between samples.

**list_arrays** (*parsed=True*, *n=0*)

List the arrays currently in the database

**Parameters parsed** : boolean

If True (default), then parse the results into a dictionary of array names as keys, schema as values

**n** : integer

the maximum number of arrays to list. If n=0, then list all

**Returns array_list** : string or dictionary

The list of arrays. If parsed=True, then the result is returned as a dictionary.

**log** (*A*)
Element-wise natural logarithm

**log10** (*A*)
Element-wise base-10 logarithm

**max** (*A*, *index=None*, *scidb_syntax=False*)
Array or axis maximum.

see SciDBArray.max()

**mean** (*A*, *index=None*, *scidb_syntax=False*)
Array or axis mean.

see SciDBArray.mean()

**merge** (*A*, *B*)
Merge two arrays

**min** (*A*, *index=None*, *scidb_syntax=False*)
Array or axis minimum.

see SciDBArray.min()

**new_array** (*shape=None*, *dtype='double'*, *persistent=False*, ***kwargs*)
Create a new array, either instantiating it in SciDB or simply reserving the name for use in a later query.

**Parameters** **shape** : int or tuple (optional)

The shape of the array to create. If not specified, no array will be created and a name will simply be reserved for later use. WARNING: if shape=None and persistent=False, an error will result when the array goes out of scope, unless the name is used to create an array on the server.

**dtype** : string (optional)

the datatype of the array. This is only referenced if *shape* is specified. Default is 'double'.

**persistent** : boolean (optional)

whether the created array should be persistent, i.e. survive in SciDB past when the object wrapper goes out of scope. Default is False.

****kwargs** : (optional)

If *shape* is specified, additional keyword arguments are passed to SciDBDataShape. Otherwise, these will not be referenced.

**Returns**

**———-**

**arr** : SciDBArray

wrapper of the new SciDB array instance.

**ones** (*shape*, *dtype='double'*, ***kwargs*)
Return an array of ones

**Parameters** **shape** : tuple or int

The shape of the array

**dtype** : string or list

The data type of the array

****kwargs :**

Additional keyword arguments are passed to SciDBDataShape.

**Returns** arr: SciDBArray

A SciDBArray consisting of all ones.

**query** (*query*, **args*, ***kwargs*)
Perform a query on the database.

This wraps a query constructor which allows the creation of sophisticated SciDB queries which act on arrays wrapped by SciDBArray objects. See Notes below for details.

**Parameters** **query** : string

The query string, with curly-braces to indicate insertions

***args, **kwargs :**

Values to be inserted (see below).

**randint** (*shape*, *dtype='uint32'*, *lower=0*, *upper=2147483647*, *persistent=False*, ***kwargs*)
Return an array of random integers between lower and upper

**Parameters** **shape** : tuple or int

The shape of the array

**dtype** : string or list

The data type of the array

**lower** : float

The lower bound of the random sample (default=0)

**upper** : float

The upper bound of the random sample (default=2147483647)

**persistent** : bool

Whether the array is persistent (default=False)

****kwargs :**

Additional keyword arguments are passed to SciDBDataShape.

**Returns** arr: SciDBArray

A SciDBArray consisting of random integers, uniformly distributed between *lower* and *upper*.

**random** (*shape*, *dtype='double'*, *lower=0*, *upper=1*, *persistent=False*, ***kwargs*)
Return an array of random floats between lower and upper

**Parameters** **shape** : tuple or int

The shape of the array

**dtype** : string or list

The data type of the array

**lower** : float

The lower bound of the random sample (default=0)

**upper** : float

The upper bound of the random sample (default=1)

**persistent** : bool

Whether the new array is persistent (default=False)

****kwargs :**

Additional keyword arguments are passed to SciDBDataShape.

**Returns** arr: SciDBArray

A SciDBArray consisting of random floating point numbers, uniformly distributed between *lower* and *upper*.

**reap**()
Reap all arrays created via new_array

**sin**(*A*)
Element-wise trigonometric sine

**std**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis standard deviation.

see `SciDBArray.std()`

**stdev**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis standard deviation.

see `SciDBArray.stdev()`

**substitute**(*A*, *value*)
Replace null values in an array

See `SciDBArray.substitute()`

**sum**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis sum.

see `SciDBArray.sum()`

**svd**(*A*, *return_U=True*, *return_S=True*, *return_VT=True*)
Compute the Singular Value Decomposition of the array A:

A = U.S.V^T

**Parameters** **A** : SciDBArray

The array for which the SVD will be computed. It should be a 2-dimensional array with a single value per cell. Currently, the svd routine requires non-overlapping chunks of size 32.

**return_U, return_S, return_VT** : boolean

if any is True, then return the associated array. All are True by default

**Returns** **[U], [S], [VT]** : SciDBArrays

Arrays storing the singular values and vectors of A.

**tan**(*A*)
Element-wise trigonometric tangent

**toarray**(*A*, *transfer_bytes=True*)
Convert a SciDB array to a numpy array

**todataframe**(*A*, *transfer_bytes=True*)
Convert a SciDB array to a pandas dataframe

**tosparse**(*A*, *sparse_fmt='recarray'*, *transfer_bytes=True*)
Convert a SciDB array to a sparse representation

**var**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis variance.

see SciDBArray.var()

**wrap_array**(*scidbname*, *persistent=True*)
Create a new SciDBArray object that references an existing SciDB array

**Parameters** **scidbname** : string

Wrap an existing scidb array referred to by *scidbname*. The SciDB array object persistent value will be set to True, and the object shape, datashape and data type values will be determined by the SciDB array.

**persistent** : boolean

If True (default) then array will not be deleted when this variable goes out of scope. Warning: if persistent is set to False, data could be lost!

**zeros**(*shape*, *dtype='double'*, ***kwargs*)
Return an array of zeros

**Parameters** **shape** : tuple or int

The shape of the array

**dtype** : string or list

The data type of the array

****kwargs :**

Additional keyword arguments are passed to SciDBDataShape.

**Returns** arr: SciDBArray

A SciDBArray consisting of all zeros.

### 3.2.2 Shim Interface

**class** scidbpy.**SciDBShimInterface**(*hostname*)
HTTP interface to SciDB via shim [1]_

**Parameters** **hostname** : string

A URL pointing to a running shim/SciDB session

**[1] https://github.com/Paradigm4/shim**

#### Attributes

| |
|---|
| afl |

#### Methods

| | |
|---|---|
| acos(A) | Element-wise trigonometric inverse cosine |
| approxdc(A[, index, scidb_syntax]) | Array or axis unique element estimate. |
| arange([start,] stop[, step,][, dtype]) | Return evenly spaced values within a given interval. |
| asin(A) | Element-wise trigonometric inverse sine |
| atan(A) | Element-wise trigonometric inverse tangent |
| avg(A[, index, scidb_syntax]) | Array or axis average. |
| cos(A) | Element-wise trigonometric cosine |
| count(A[, index, scidb_syntax]) | Array or axis count. |
| cross_join(A, B, *dims) | Perform a cross-join on arrays A and B. |
| dot(A, B) | Compute the matrix product of A and B |
| exp(A) | Element-wise natural exponent |
| from_array(A[, instance_id]) | Initialize a scidb array from a numpy array |
| from_dataframe(A[, instance_id]) | Initialize a scidb array from a pandas dataframe |
| from_sparse(A[, instance_id]) | Initialize a scidb array from a sparse array |
| identity(n[, dtype, sparse]) | Return a 2-dimensional square identity matrix of size n |
| join(*args) | Perform a series of array joins on the arguments and return the result. |
| linspace(start, stop[, num, endpoint, retstep]) | Return evenly spaced numbers over a specified interval. |
| list_arrays([parsed, n]) | List the arrays currently in the database |
| log(A) | Element-wise natural logarithm |
| log10(A) | Element-wise base-10 logarithm |
| max(A[, index, scidb_syntax]) | Array or axis maximum. |
| mean(A[, index, scidb_syntax]) | Array or axis mean. |
| merge(A, B) | Merge two arrays |
| min(A[, index, scidb_syntax]) | Array or axis minimum. |
| new_array([shape, dtype, persistent]) | Create a new array, either instantiating it in SciDB or simply reserving the name |
| ones(shape[, dtype]) | Return an array of ones |
| query(query, *args, **kwargs) | Perform a query on the database. |
| randint(shape[, dtype, lower, upper, persistent]) | Return an array of random integers between lower and upper |
| random(shape[, dtype, lower, upper, persistent]) | Return an array of random floats between lower and upper |
| reap() | Reap all arrays created via new_array |
| sin(A) | Element-wise trigonometric sine |
| std(A[, index, scidb_syntax]) | Array or axis standard deviation. |
| stdev(A[, index, scidb_syntax]) | Array or axis standard deviation. |
| substitute(A, value) | Replace null values in an array |
| sum(A[, index, scidb_syntax]) | Array or axis sum. |
| svd(A[, return_U, return_S, return_VT]) | Compute the Singular Value Decomposition of the array A: |
| tan(A) | Element-wise trigonometric tangent |
| toarray(A[, transfer_bytes]) | Convert a SciDB array to a numpy array |
| todataframe(A[, transfer_bytes]) | Convert a SciDB array to a pandas dataframe |
| tosparse(A[, sparse_fmt, transfer_bytes]) | Convert a SciDB array to a sparse representation |
| var(A[, index, scidb_syntax]) | Array or axis variance. |
| wrap_array(scidbname[, persistent]) | Create a new SciDBArray object that references an existing SciDB |
| zeros(shape[, dtype]) | Return an array of zeros |

