

---

# **DistArray Documentation**

***Release 0.5.0***

**IPython Development Team and Enthought, Inc.**

July 08, 2015



<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Building the docs</b>	<b>5</b>
<b>3</b>	<b>Getting Started</b>	<b>7</b>
<b>4</b>	<b>History</b>	<b>9</b>
<b>5</b>	<b>Other Documentation</b>	<b>11</b>
5.1	DistArray API Reference . . . . .	11
5.2	Building HDF5 and h5py for DistArray . . . . .	45
5.3	Notes on building environment-modules . . . . .	46
5.4	Licence for <i>six.py</i> version 1.5.2 . . . . .	48
<b>6</b>	<b>Release Notes</b>	<b>51</b>
6.1	DistArray 0.2: development release . . . . .	51
6.2	DistArray 0.3: development release . . . . .	52
6.3	DistArray 0.4 development release . . . . .	54
6.4	DistArray 0.5 release . . . . .	56
<b>7</b>	<b>Indices and tables</b>	<b>59</b>
	<b>Bibliography</b>	<b>61</b>
	<b>Python Module Index</b>	<b>63</b>



*Think globally, act locally.*

DistArray provides general multidimensional NumPy-like distributed arrays to Python. It intends to bring the strengths of NumPy to data-parallel high-performance computing. DistArray has a similar API to [NumPy](#).

DistArray is ready for real-world testing and deployment; however, the project is still evolving rapidly, and we appreciate continued input from the scientific-Python community.

DistArray is for users who

- know and love Python and NumPy,
- want to scale NumPy to larger distributed datasets,
- want to interactively play with distributed data but also
- want to run batch-oriented distributed programs;
- want an easier way to drive and coordinate existing MPI-based codes,
- have a lot of data that may already be distributed,
- want a global view (“think globally”) with local control (“act locally”),
- need to tap into existing parallel libraries like Trilinos, PETSc, or Elemental,
- want the interactivity of IPython and the performance of MPI.

DistArray is designed to work with other packages that implement the [Distributed Array Protocol](#).



---

## Installation

---

Dependencies for DistArray:

- NumPy
- IPython
- MPI4Py

Optional dependencies:

- For HDF5 IO: h5py built against a parallel-enabled build of HDF5
- For plotting: matplotlib

If you have the above, you should be able to install this package with:

```
python setup.py install
```

or:

```
python setup.py develop
```

To run the tests, you will need to start an IPython.parallel cluster. You can use `ipcluster`, or you can use the `dacluster` command which comes with DistArray:

```
dacluster start
```

You should then be able to run all the tests from the DistArray source directory with:

```
make test
```

or from anywhere with:

```
python -m distarray.run_tests
```





---

## Building the docs

---

Dependencies to build the documentation:

- Sphinx
- sphinxcontrib.napoleon
- sphinxcontrib.programoutput

If you have the dependencies listed above, and you want to build the documentation (also available at <http://distarray.readthedocs.org>), navigate to the `docs` subdirectory of the DistArray source and use the Makefile there.

For example, to build the html documentation:

```
make html
```

from the `docs` directory.

Try:

```
make help
```

for more options.



---

# Getting Started

---

To see some initial examples of what DistArray can do, check out the IPython notebooks and python scripts in the `examples` directory. To start, see the `features` notebook, also viewable on nbviewer.

If you have questions or would like to contribute, contact us

- on the DistArray mailing list: [distarray.googlegroups.com](http://distarray.googlegroups.com),
- in the DistArray IRC channel: `#distarray` on freenode, or
- through the DistArray GitHub repo: <https://github.com/enthought/distarray> (for bug reports and pull requests).



---

### History

---

DistArray was started by Brian Granger in 2008 and is currently being developed at Enthought by a team led by Kurt Smith, in partnership with Bill Spotz from Sandia's (Py)Trilinos project and Brian Granger and Min RK from the IPython project.



---

## Other Documentation

---

### 5.1 DistArray API Reference

#### 5.1.1 `distarray` Package

##### **DistArray: Distributed NumPy-like arrays**

Documentation is available in docstrings and online at <http://distarray.readthedocs.org>.