**acos**(*A*)
Element-wise trigonometric inverse cosine

**approxdc**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis unique element estimate.

see SciDBArray.approxdc()

**arange**([*start*], *stop*[, *step*], *dtype=None*, ***kwargs*)
Return evenly spaced values within a given interval.

Values are generated within the half-open interval `[start, stop)` (in other words, the interval including *start* but excluding *stop*). For integer arguments the behavior is equivalent to the Python range function, but returns an ndarray rather than a list.

When using a non-integer step, such as 0.1, the results will often not be consistent. It is better to use `linspace` for these cases.

**Parameters** **start** : number, optional

Start of interval. The interval includes this value. The default start value is 0.

**stop** : number

End of interval. The interval does not include this value, except in some cases where *step* is not an integer and floating point round-off affects the length of *out*.

**step** : number, optional

Spacing between values. For any output *out*, this is the distance between two adjacent values, `out[i+1] - out[i]`. The default step size is 1. If *step* is specified, *start* must also be given.

**dtype** : dtype

The type of the output array. If *dtype* is not given, it is inferred from the type of the input arguments.

****kwargs :**

Additional arguments are passed to SciDBDatashape when creating the output array.

**Returns** **arange** : SciDBArray

Array of evenly spaced values.

For floating point arguments, the length of the result is `ceil((stop - start)/step)`. Because of floating point overflow, this rule may result in the last element of *out* being greater than *stop*.

**asin**(*A*)
Element-wise trigonometric inverse sine

**atan**(*A*)
Element-wise trigonometric inverse tangent

**avg**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis average.

see `SciDBArray.avg()`

**cos**(*A*)
Element-wise trigonometric cosine

**count**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis count.

see `SciDBArray.count()`

**cross_join**(*A*, *B*, **dims*)
Perform a cross-join on arrays A and B.

**Parameters** **A, B** : SciDBArray

***dims** : tuples

The remaining arguments are tuples of dimension indices which should be joined.

**dot** (*A*, *B*)
Compute the matrix product of A and B

**Parameters** **A** : SciDBArray

A must be a two-dimensional matrix of shape (n, p)

**B** : SciDBArray

B must be a two-dimensional matrix of shape (p, m)

**Returns** **C** : SciDBArray

The wrapper of the SciDB Array, of shape (n, m), consisting of the matrix product of A and B

**exp** (*A*)
Element-wise natural exponent

**from_array** (*A*, *instance_id=0*, ***kwargs*)
Initialize a scidb array from a numpy array

**Parameters** **A** : array_like (numpy array or sparse array)

input array from which the scidb array will be created

**instance_id** : integer

the instance ID used in loading (default=0; see SciDB documentation)

****kwargs :**

Additional keyword arguments are passed to new_array()

**Returns** **arr** : SciDBArray

SciDB Array object built from the input array

**from_dataframe** (*A*, *instance_id=0*, ***kwargs*)
Initialize a scidb array from a pandas dataframe

**Parameters** **A** : pandas dataframe

data from which the scidb array will be created.

**instance_id** : integer

the instance ID used in loading (default=0; see SciDB documentation)

****kwargs :**

Additional keyword arguments are passed to new_array()

**Returns** **arr** : SciDBArray

SciDB Array object built from the input array

**from_sparse** (*A*, *instance_id=0*, ***kwargs*)
Initialize a scidb array from a sparse array

**Parameters** **A** : sparse array

sparse input array from which the scidb array will be created. Note that this array will internally be converted to COO format.

**instance_id** : integer

the instance ID used in loading (default=0; see SciDB documentation)

****kwargs :**

Additional keyword arguments are passed to new_array()

**Returns arr** : SciDBArray

SciDB Array object built from the input array

**identity**(*n*, *dtype='double'*, *sparse=False*, ***kwargs*)

Return a 2-dimensional square identity matrix of size n

**Parameters n** : integer

the number of rows and columns in the matrix

**dtype** : string or list

The data type of the array

**sparse** : boolean

specify whether to create a sparse array (default=False)

****kwargs :**

Additional keyword arguments are passed to SciDBDataShape.

**Returns arr** : SciDBArray

A SciDBArray containint an [n x n] identity matrix

**join**(**args*)

Perform a series of array joins on the arguments and return the result.

**linspace**(*start*, *stop*, *num=50*, *endpoint=True*, *retstep=False*, ***kwargs*)

Return evenly spaced numbers over a specified interval.

Returns *num* evenly spaced samples, calculated over the interval [*start*, *stop* ].

The endpoint of the interval can optionally be excluded.

**Parameters start** : scalar

The starting value of the sequence.

**stop** : scalar

The end value of the sequence, unless *endpoint* is set to False. In that case, the sequence consists of all but the last of `num + 1` evenly spaced samples, so that *stop* is excluded. Note that the step size changes when *endpoint* is False.

**num** : int, optional

Number of samples to generate. Default is 50.

**endpoint** : bool, optional

If True, *stop* is the last sample. Otherwise, it is not included. Default is True.

**retstep** : bool, optional

If True, return (*samples*, *step*), where *step* is the spacing between samples.

****kwargs :**

additional keyword arguments are passed to SciDBDataShape

**Returns samples** : SciDBArray

There are *num* equally spaced samples in the closed interval `[start, stop]` or the half-open interval `[start, stop)` (depending on whether *endpoint* is True or False).

**step** : float (only if *retstep* is True)

Size of spacing between samples.

**list_arrays** (*parsed=True*, *n=0*)
List the arrays currently in the database

**Parameters** **parsed** : boolean

If True (default), then parse the results into a dictionary of array names as keys, schema as values

**n** : integer

the maximum number of arrays to list. If n=0, then list all

**Returns** **array_list** : string or dictionary

The list of arrays. If parsed=True, then the result is returned as a dictionary.

**log** (*A*)
Element-wise natural logarithm

**log10** (*A*)
Element-wise base-10 logarithm

**max** (*A*, *index=None*, *scidb_syntax=False*)
Array or axis maximum.

see `SciDBArray.max()`

**mean** (*A*, *index=None*, *scidb_syntax=False*)
Array or axis mean.

see `SciDBArray.mean()`

**merge** (*A*, *B*)
Merge two arrays

**min** (*A*, *index=None*, *scidb_syntax=False*)
Array or axis minimum.

see `SciDBArray.min()`

**new_array** (*shape=None*, *dtype='double'*, *persistent=False*, ***kwargs*)
Create a new array, either instantiating it in SciDB or simply reserving the name for use in a later query.

**Parameters** **shape** : int or tuple (optional)

The shape of the array to create. If not specified, no array will be created and a name will simply be reserved for later use. WARNING: if shape=None and persistent=False, an error will result when the array goes out of scope, unless the name is used to create an array on the server.

**dtype** : string (optional)

the datatype of the array. This is only referenced if *shape* is specified. Default is 'double'.

**persistent** : boolean (optional)

whether the created array should be persistent, i.e. survive in SciDB past when the object wrapper goes out of scope. Default is False.

****kwargs** : (optional)

If *shape* is specified, additional keyword arguments are passed to SciDBDataShape. Otherwise, these will not be referenced.

**Returns**

**——-**

**arr** : SciDBArray

wrapper of the new SciDB array instance.

**ones** (*shape*, *dtype='double'*, ***kwargs*)

Return an array of ones

**Parameters** **shape** : tuple or int

The shape of the array

**dtype** : string or list

The data type of the array

****kwargs :**

Additional keyword arguments are passed to SciDBDataShape.

**Returns** arr: SciDBArray

A SciDBArray consisting of all ones.

**query** (*query*, **args*, ***kwargs*)

Perform a query on the database.

This wraps a query constructor which allows the creation of sophisticated SciDB queries which act on arrays wrapped by SciDBArray objects. See Notes below for details.

**Parameters** **query** : string

The query string, with curly-braces to indicate insertions

***args, **kwargs :**

Values to be inserted (see below).

**randint** (*shape*, *dtype='uint32'*, *lower=0*, *upper=2147483647*, *persistent=False*, ***kwargs*)

Return an array of random integers between lower and upper

**Parameters** **shape** : tuple or int

The shape of the array

**dtype** : string or list

The data type of the array

**lower** : float

The lower bound of the random sample (default=0)

**upper** : float

The upper bound of the random sample (default=2147483647)

**persistent** : bool

Whether the array is persistent (default=False)

****kwargs :**

Additional keyword arguments are passed to SciDBDataShape.

**Returns** arr: SciDBArray

A SciDBArray consisting of random integers, uniformly distributed between *lower* and *upper*.

**random**(*shape*, *dtype='double'*, *lower=0*, *upper=1*, *persistent=False*, ***kwargs*)
Return an array of random floats between lower and upper

**Parameters** **shape** : tuple or int

The shape of the array

**dtype** : string or list

The data type of the array

**lower** : float

The lower bound of the random sample (default=0)

**upper** : float

The upper bound of the random sample (default=1)

**persistent** : bool

Whether the new array is persistent (default=False)

****kwargs :**

Additional keyword arguments are passed to SciDBDataShape.

**Returns** arr: SciDBArray

A SciDBArray consisting of random floating point numbers, uniformly distributed between *lower* and *upper*.

**reap**()
Reap all arrays created via new_array

**sin**(*A*)
Element-wise trigonometric sine

**std**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis standard deviation.

see `SciDBArray.std()`

**stdev**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis standard deviation.

see `SciDBArray.stdev()`

**substitute**(*A*, *value*)
Replace null values in an array

See `SciDBArray.substitute()`

**sum**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis sum.

see `SciDBArray.sum()`

**svd**(*A*, *return_U=True*, *return_S=True*, *return_VT=True*)
Compute the Singular Value Decomposition of the array A:

A = U.S.V^T

**Parameters** **A** : SciDBArray

The array for which the SVD will be computed. It should be a 2-dimensional array with a single value per cell. Currently, the svd routine requires non-overlapping chunks of size 32.

**return_U, return_S, return_VT** : boolean

if any is True, then return the associated array. All are True by default

**Returns** **[U], [S], [VT]** : SciDBArrays

Arrays storing the singular values and vectors of A.

**tan**(*A*)
Element-wise trigonometric tangent

**toarray**(*A*, *transfer_bytes=True*)
Convert a SciDB array to a numpy array

**todataframe**(*A*, *transfer_bytes=True*)
Convert a SciDB array to a pandas dataframe

**tosparse**(*A*, *sparse_fmt='recarray'*, *transfer_bytes=True*)
Convert a SciDB array to a sparse representation

**var**(*A*, *index=None*, *scidb_syntax=False*)
Array or axis variance.

see SciDBArray.var()

**wrap_array**(*scidbname*, *persistent=True*)
Create a new SciDBArray object that references an existing SciDB array

**Parameters** **scidbname** : string

Wrap an existing scidb array referred to by *scidbname*. The SciDB array object persistent value will be set to True, and the object shape, datashape and data type values will be determined by the SciDB array.

**persistent** : boolean

If True (default) then array will not be deleted when this variable goes out of scope. Warning: if persistent is set to False, data could be lost!

**zeros**(*shape*, *dtype='double'*, ***kwargs*)
Return an array of zeros

**Parameters** **shape** : tuple or int

The shape of the array

**dtype** : string or list

The data type of the array

****kwargs :**

Additional keyword arguments are passed to SciDBDataShape.

**Returns** arr: SciDBArray

A SciDBArray consisting of all zeros.

## 3.3 Visualization and Analysis

scidbpy.**histogram**(*X*, *bins=10*, *att=None*, *range=None*, *plot=False*, ***kwargs*)

Build a 1D histogram from a SciDBArray.

**Parameters** **X** : SciDBArray

The array to compute a histogram for

**att** : str (optional)

The attribute of the array to consider. Defaults to the first attribute.

**bins** : int (optional)

The number of bins

**range** : [min, max] (optional)

The lower and upper limits of the histogram. Defaults to data limits.

**plot** : bool

If True, plot the results with matplotlib

**histtype** : 'bar' | 'step' (default='bar')

If plotting, the kind of hisogram to draw. See matplotlib.hist for more details.

**kwargs** : optional

Additional keywords passed to matplotlib

**Returns** (counts, edges [, artists])

- edges is a NumPy array of edge locations (length=bins+1)
- counts is the number of data betwen [edges[i], edges[i+1]] (length=bins)
- artists is a list of the matplotlib artists created if *plot=True*

CHAPTER 4

# AFL Operator Reference

The `afl` namespace provides direct access to SciDB AFL operators. More information on these operators can be found on the SciDB documentation.

AFL Operators can be used as follows:

```
sdb = connect()
x = sdb.random((3, 4))
afl = sdb.afl
query = afl.aggregate(x, 'count(*)')
query.query  # The query string
output = query.eval()  # send query to SciDB, return as a SciDB Array
```

## 4.1 Operator Reference

`scidbpy.afl.`**`register_operator`**(*entry*, *interface=None*)

Create a new AFL operator, based on a description dictionary

**Parameters** **entry: dict with the following keys:**

- name : string giving the name of the operator
- doc : docstring
- signature : list giving the type of each argument

**interface** : SciDBinterface instance

Which SciDBinterface instance to bind the operator class to

**Returns** A new AFLExpression subclass, representing the operator

**class** `scidbpy.afl.`**`AFLNamespace`**(*interface*)

A module-like namespace for all registered operators.

#### Methods

| | |
|---|---|
| `count`(array) | |
| `papply`(array, attr, expression) | Shorthand for project(apply(array, attr, expression), attr) |
| `quote`(val) | Wrap the argument in single quotes. |
| `redimension_store`(arr_in, arr_out) | |

**papply**(*array*, *attr*, *expression*)
Shorthand for project(apply(array, attr, expression), attr)

**quote**(*val*)
Wrap the argument in single quotes.

Useful for AFL operators which expected quoted strings

**class** scidbpy.afl.**adddim**(*srcArray*, *newDimName*)
Produces a result array with one more dimension than the source array.

**Parameters - srcArray: a source array with srcAttrs and srcDims.**

- newDimName: the name of a new dimension.

### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•adddim(A, loc) <quantity: uint64, sales: double> [loc, year, item] =

loc, year, item, quantity, sales 0, 2011, 2, 7, 31.64 0, 2011, 3, 6, 19.98 0, 2012, 1, 5, 41.65 0, 2012, 2, 9, 40.68 0, 2012, 3, 8, 26.64

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**aggregate**(**args*)

**aggregate( srcArray {, AGGREGATE_CALL}+ {, groupbyDim}* )**

AGGREGATE_CALL := AGGREGATE_FUNC(inputAttr) [as resultName] AGGREGATE_FUNC := approxdc | avg | count | max | min | sum | stdev

var | some_use_defined_aggregate_function

Calculates aggregates over groups of values in an array, given the aggregate types and attributes to aggregate on.

**Parameters - srcArray: a source array with srcAttrs and srcDims.**

- 1 or more aggregate calls. Each aggregate call has an AGGREGATE_FUNC, an inputAttr and a resultName. The default resultName is inputAttr followed by '_' and then AGGREGATE_FUNC. For instance, the default resultName for sum(sales) is 'sales_sum'. The count aggregate may take * as the input attribute, meaning to count all the items in the group including null items. The default resultName for count(*) is 'count'.
- 0 or more dimensions that together determines the grouping criteria.

#### Notes

•All the aggregate functions ignore null values, except count(*).

#### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•aggregate(A, count(*), max(quantity), sum(sales), year) <count: uint64, quantity_max: uint64, sales_sum: double> [year] =

year, count, quantity_max, sales_sum 2011, 2, 7, 51.62 2012, 3, 9, 108.97

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**cumulate**(**args*)

**cumulate ( inputArray {, AGGREGATE_ALL}+ [, aggrDim] )**

AGGREGATE_CALL := AGGREGATE_FUNC ( inputAttribute ) [ AS aliasName

] AGGREGATE_FUNC := approxdc | avg | count | max | min | sum | stdev

var | some_use_defined_aggregate_function

Calculates a running aggregate over some aggregate along some fluxVector (a single dimension of the inputArray).

**Parameters** **cumulate(input, sum(v) as sum_v, count(*) as cnt, I) +-I-> J| 00**

01 02 03 00 01 02 03 V +—-+—-+—-+—-+ +———+———+———+———+ 00 | 01 | | 02 | | 00 | (1, 1) | | (3, 2) | | +—-+—-+—-+—-+ +———+———+———+———+ 01 | | 03 | | 04 | 01 | | (3, 1) | | (7, 2) | +—-+—-+—-+—-+ +———+———+———+———+ 02 | 05 | | 06 | | 02 | (5, 1) | | (11, 2)| | +—-+—-+—-+—-+ +———+———+———+———+ 03 | | 07 | | 08 | 03 | | (7, 1) | | (15, 2)| +—-+—-+—-+—-+ +———+———+———+———+

#### Notes

•For now, cumulate does NOT handle input array that have overlaps.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**regrid**(**args*)

**regrid( srcArray {, blockSize}+ {, AGGREGATE_CALL}+ )**

AGGREGATE_CALL := AGGREGATE_FUNC(inputAttr) [as resultName] AGGREGATE_FUNC := approxdc | avg | count | max | min | sum | stdev

var | some_use_defined_aggregate_function

Partitions the cells in the source array into blocks (with the given blockSize in each dimension), and for each block, calculates the required aggregates.

**Parameters - srcArray: the source array with srcAttrs and srcDims.**

- A list of blockSizes, one for each dimension.
- 1 or more aggregate calls. Each aggregate call has an AGGREGATE_FUNC, an inputAttr and a resultName. The default resultName is inputAttr followed by '_' and then AGGREGATE_FUNC. For instance, the default resultName for sum(sales) is 'sales_sum'. The count aggregate may take * as the input attribute, meaning to count all the items in the group including null items. The default resultName for count(*) is 'count'.

### Notes

•Regrid does not allow a block to span chunks. So for every dimension, the chunk interval needs to be a multiple of the block size.

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**

Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval** (*out=None*, *store=True*, ***kwargs*)

Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**window**(**args*)

**window( srcArray {, leftEdge, rightEdge}+ {, AGGREGATE_CALL}+ [,**

**METHOD ] )** AGGREGATE_CALL := AGGREGATE_FUNC(inputAttr) [as resultName] AGGREGATE_FUNC := approxdc | avg | count | max | min | sum | stdev

var | some_use_defined_aggregate_function METHOD := 'materialize' | 'probe'

Produces a result array with the same size and dimensions as the source array, where each ouput cell stores some aggregate calculated over a window around the corresponding cell in the source array. A pair of window specification values (leftEdge, rightEdge) must exist for every dimension in the source and output array.

**Parameters - srcArray: a source array with srcAttrs and srcDims.**

- leftEdge: how many cells to the left of the current cell (in one dimension) are included in the window.
- rightEdge: how many cells to the right of the current cell (in one dimension) are included in the window.
- 1 or more aggregate calls. Each aggregate call has an AGGREGATE_FUNC, an inputAttr and a resultName. The default resultName is inputAttr followed by '_' and then AGGREGATE_FUNC. For instance, the default resultName for sum(sales) is 'sales_sum'. The count aggregate may take * as the input attribute, meaning to count all the items in the group including null items. The default resultName for count(*) is 'count'.
- An optional final argument that specifies how the operator is to perform its calculation. At the moment, we support two internal algorithms: 'materialize' (which materializes an entire source chunk before computing the output windows) and 'probe' (which probes the source array for the data in each window). In general, materializing the input is a more efficient strategy, but when we're using thin(...) in conjunction with window(...), we're often better off using probes, rather than materilization. This is a decision that the optimizer needs to make.

#### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•window(A, 0, 0, 1, 0, sum(quantity)) <quantity_sum: uint64> [year, item] =

year, item, quantity_sum 2011, 2, 7 2011, 3, 13 2012, 1, 5 2012, 2, 14 2012, 3, 17

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |

Continued on next page

Table 4.10 – continued from previous page

| | |
|---|---|
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval** (*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**allversions**(*srcArray*)
Creates a single array containing all versions of an existing array.

**Parameters** **- srcArray: a source array with srcAttrs and srcDims.**

#### Examples

- **Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64
- allversions(A) <quantity: uint64, sales:double> [VersionNo, year, item] =

  VersionNo, year, item, quantity, sales 1, 2011, 2, 7, 31.64 1, 2011, 3, 6, 19.98 1, 2012, 1, 5, 41.65 1, 2012, 2, 9, 40.68 1, 2012, 3, 8, 26.64

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**analyze**(*srcArray {, attr}**)

**Returns an array describing the following characteristics of the specified attributes (or all the attributes, if no attribute is**

- attribute_name
- min
- max
- distinct_count: approximate count of distinct values.
- non_null_count: the number of cells with non-null values.

**Parameters** **- srcArray: a source array with srcAttrs and srcDims.**

- 0 or more attributes.

### Notes

- If multiple attributes are specified, the ordering of the attributes in the result array is determined by the ordering of the attributes in srcAttrs.
- The value of attribute_number may be different from the number of an attribute in srcAttrs.

### Examples

- **Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64
- analyze(A) <attribute_name:string, min:string, max:string, distinct_count:uint64, non_null_count:uint64> [attribute_number] =

  attribute_number, attribute_name, min, max, distinct_count,

  **non_null_count** 0, 'quantity' '5' '9' 5, 5 1, 'sales' '19.98' '41.65' 5, 5

### Attributes

| | |
|---|---|
| `cached_result` | Return the result if already evaluated, or None. |
| `interface` | |
| `name` | The SciDB operator name, assumed to be the same as the class name |
| `query` | |

### Methods

| | |
|---|---|
| `eval`([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| `toarray`() | Return the result of the expression as a numpy array |

**`cached_result`**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**`eval`** (*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**apply**(*srcArray {*, *newAttr*, *expression}+*)
Produces a result array with new attributes and computes values for them.

**Parameters - srcArray: a source array with srcAttrs and srcDims.**

- 1 or more pairs of a new attribute and the expression to compute the values for the attribute.

### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•apply(A, unitprice, sales/quantity) <quantity: uint64, sales: double, unitprice: double> [year, item] =

year, item, quantity, sales, unitprice 2011, 2, 7, 31.64, 4.52 2011, 3, 6, 19.98, 3.33 2012, 1, 5, 41.65, 8.33 2012, 2, 9, 40.68, 4.52 2012, 3, 8, 26.64, 3.33

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**attribute_rename**(*srcArray {, srcAttr, newAttr}+*)
Produces a result array the same as srcArray, but with at least one attribute renamed.

**Parameters - srcArray: a source array with srcAttrs and srcDims.**

- 1 or more pairs of a source attribute and the new attribute to rename to.

#### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•attribute_rename(A, sales, totalsales) <quantity: uint64, totalsales:double> [year, item] =

year, item, quantity, totalsales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**attributes**(*srcArray*)
Produces a 1D result array where each cell describes one attribute of the source array.

**Parameters** **- srcArray: a source array with srcAttrs and srcDims.**

#### Examples

- **Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64
- **attributes(A) <name:string, type_id:string, nullable:bool> [No] =** No, name, type_id, nullable 0, 'quantity', 'uint64', false 1, 'sales', 'double', false

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**bernoulli**(*srcArray*, *probability*[, *seed* ])
Evaluates whether to include a cell in the result array by generating a random number and checks if it is less than probability.

**Parameters** **- srcArray: a source array with srcAttrs and srcDims.**

- probability: the probability threshold, in [0..1]
- an optional seed for the random number generator.

#### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•bernoulli(A, 0.5, 100) <quantity: uint64, sales:double> [year, item] =

year, item, quantity, sales 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 3, 8, 26.64

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**between**(*srcArray {, lowCoord}+ {, highCoord}+*)
Produces a result array from a specified, contiguous region of a source array.