Check out the `examples` directory in the source distribution for several example modules and IPython notebooks using DistArray.

#### 5.1.2 `__version__` Module

Version information for the DistArray package.

#### 5.1.3 `error` Module

Exception classes for DistArray errors.

**exception** `distarray.error.ContextError`

Bases: `distarray.error DistArrayError`

Exception class when a unique Context cannot be found.

**exception** `distarray.error.DistArrayError`

Bases: `exceptions.Exception`

Base exception class for DistArray errors.

**exception** `distarray.error.DistributionError`

Bases: `distarray.error DistArrayError`

Exception class when inconsistent distributions are used.

**exception** `distarray.error.InvalidCommSizeError`

Bases: `distarray.error.MPIDistArrayError`

Exception class when a requested communicator is too large.

**exception** `distarray.error.InvalidRankError`

Bases: `distarray.error.MPIDistArrayError`

Exception class when an invalid rank is used in a communicator.

**exception** `distarray.error.MPIDistArrayError`

Bases: `distarray.error.DistArrayError`

Base exception class for MPI distribution errors.

## 5.1.4 metadata\_utils Module

Utility functions for dealing with DistArray metadata.

**exception** `distarray.metadata_utils.GridShapeError`

Bases: `exceptions.Exception`

Exception class when it is not possible to distribute the processes over the number of dimensions.

**exception** `distarray.metadata_utils.InvalidGridShapeError`

Bases: `exceptions.Exception`

Exception class when the grid shape is incompatible with the distribution or communicator.

`distarray.metadata_utils.block_cyclic_size(dim_data)`

Get a size from a block-cyclic `dim_data`.

`distarray.metadata_utils.block_size(dim_data)`

Get a size from a block distributed `dim_data`.

`distarray.metadata_utils.c_or_bc_chooser(dim_data)`

Get a size from a cyclic or block-cyclic `dim_data`.

`distarray.metadata_utils.check_grid_shape_postconditions(grid_shape, shape, dist, comm_size)`

Check `grid_shape` for reasonableness after creating it.

`distarray.metadata_utils.check_grid_shape_preconditions(shape, dist, comm_size)`

Verify various `distarray` parameters are correct before making a `grid_shape`.

`distarray.metadata_utils.cyclic_size(dim_data)`

Get a size from a cyclic `dim_data`.

`distarray.metadata_utils.distribute_block_indices(dd)`

Fill in *start* and *stop* in `dim` dict `dd`.

`distarray.metadata_utils.distribute_cyclic_indices(dd)`

Fill in *start* in `dim` dict `dd`.

`distarray.metadata_utils.distribute_indices(dd)`

Fill in index related keys in `dim` dict `dd`.

`distarray.metadata_utils.make_grid_shape(shape, dist, comm_size)`

Generate a `grid_shape` from `shape` tuple and `dist` tuple.

Does not assume that `dim_data` has `proc_grid_size` set for each dimension.

Attempts to allocate processes optimally for distributed dimensions.

### Parameters

- **shape** (*tuple of int*) – The global shape of the array.
- **dist** (*tuple of str*) – `dist_type` character per dimension.



- **comm\_size** (*int*) – Total number of processes to distribute.

**Returns** `dist_grid_shape`

**Return type** tuple of int

**Raises** `GridShapeError` – if not possible to distribute `comm_size` processes over number of dimensions.

`distarray.metadata_utils.non_dist_size(dim_data)`

Get a size from a nondistributed `dim_data`.

`distarray.metadata_utils.normalize_dim_dict(dd)`

Fill out some degenerate `dim_dicts`.

`distarray.metadata_utils.normalize_dist(dist, ndim)`

Return a tuple containing dist-type for each dimension.