**Parameters** **- srcArray: a source array with srcAttrs and srcDims.**

- the low coordinates
- the high coordinates

#### Notes

•Almost the same as subarray. The only difference is that the dimensions retain the original start/end/boundaries.

#### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•between(A, 2011, 1, 2012, 2) <quantity: uint64, sales:double> [year, item] =

year, item, quantity, sales 2011, 2, 7, 31.64 2012, 1, 5, 41.65 2012, 2, 9, 40.68

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**build**(*schemaArray | schema*, *expression*, *mustBeConstant = false*)
Produces a result array according to a given schema, and populates values based on the given expression. The schema must have a single attribute.

**Parameters** **- schemaArray | schema: an array or a schema, from which attrs and**

dims will be used by the output array.

- expression: the expression which is used to compute values for the output array.
- mustBeConstant: whether the expression must be a constant.

#### Notes

•The build operator can only take as input bounded dimensions.

#### Examples

•**Given array A <quantity: uint64> [year, item] =** year, item, quantity 2011, 2, 7 2011, 3, 6 2012, 1, 5 2012, 2, 9 2012, 3, 8

•**build(A, 0) <quantity: uint64> [year, item] =** year, item, quantity 2011, 1, 0 2011, 2, 0 2011, 3, 0 2012, 1, 0 2012, 2, 0 2012, 3, 0 Note that the cell (2011, 1), which was empty in the

source array, is populated.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval** (*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**build_sparse** (*srcArray | schema*, *expression*, *expressionIsNonEmpty*)
Produces a sparse array and assigns values to its non-empty cells. The schema must have a single attribute.

**Parameters** **- schemaArray | schema: an array or a schema, from which attrs and**

dims will be used by the output array.

- expression: the expression which is used to compute values for the non-empty cells.
- expressionIsNonEmpty: the expression which is used to compute whether a cell is not empty.

#### Notes

•The build_sparse operator can only take as input bounded dimensions.

### Examples

- **Given array A <quantity: uint64> [year, item] =** year, item, quantity 2011, 2, 7 2011, 3, 6 2012, 1, 5 2012, 2, 9 2012, 3, 8
- **build_sparse(A, 0, item!=2) <quantity: uint64> [year, item] =** year, item, quantity 2011, 1, 0 2011, 3, 0 2012, 1, 0 2012, 3, 0

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**cancel**(*queryId*)
Cancels a query by ID.

**Parameters** **- queryId: the query ID that can be obtained from the SciDB log or**

via the list() command.

### Notes

•This operator is designed for internal use.

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**cast**(*srcArray*, *schemaArray | schema*)

**Produces a result array with data from srcArray but with the provided schema. There are three primary purposes:**

- To change names of attributes or dimensions.
- To change types of attributes
- To change a non-integer dimension to an integer dimension.

- To change a nulls-disallowed attribute to a nulls-allowed attribute.

**Parameters - srcArray: a source array.**

- schemaArray | schema: an array or a schema, from which attrs and dims will be used by the output array.

### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•cast(A, <q:uint64, s:double>[y=2011:2012,2,0, i=1:3,3,0]) <q:uint64, s:double> [y, i] =

y, i, q, s 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**concat**(*srcArray1*, *srcArray2*)
Produces a result array as the concatenation of two source arrays. The concatenation is performed by the first dimension.

**Parameters - srcArray1: the first source array with srcAttrs and srcDims1.**

- srcArray2: the second source array with srcAttrs and srcDim2, where srcDim2 may differ from srcDims1 only in the start/end of the first dimension.

### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•**concat(A, A) <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64 2013, 2, 7, 31.64 2013, 3, 6, 19.98 2014, 1, 5, 41.65 2014, 2, 9, 40.68 2014, 3, 8, 26.64

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**consume**(*array*[, *numAttrsToScanAtOnce* ])
Causes array parameter to be materialized if not already. numAttrsToScanAtOnce determines the number of attributes to scan as a group. Setting this value to '1' will result in a 'vertical' scan—all chunks of the current attribute will be scanned before moving on to the next attribute. Setting this value to the number of attributes will result in a 'horizontal' scan—chunk i of every attribute will be scanned before moving on to chunk i+1

**Parameters - array: the array to consume**

- **numAttrsToScanAtOnce: optional 'stride' of the scan, default is 1** Output array (an empty array):

< ]

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**cross_join**(*leftArray*, *rightArray {*, *attrLeft*, *attrRight}**)
Calculates the cross product of two arrays, with 0 or more equality conditions on the dimensions. Assume p pairs of equality conditions exist. The result is an (m+n-p) dimensional array. From the coordinates of each cell in the result array, a single cell in leftArray and a single cell in rightArray can be located. The cell in the result array contains the concatenation of the attributes from the two source cells. If a pair of join dimensions have different lengths, the result array uses the smaller of the two.

**Parameters - leftArray: the left-side source array with leftAttrs and**

leftDims.

- rightArray: the right-side source array with rightAttrs and rightDims.
- 0 or more pairs of an attribute from leftArray and an attribute from rightArray.

#### Notes

•Joining non-integer dimensions does not work.

#### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•**Given array B <v:uint64> [k] =** k, v 1, 10 2, 20 3, 30 4, 40 5, 50

•cross_join(A, B, item, k) <quantity: uint64, sales:double, v:uint64> [year, item] =

year, item, quantity, sales, v 2011, 2, 7, 31.64, 20 2011, 3, 6, 19.98, 30 2012, 1, 5, 41.65, 10 2012, 2, 9, 40.68, 20 2012, 3, 8, 26.64, 30

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**deldim**(*srcArray*)
Produces a result array with one fewer dimension than the source array, by deleting the first dimension which must have size 1.

**Parameters** **- srcArray: a source array with dim1, dim2, ..., dim_kThe first**

dimension must have size 1.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**dimensions**(*srcArray*)
List the dimensions of the source array.

**Parameters** **- srcArray: a source array.**

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**diskinfo**
Checks disk usage.

#### Notes

•For internal usage.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**echo**(*str*)
Produces a single-element array containing the input string.

**Parameters - str: an input string.**

#### Notes

•For internal usage.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**explain_logical**(*query*, *language = 'aql'*)
Produces a single-element array containing the logical query plan.

**Parameters - query: a query string.**

- language: the language string; either 'aql' or 'afl'; default is 'aql'

#### Notes

•For internal usage.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**explain_physical**(*query*, *language = 'aql'*)
Produces a single-element array containing the physical query plan.

**Parameters** - query: a query string.

- language: the language string; either 'aql' or 'afl'; default is 'aql'

### Notes

•For internal usage.

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**filter**(*srcArray*, *expression*)
Produces a result array by filtering out (mark as empty) the cells in the source array for which the expression evaluates to False.

**Parameters** - srcArray: a source array with srcAttrs and srcDims.

- expression: an expression which takes a cell in the source array as input and evaluates to either True or False.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**help**(*operator*)
Produces a single-element array containing the help information for an operator.

**Parameters** **- operator: the name of an operator.**

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |

Continued on next page

Table 4.54 – continued from previous page

| | |
|---|---|
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**index_lookup**(**args*)

**index_lookup (input_array, index_array,** input_array.attribute_name [,output_attribute_name] [,'memory_limit=MEMORY_LIMIT'])

**The input_array may have any attributes or dimensions. The index_array must have a single dimension and a single non-**
The operator will create a new attribute, named input_attribute_name_index by default, or using the provided name, which will be the new last non-empty-tag attribute in the output array. The output attribute will be of type int64 nullable and will contain the respective coordinate of the corresponding input_attribute in index_array. If the corresponding input_attribute is null, or if no value for input_attribute exists in the index_array, the output attribute at that position shall be set to null. The output attribute shall be returned along all the input attributes in a fashion similar to the apply() operator. The operator uses some memory to cache a part of the index_array for fast lookup of values. By default, the size of this cache is limited to MEM_ARRAY_THRESHOLD. Note this is in addition to the memory already consumed by cached MemArrays as the operator is running. If a larger or smaller limit is desired, the 'memory_limit' parameter may be used. It is provided in units of megabytes and must be at least 1. The operator may be further optimized to reduce memory footprint, optimized with a more clever data distribution pattern and/or extended to use multiple index arrays at the same time.

**Parameters** **input_array <..., input_attribute: type,... > [*]**

index_array <index_attribute: type not null>

**[dimension=0:any,any,any]** input_attribute –the name of the input attribute [output_attribute_name] –the name for the output attribute if

**desired** ['memory_limit=MEMORY_LIMIT'] –the memory limit to use MB)

ticker_id, 'memory_limit=1024')

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval** (*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**input** (*schemaArray | schema*, *filename*, *instance=-2*, *format=''*, *maxErrors=0*, *shadowArray=''*)
Produces a result array and loads data from a given file, and optionally stores to shadowArray.