**Parameters**

- **dist** (*str, list, tuple, or dict*) –
- **ndim** (*int*) –

**Returns** Contains string distribution type for each dim.

**Return type** tuple of str

## Examples

```
>>> normalize_dist({0: 'b', 3: 'c'}, 4)
('b', 'n', 'n', 'c')
```

`distarray.metadata_utils.normalize_grid_shape(grid_shape, shape, dist, comm_size)`

Adds 1s to `grid_shape` so it has `ndims` dimensions. Validates `grid_shape` tuple against the `dist` tuple and `comm_size`.

`distarray.metadata_utils.normalize_reduction_axes(axes, ndim)`

`distarray.metadata_utils.positivify(index, size)`

Check that an index is within bounds and return a positive version.

**Parameters**

- **index** (*Integral or slice*) –
- **size** (*Integral*) –

**Raises** `IndexError` – for out-of-bounds indices

`distarray.metadata_utils.sanitize_indices(indices, ndim=None, shape=None)`

Classify and sanitize `indices`.

- Wrap naked Integral, slice, or Ellipsis indices into tuples
- Classify result as ‘value’ or ‘view’
- Expand *Ellipsis* objects to slices
- If the length of the tuple-ized `indices` is < `ndim` (and it’s provided), add `slice(None)`’s to indices until `indices` is `ndim` long
- If `shape` is provided, call *positivify* on the indices

**Raises**

- `TypeError` – If *indices* is other than Integral, slice or a Sequence of these
- `IndexError` – If `len(indices) > ndim`

**Returns**

**Return type** 2-tuple of (str, n-tuple of slices and Integral values)

`distarray.metadata_utils.shapes_from_dim_data_per_rank(ddpr)`

Given a `dim_data_per_rank` object, return the shapes of the localarrays. This requires no communication.

`distarray.metadata_utils.size_chooser(dist_type)`

Get a function from a `dist_type`.

`distarray.metadata_utils.size_from_dim_data(dim_data)`

Get a size from a `dim_data`.

`distarray.metadata_utils.tuple_intersection(t0, t1)`

Compute intersection of a (start, stop, step) and a (start, stop) tuple.

Assumes all values are positive.

**Parameters**

- **`t0`** (2-tuple or 3-tuple) – Tuple of (start, stop, [step]) representing an index range
- **`t1`** (2-tuple) – Tuple of (start, stop) representing an index range

**Returns** A tightly bounded interval.

**Return type** 3-tuple or None

`distarray.metadata_utils.unstructured_size(dim_data)`

Get a size from an unstructured `dim_data`.

### 5.1.5 mpi\_engine Module

The `engine_loop` function and utilities necessary for it.

**class** `distarray.mpi_engine.Engine`

Bases: `object`

**INTERCOMM** = None

**arg\_kwarg\_proxy\_converter** (*args*, *kwargs*)

**builtin\_call** (*msg*)

**delete** (*msg*)

**engine\_make\_targets\_comm** (*msg*)

**execute** (*msg*)

**free\_comm** (*msg*)

**func\_call** (*msg*)

**is\_engine** ()

**kill** (*msg*)

Break out of the engine loop.

**parse\_msg** (*msg*)

**pull** (*msg*)

`push(msg)`

### 5.1.6 mpionly\_utils Module

Utilities for running Distarray in MPI mode.

`distarray.mpionly_utils.get_comm_world()`

`distarray.mpionly_utils.get_engines()`

Get the number of engines which must be `COMM_WORLD.size - 1` (for the client)

`distarray.mpionly_utils.get_world_rank()`

`distarray.mpionly_utils.initial_comm_setup()`

Setup client and engine intracomm, and intercomm.

`distarray.mpionly_utils.is_solo_mpi_process()`

`distarray.mpionly_utils.make_base_comm()`

Creates an intracomm consisting of all the engines. Then sets: `__main__._base_comm = comm_name`

`distarray.mpionly_utils.make_intercomm(targets=None)`

`distarray.mpionly_utils.make_targets_comm(targets)`

`distarray.mpionly_utils.push_function(context, key, func, targets=None)`

`distarray.mpionly_utils.setup_engine_comm(targets=None)`

### 5.1.7 run\_tests Module

Functions for running DistArray tests.