**Parameters - schemaArray | schema: the array schema.**

- filename: where to load data from.
- instance: which instance; default is -2. ??
- format: ??
- maxErrors: ??
- shadowArray: if provided, the result array will be written to it.

### Notes

•[comment from author] Must be called as INPUT('existing_array_name', '/path/to/file/on/instance'). ?? schema not allowed??

•This really needs to be modified by the author.

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**

Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval** (*out=None*, *store=True*, ***kwargs*)

Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**insert**(*sourceArray*, *targetArrayName*)
Inserts all data from left array into the persistent targetArray. targetArray must exist with matching dimensions and attributes. targetArray must also be mutable. The operator shall create a new version of targetArray that contains all data of the array that would have been received by merge(sourceArray, targetArrayName). In other words, new data is inserted between old data and overwrites any overlapping old values. The resulting array is then returned.

**Parameters - sourceArray the array or query that provides inserted data**

- targetArrayName: the name of the persistent array inserted into

#### Notes

Some might wonder - if this returns the same result as merge(sourceArray, targetArrayName), then why not use store(merge())? The answer is that 1. this runs a lot faster - it does not perform a full scan of

targetArray

2.this also generates less chunk headers

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**join**(*leftArray*, *rightArray*)
Combines the attributes of two arrays at matching dimension values. The two arrays must have the same dimension start coordinates, the same chunk size, and the same chunk overlap. The join result has the same dimension names as the first input. The cell in the result array contains the concatenation of the attributes from the two source cells. If a pair of join dimensions have different lengths, the result array uses the smaller of the two.

**Parameters** **- leftArray: the left-side source array with leftAttrs and**

leftDims.

- rightArray: the right-side source array with rightAttrs and rightDims.

#### Notes

•join() is a special case of cross_join() with all pairs of dimensions given.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**list**(*what='arrays'*, *showSystem=false*)

**Produces a result array and loads data from a given file, and optionally stores to shadowArray. The available things to list**

- aggregates: show all the aggregate operators.
- arrays: show all the arrays.
- chunk descriptors: show all the chunk descriptors.
- chunk map: show the chunk map.
- functions: show all the functions.
- instances: show all SciDB instances.
- libraries: show all the libraries that are loaded in the current SciDB session.
- operators: show all the operators and the libraries in which they reside.
- types: show all the datatypes that SciDB supports.
- queries: show all the active queries.

**Parameters** **- what: what to list.**

- showSystem: whether to show systems information.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**load**(*outputArray*, *filename*, *instanceId=-2*, *format=''*, *maxErrors=0*, *shadowArray=''*)
Loads data to an existing outputArray from a given file, and optionally stores to shadowArray.

**Parameters** **- outputArray: the output array to store data into.**

- filename: A path to file where to load data from.
- instanceId: positive number means an instance ID on which file will be saved. -1 means to save file on every instance. -2 - on coordinator.
- format: format in which file will be stored. Possible values are 'store', 'lcsv+', 'lsparse', 'dcsv', 'opaque', '(<custom plugin>='')'
- maxErrors: a maximum number of errors which can take place due loading. After that exception is raised.
- shadowArray: if provided a name of array where error of reading will be specified. The schema of array is the same as output array but all attribute has string data type + attribute [row_offset: int64]. It contains an error or reading every attribute with related name and row_offset - a position in file where an error was detected.

#### Notes

•Must be called as INPUT('existing_array_name', '/path/to/file/on/instance').

•This really needs to be checked by the author.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**load_library**(*library*)
Loads a SciDB plugin.

**Parameters** **- library: the name of the library to load.**

#### Notes

•A library may be unloaded using unload_library()

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |

Continued on next page

Table 4.68 – continued from previous page

| | |
|---|---|
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**load_module**(**args*)

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**lookup**(*coordArray*, *srcArray*)
Retrieves the elements from srcArray, using coordinates stored in coordArray.

**Parameters** **- coordArray: coordDims will be used as the dims in the output**

array, coordAttrs define coordinates in srcArray.

- srcArray: srcDims and srcAttrs.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)

Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**materialize**(*srcArray*, *format*)
Produces a materialized version of an source array.

**Parameters** **- srcArray: the sourcce array with srcDims and srcAttrs.**

- format: uint32, the materialize format.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**merge**(*leftArray*, *rightArray*)
Combines elements from the input arrays the following way: for each cell in the two inputs, if the cell of leftArray is not empty, the attributes from that cell are selected and placed in the output array; otherwise, the attributes from the corresponding cell in rightArray are taken. The two arrays should have the same attribute list, number of dimensions, and dimension start index. If the dimensions are not the same size, the output array uses the larger of the two.

**Parameters** **- leftArray: the left-hand-side array.**

- rightArray: the right-hand-side array.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**mstat**
Gathers mallinfo from all the instances.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**normalize**(*srcArray*)

Produces a result array by dividing each element of a 1-attribute vector by the square root of the sum of squares of the elements.

**Parameters** **- srcArray: the source array with srcAttrs and srcDims. There**

should be exactly one attribute (of double type) and exactly one dimension.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**

Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)

Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**

The SciDB operator name, assumed to be the same as the class name

**toarray**()

Return the result of the expression as a numpy array

**class** scidbpy.afl.**project**(*srcArray {*, *selectedAttr}+*)

Produces a result array that includes some attributes of the source array.

**Parameters** **- srcArray: the source array with srcAttrs and srcDims.**

- a list of at least one selectedAttrs from the source array.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**avg_rank**(*srcArray*[, *attr {*, *groupbyDim}**])
Ranks the array elements, where each element is ranked as the average of the upper bound (UB) and lower bound (LB) rankings. The LB ranking of an element E is the number of elements less than E, plus 1. The UB ranking of an element E is the number of elements less than or equal to E, plus 1.

**Parameters** **- srcArray: a source array with srcAttrs and srcDims.**

- 0 or 1 attribute to rank with. If no attribute is provided, the first attribute is used.
- an optional list of groupbyDims used to group the elements, such that the rankings are calculated within each group. If no groupbyDim is provided, the whole array is treated as one group.

### Notes

•For any element with a distinct value, its UB ranking and LB ranking are equal.

### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•avg_rank(A, sales, year) <sales:double, sales_rank: uint64> [year, item] =

year, item, sales, sales_rank 2011, 2, 31.64, 2 2011, 3, 19.98, 1 2012, 1, 41.65, 3 2012, 2, 40.68, 2 2012, 3, 26.64, 1

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**quantile**(*srcArray*, *numQuantiles*[, *attr {*, *groupbyDim}** ])
Computes the quantiles of an array, based on the ordering of attr (within each group as specified by groupbyDim, if specified). If groupbyDim is not specified, global ordering will be performed. If attr is not specified, the first attribute will be used.

**Parameters - srcArray: the source array with srcAttrs and srcDims.**

- numQuantiles: the number of quantiles.
- attr: which attribute to sort on. The default is the first attribute.
- groupbyDim: if provided, the ordering will be performed among the records in the same group.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**rank**(*srcArray*[, *attr {, groupbyDim}** ])
Computes the rankings of an array, based on the ordering of attr (within each group as specified by the list of groupbyDims, if provided). If groupbyDims is not specified, global ordering will be performed. If attr is not specified, the first attribute will be used.

**Parameters - srcArray: the source array with srcAttrs and srcDims.**

- attr: which attribute to sort on. The default is the first attribute.
- groupbyDim: if provided, the ordering will be performed among the records in the same group.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**redimension**(**args*)

**redimension( srcArray, schemaArray | schema {, AGGREGATE_CALL}* )**

AGGREGATE_CALL := AGGREGATE_FUNC(inputAttr) [as resultName] AGGREGATE_FUNC := approxdc | avg | count | max | min | sum | stdev

var | some_use_defined_aggregate_function

Produces a array using some or all of the variables of a source array, potentially changing some or all of those variables from dimensions to attributes or vice versa, and optionally calculating aggregates to be included in the new array.

**Parameters - srcArray: a source array with srcAttrs and srcDims.**

- schemaArray | schema: an array or schema from which outputAttrs and outputDims can be acquired. All the dimensions in outputDims must exist either in srcAttrs or in srcDims, with one exception. One new dimension called the synthetic dimension is allowed. All the attributes in outputAttrs, which is not the result of an aggregate, must exist either in srcAttrs or in srcDims.
- 0 or more aggregate calls. Each aggregate call has an AGGREGATE_FUNC, an inputAttr and a resultName. The default resultName is inputAttr followed by '_' and then AGGREGATE_FUNC. The resultNames must already exist in outputAttrs.

### Notes

•The synthetic dimension cannot co-exist with aggregates. That is, if there exists at least one aggregate call, the synthetic dimension must not exist.

•When multiple values are 'redimensioned' into the same cell in the output array, the collision handling depends on the schema: (a) If there exists a synthetic dimension, all the values are retained in a vector along the synthetic dimension. (b) Otherwise, for an aggregate attribute, the aggregate result of the values is stored. (c) Otherwise, an arbitrary value is picked and the rest are discarded.

•Current redimension() does not support Non-integer dimensions or data larger than memory.

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**

Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)

Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**reduce_distro**(*replicatedArray*, *partitioningSchema*)
Makes a replicated array appear as if it has the required partitioningSchema.

**Parameters** **- replicatedArray: an source array which is replicated across all**

the instances.

- partitioningSchema: the desired partitioning schema.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**remove**(*arrayToRemove*)
Drops an array.

**Parameters** **- arrayToRemove: the array to drop.**

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**rename**(*oldArray*, *newArray*)
Changes the name of an array.

**Parameters - oldArray: an existing array.**

- newArray: the new name of the array.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**repart**(*srcArray*, *schema*)
Produces a result array similar to the source array, but with different chunk sizes, different chunk overlaps, or both.

**Parameters** - srcArray: the source array with srcAttrs and srcDims.

- schema: the desired schema.

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**reshape**(*srcArray*, *schema*)
Produces a result array containing the same cells as, but a different shape from, the source array.

**Parameters** - srcArray: the source array with srcAttrs and srcDims.

- schema: the desired schema, with the same attributes as srcAttrs, but with different size and/or number of dimensions. The restriction is that the product of the dimension sizes is equal to the number of cells in srcArray.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**reverse**(*srcArray*)
Produces a result array, where the values of every dimension is reversed.

**Parameters** **- srcArray: the source array with srcAttrs and srcDims.**

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**sample**(*srcArray*, *probability*[, *seed* ])
Produces a result array containing randomly sampled chunks from srcArray.

**Parameters** **- srcArray: the source array with srcAttrs and srcDims.**

- probability: a double value from 0 to 1, as the probability that a chunk is selected.
- seed: an int64 value as the seed to the random number generator.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**save**(*srcArray*, *file*, *instanceId = -2*, *format = 'store'*)
Saves the data in an array to a file.

**Parameters** **- srcArray: the source array to save from.**

- file: the file to save to.
- instanceId: positive number means an instance ID on which file will be saved. -1 means to save file on every instance. -2 - on coordinator.
- format: format in which file will be stored. Possible values are 'store', 'lcsv+', 'lsparse', 'dcsv', 'opaque', '(<custom plugin>='')'

#### Notes

n/a Must be called as SAVE('existing_array_name', '/path/to/file/on/instance')

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

Continued on next page

Table 4.107 – continued from previous page

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**scan**(*srcArray*)
Produces a result array that is equivalent to a stored array.

**Parameters** **- srcArray: the array to scan, with srcAttrs and srcDims**

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**setopt**(*option*[, *newValue*])
Gets/Sets a config option at runtime.

**Parameters** **- option: the config option.**

- newValue: an optional new value for the config option. If provided, the option is set. Either way, the option value(s) is returned.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**sg**(*srcArray*, *partitionSchema*, *instanceId=-1*, *outputArray=''*, *isStore=true*, *offsetVector=null*)
SCATTER/GATHER distributes array chunks over the instances of a cluster. The result array is returned. It is the only operator that uses the network manager. Typically this operator is inserted by the optimizer into the physical plan.

**Parameters** **- srcArray: the source array, with srcAttrs and srcDims.**

- **partitionSchema:** 0 = psReplication, 1 = psHashPartitioned, 2 = psLocalInstance, 3 = psByRow, 4 = psByCol, 5 = psUndefined.
- **instanceId:** -2 = to coordinator (same with 0), -1 = all instances participate, 0..#instances-1 = to a particular instance. [TO-DO: The usage of instanceId, in calculating which instance a

  chunk should go to, requires further documentation.]
- outputArray: if not empty, the result will be stored into this array
- **isStore: whether to store into the specified outputArray.** [TO-DO: Donghui believes this parameter is not needed and should

  be removed.]
- **offsetVector: a vector of #dimensions values.** To calculate which instance a chunk belongs, the chunkPos is

  augmented with the offset vector before calculation.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |

Continued on next page

Table 4.113 – continued from previous page

| | |
|---|---|
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**show**(*schemaArray | schema | queryString*[, *'aql' | 'afl'* ])
Shows the schema of an array.

**Parameters** **- schemaArray | schema | queryString: an array where the schema is**

used, the schema itself or arbitrary query string

o

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**slice**(*srcArray {*, *dim*, *dimValue}**)
Produces a 'slice' of the source array, by holding zero or more dimension values constant. The result array does not include the dimensions that are used for slicing.

**Parameters** **- srcArray: the source array with srcAttrs and srcDims.**

- dim: one of the dimensions to be used for slicing.
- dimValue: the constant value in the dimension to slice.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**sort**(*srcArray {, attr [asc | desc]}* {, chunkSize}?*)
Produces a 1D array by sorting the non-empty cells of a source array.

**Parameters** **- srcArray: the source array with srcAttrs and srcDim.**

- attr: the list of attributes to sort by. If no attribute is provided, the first attribute will be used.
- asc | desc: whether ascending or descending order of the attribute should be used. The default is asc.
- chunkSize: the size of a chunk in the result array. If not provided, 1M will be used.

#### Notes

Assuming null < NaN < other values

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**sort2**(*srcArray {, attr [asc | desc]}* {, chunkSize}?*)
Produces a 1D array by sorting the non-empty cells of a source array.

**Parameters** **- srcArray: the source array with srcAttrs and srcDim.**

- attr: the list of attributes to sort by. If no attribute is provided, the first attribute will be used.
- asc | desc: whether ascending or descending order of the attribute should be used. The default is asc.
- chunkSize: the size of a chunk in the result array. If not provided, 1M will be used.

#### Notes

Assuming null < NaN < other values

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)

Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**store**(*srcArray*, *outputArray*)
Stores an array to the database. Each execution of store() causes a new version of the array to be created.

**Parameters** **- srcArray: the source array with srcAttrs and srcDim.**

- outputArray: an existing array in the database, with the same schema as srcArray.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**subarray**(*srcArray {, lowCoord}+ {, highCoord}+*)
Produces a result array from a specified, contiguous region of a source array.

**Parameters** **- srcArray: a source array with srcAttrs and srcDims.**

- the low coordinates
- the high coordinates

#### Notes

•Almost the same as between(). The only difference is that the dimensions are 'cropped'.

#### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•subarray(A, 2011, 1, 2012, 2) <quantity: uint64, sales:double> [year, item] =

year, item, quantity, sales 0, 1, 7, 31.64 1, 0, 5, 41.65 1, 1, 9, 40.68

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**substitute**(*srcArray*, *substituteArray {*, *attr}**)
Produces a result array the same as srcArray, but with null values (of selected attributes) substituted using the values in substituteArray.

**Parameters** **- srcArray: a source array with srcAttrs and srcDims, that may**

contain null values.

- substituteArray: the array from which the values may be used to substitute the null values in srcArray. It must have a single dimension which starts at 0, and a single attribute.
- An optional list of attributes to substitute. The default is to substitute all nullable attributes.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**thin**(*srcArray {*, *start*, *step}+*)
Selects regularly-spaced elements of the source array in each dimension. A (start, step) pair must be provided for every dimension.

**Parameters** **- srcArray: a source array with srcAttrs and srcDims.**

- start: the starting coordinate of a dimension.
- step: how many coordinates to advance to the next coordinate to select. A step of 1 means to select everything.

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**transpose**(*srcArray*)
Produces an array with the same data in srcArray but with the list of dimensions reversd.