`distarray.run_tests.test()`

Run all DistArray tests.

### 5.1.8 testing Module

Functions used for tests.

**class** `distarray.testing.BaseContextTestCase` (*methodName='runTest'*)

Bases: `unittest.case.TestCase`

Base test class for test cases that use a Context.

Overload the `ntargets` class attribute to change the default number of engines required. A `cls.context` object will be created with `targets=range(cls.ntargets)`. Tests will be skipped if there are too few targets.

**ntargets**

*int* or *'any'*, *default=4* – If an *int*, indicates how many engines are required for this test to run. If the string *'any'*, indicates that any number of engines may be used with this test.

**ntargets = 4**

**classmethod** `setUpClass()`

**classmethod** `tearDownClass()`

**class** `distarray.testing.CommNullPasser`

Bases: `type`

Metaclass.

Applies the `comm_null_passes` decorator to every method on a generated class.

**class** `distarray.testing.DefaultContextTestCase` (*methodName='runTest'*)

Bases: `distarray.testing.BaseContextTestCase`

**classmethod** `make_context` (*targets=None*)

**class** `distarray.testing.IPythonContextTestCase` (*methodName='runTest'*)

Bases: `distarray.testing.BaseContextTestCase`

**classmethod** `make_context` (*targets=None*)

**classmethod** `setUpClass` ()

**class** `distarray.testing.MPIContextTestCase` (*methodName='runTest'*)

Bases: `distarray.testing.BaseContextTestCase`

**classmethod** `make_context` (*targets=None*)

**class** `distarray.testing.ParallelTestCase` (*methodName='runTest'*)

Bases: `unittest.case.TestCase`

Base test class for fully distributed and client-less test cases.

Overload the `comm_size` class attribute to change the default number of processes required.

**comm\_size**

*int, default=4* – Indicates how many MPI processes are required for this test to run. If fewer than `comm_size` are available, the test will be skipped.

**comm\_size = 4**

**classmethod** `setUpClass` ()

**classmethod** `tearDownClass` ()

`distarray.testing.assert_localarrays_allclose` (*l0, l1, check\_dtype=False, rtol=1e-07, atol=0*)

Call `np.testing.assert_allclose` on *l0* and *l1*.

Also, check that LocalArray properties are equal.

`distarray.testing.assert_localarrays_equal` (*l0, l1, check\_dtype=False*)

Call `np.testing.assert_equal` on *l0* and *l1*.

Also, check that LocalArray properties are equal.

`distarray.testing.check_targets` (*required, available*)

If `available < required`, raise a `SkipTest` with a nice error message.

`distarray.testing.comm_null_passes` (*fn*)

Decorator. If `self.comm` is `COMM_NULL`, pass.

This allows our tests to pass on processes that have nothing to do.

`distarray.testing.import_or_skip` (*name*)

Try importing *name*, raise `SkipTest` on failure.

**Parameters** *name* (*str*) – Module name to try to import.

**Returns** *module* – Module object imported by `importlib`.

**Return type** *module object*

**Raises** `unittest.SkipTest` – If the attempted import raises an `ImportError`.

### Examples

```
>>> h5py = import_or_skip('h5py')
>>> h5py.get_config()
<h5py.h5.H5PYConfig at 0x103dd5a78>
```

`distarray.testing.raise_typeerror(fn)`

Decorator for protocol validator functions.

These functions return (success, err\_msg), but sometimes we would rather have an exception.

`distarray.testing.temp_filepath(extension='')`

Return a randomly generated filename.

This filename is appended to the directory path returned by `tempfile.gettempdir()` and has *extension* appended to it.

## 5.1.9 utils Module

Utilities.

`distarray.utils.all_equal(iterable)`

Return True if all elements in *iterable* are equal.

Also returns True if iterable is empty.

**class** `distarray.utils.count_round_trips(client)`

Bases: object

Context manager for counting the number of roundtrips between a IPython client and controller.