**Parameters** **- srcArray: a source array with srcAttrs and srcDims.**

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**uniq**(**args*)
uniq (input_array [,'chunk_size=CHUNK_SIZE'] )

The input array must have a single attribute of any type and a single dimension. The data in the input array must be sorted and dense. The operator is built to accept the output produced by sort() with a single attribute. The output array shall have the same attribute with the dimension i starting at 0 and chunk size of 1 million. An optional chunk_size parameter may be used to set a different output chunk size. Data is compared using a simple bitwise comparison of underlying memory. Null values are discarded from the output.

**Parameters** **array <single_attribute: INPUT_ATTRIBUTE_TYPE> [single_dimension=**

*]

'chuk_size=100000'), string_attribute_index )

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**unload_library**(*library*)
Unloads a SciDB plugin.

**Parameters - library: the name of the library to unload.**

#### Notes

•This operator is the reverse of load_library().

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**old_unpack**(*srcArray*, *newDim*)
Unpacks a multi-dimensional array into a single-dimensional array, creating new attributes to represent the dimensions in the source array.

**Parameters - srcArray: a source array with srcAttrs and srcDims.**

- newDim: the name of the dimension in the result 1D array.

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**unpack**(*srcArray*, *newDim*)

Unpacks a multi-dimensional array into a single-dimensional array, creating new attributes to represent the dimensions in the source array.

**Parameters - srcArray: a source array with srcAttrs and srcDims.**

- newDim: the name of the dimension in the result 1D array.

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**

Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)

Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**

The SciDB operator name, assumed to be the same as the class name

**toarray**()

Return the result of the expression as a numpy array

**class** scidbpy.afl.**variable_window**(**args*)

**variable_window( srcArray, dim, leftEdge, rightEdge {,**

**AGGREGATE_CALL}+ )** AGGREGATE_CALL := AGGREGATE_FUNC(inputAttr) [as resultName] AGGREGATE_FUNC := approxdc | avg | count | max | min | sum | stdev

var | some_use_defined_aggregate_function

Produces a result array with the same dimensions as the source array, where each cell stores some aggregates calculated over a 1D window covering the current cell. The window has fixed number of non-empty elements. For instance, when rightEdge is 1, the window extends to the right-hand side however number of coordinatesthat are needed, to cover the next larger non-empty cell.

**Parameters - srcArray: a source array with srcAttrs and srcDims.**

- dim: along which dimension is the window defined.
- leftEdge: how many cells to the left of the current cell are included in the window.
- rightEdge: how many cells to the right of the current cell are included in the window.
- 1 or more aggregate calls. Each aggregate call has an AGGREGATE_FUNC, an inputAttr and a resultName. The default resultName is inputAttr followed by '_' and then AGGREGATE_FUNC.

### Notes

•For a dense array, this is a special case of window().

•For the aggregate function approxdc(), the attribute name is currently non-conventional. It is xxx_ApproxDC instead of xxx_approxdc. Should change.

### Examples

•**Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64

•variable_window(A, item, 1, 0, sum(quantity)) <quantity_sum: uint64> [year, item] =

year, item, quantity_sum 2011, 2, 7 2011, 3, 13 2012, 1, 5 2012, 2, 14 2012, 3, 17

### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**

Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**versions**(*srcArray*)
Lists all versions of an array in the database.

**Parameters** **- srcArray: a source array.**

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**xgrid**(*srcArray {, scale}+*)
Produces a result array by 'scaling up' the source array. Within each dimension, the operator duplicates each cell a specified number of times before moving to the next cell. A scale must be provided for every dimension.

**Parameters** **- srcArray: a source array with srcAttrs and srcDims.**

- scale: for each dimension, a scale is provided telling how much larger the dimension should grow.

#### Examples

- **Given array A <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 2, 7, 31.64 2011, 3, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 9, 40.68 2012, 3, 8, 26.64
- **xgrid(A, 1, 2) <quantity: uint64, sales:double> [year, item] =** year, item, quantity, sales 2011, 3, 7, 31.64 2011, 4, 7, 31.64 2011, 5, 6, 19.98 2011, 6, 6, 19.98 2012, 1, 5, 41.65 2012, 2, 5, 41.65 2012, 3, 9, 40.68 2012, 4, 9, 40.68 2012, 5, 8, 26.64 2012, 6, 8, 26.64

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**abs**(**args*)
The scalar function abs

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**acos**(**args*)
The scalar function acos

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**and**(**args*)
The scalar function and

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**append_offset**(**args*)
The scalar function append_offset

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**apply_offset**(**args*)
The scalar function apply_offset

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**asin**(**args*)
The scalar function asin

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**atan**(**args*)
The scalar function atan

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**ceil**(**args*)
The scalar function ceil

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**cos**(**args*)
The scalar function cos

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**day_of_week**(**args*)
The scalar function day_of_week

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**exp**(**args*)
The scalar function exp

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**first_index**(**args*)
The scalar function first_index

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**floor**(**args*)
The scalar function floor

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**format**(**args*)
The scalar function format

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**get_offset**(**args*)
The scalar function get_offset

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**high**(**args*)
The scalar function high

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**hour_of_day**(**args*)
The scalar function hour_of_day

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**iif**(**args*)
The scalar function iif

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**instanceid**(**args*)
The scalar function instanceid

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**is_nan**(**args*)
The scalar function is_nan

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**is_null**(**args*)
The scalar function is_null

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**last_index**(**args*)
The scalar function last_index

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**length**(**args*)
The scalar function length

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**log**(**args*)
The scalar function log

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |

Continued on next page

Table 4.192 – continued from previous page

| |
|---|
| query |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**log10**(**args*)
The scalar function log10

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**low**(**args*)
The scalar function low

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**max**(**args*)
The scalar function max

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**min**(**args*)
The scalar function min

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**missing**(**args*)
The scalar function missing

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**missing_reason**(**args*)
The scalar function missing_reason

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**not**(**args*)
The scalar function not

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**now**(**args*)
The scalar function now

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**or**(**args*)
The scalar function or

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**pow**(**args*)
The scalar function pow

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**random**(**args*)
The scalar function random

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |

Continued on next page

Table 4.214 – continued from previous page

| |
|---|
| query |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**regex**(**args*)
The scalar function regex

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**sin**(**args*)
The scalar function sin

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**sqrt**(**args*)
The scalar function sqrt

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**strchar**(**args*)
The scalar function strchar

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**strftime**(**args*)
The scalar function strftime

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**strip_offset**(**args*)
The scalar function strip_offset

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**strlen**(**args*)
The scalar function strlen

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**substr**(**args*)
The scalar function substr

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**tan**(**args*)
The scalar function tan

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**togmt**(**args*)
The scalar function togmt

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**tznow**(**args*)
The scalar function tznow

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |

Continued on next page

Table 4.236 – continued from previous page

| |
|---|
| query |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**as_**(**args*)
The operator as

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**add**(**args*)
The operator +

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**sub**(**args*)
The operator -

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**mul**(**args*)
The operator *

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**div**(**args*)
The operator /

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**mod**(**args*)
The operator %

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**lt**(**args*)
The operator <

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**le**(**args*)
The operator <=

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**ne**(**args*)
The operator <>

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**eq**(**args*)
The operator =

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**ge**(**args*)
The operator >=

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |

Continued on next page

Table 4.258 – continued from previous page

| |
|---|
| query |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

**class** scidbpy.afl.**gt**(**args*)
The operator >

#### Attributes

| | |
|---|---|
| cached_result | Return the result if already evaluated, or None. |
| interface | |
| name | The SciDB operator name, assumed to be the same as the class name |
| query | |

#### Methods

| | |
|---|---|
| eval([out, store]) | Evaluate the expression if necessary, and return the result as a SciDBArray. |
| toarray() | Return the result of the expression as a numpy array |

**cached_result**
Return the result if already evaluated, or None.

**Returns** A SciDBArray instance, or None

**eval**(*out=None*, *store=True*, ***kwargs*)
Evaluate the expression if necessary, and return the result as a SciDBArray.

**Parameters** **out** : SciDBArray instance (optional)

The array to store the result in. One will be created if necessary

**store** : bool

If True (the default), the query will be wrapped in a store call, and wrapped in a SciDBAarray (specified by *out*). If false, this executes the query, but doesn't save the result

**Returns** **out** : SciDBArray instance, or None

#### Notes

The result of eval() is cached, so subsequent calls will not trigger additional database computation

**name**
The SciDB operator name, assumed to be the same as the class name

**toarray**()
Return the result of the expression as a numpy array

CHAPTER 5

# Indices and tables

- *genindex*
- *modindex*
- *search*

## S