**Usage:**

```
>>> with count_round_trips(client) as r:
...     send_42_messages()
```

```
>>> r.count
42
```

`update_count()`

`distarray.utils.distarray_random_getstate()`

Get the state of the global random number generator.

`distarray.utils.distarray_random_setstate(state)`

Set the state of the global random number generator.

`distarray.utils.divisors_minmax(n, dmin, dmax)`

Find the divisors of *n* in the interval (*dmin*,*dmax*].

`distarray.utils.flatten(seq, to_expand=<function list_or_tuple>)`

Flatten a nested sequence.

`distarray.utils.get_from_dotted_name(dotted_name)`

`distarray.utils.has_exactly_one(iterable)`

Does *iterable* have exactly one non-None element?

`distarray.utils.list_or_tuple(seq)`

Is the object either a list or a tuple?

`distarray.utils.mirror_sort(seq, ref_seq)`

Sort *seq* into the order that *ref\_seq* is in.

```
>>> mirror_sort(range(5), [1, 5, 2, 4, 3])
[0, 4, 1, 3, 2]
```

`distarray.utils.mult_partitions(n, s)`

Compute the multiplicative partitions of *n* of size *s*

```
>>> mult_partitions(52, 3)
[(2, 2, 13)]
>>> mult_partitions(52, 2)
[(2, 26), (4, 13)]
```

`distarray.utils.mult_partitions_recurs(n, s, pd=0)`

`distarray.utils.multi_for(iterables)`

`distarray.utils.remove_elements(to_remove, seq)`

Return a list, with the elements with specified indices removed.

#### Parameters

- **to\_remove** (*iterable*) – Indices of elements in list to remove
- **seq** (*iterable*) – Elements in the list.

#### Returns

**Return type** List with the specified indices removed.

`distarray.utils.set_from_dotted_name(name, val)`

`distarray.utils.slice_intersection(s1, s2)`

Compute a slice that represents the intersection of two slices.

Currently only implemented for steps of size 1.

**Parameters** **s2** (*s1*,) –

#### Returns

**Return type** slice object

`distarray.utils.uid()`

Get a unique name for a distarray object.

## 5.1.10 Subpackages

### apps Package

### apps Package

Scripts for use with the DistArray package.

## dacluster Module

Start, stop and manage a IPython.parallel cluster. *dacluster* can take all the commands IPython's *ipcluster* can, and a few extras that are distarray specific.

`distarray.apps.dacluster.clear(**kwargs)`

Removes all distarray-related modules from engines' `sys.modules`.

`distarray.apps.dacluster.main()`

Main function for dacluster utility.

Either start, stop, restart, or clear is called depending on the command line arguments.

`distarray.apps.dacluster.restart(n=4, engines=None, **kwargs)`

Convenient way to restart an ipcluster.

`distarray.apps.dacluster.start(n=4, engines=None, **kwargs)`

Convenient way to start an ipcluster for testing.

Doesn't exit until the ipcluster prints a success message.

`distarray.apps.dacluster.stop(**kwargs)`

Convenient way to stop an ipcluster.

```
/home/docs/bin/python: No module named distarray.apps
```

## engine Module

Script for facilitating MPI-only mode.

Starts an MPI-process-based engine.

```
/home/docs/bin/python: No module named distarray.apps
```

## globalapi Package

### globalapi Package

Modules dealing with the global index-space view of *DistArrays*.

In other words, the view from the client.

## context Module

*Context* objects contain the information required for *DistArrays* to communicate with *LocalArrays*.

**class** `distarray.globalapi.context.BaseContext`

Bases: `object`

Context objects manage the setup and communication of the worker processes for *DistArray* objects. A *DistArray* object has a context, and contexts have an MPI intracommunicator that they use to communicate with worker processes.

Typically there is just one context object that uses all processes, although it is possible to have more than one context with a different selection of engines.

**allclose** (*a*, *b*, *rtol*=1e-05, *atol*=1e-08)

**apply** (*func*, *args=None*, *kwargs=None*, *targets=None*)

**cleanup** ()

**close** ()

**delete\_key** (*key*, *targets=None*)

Delete the specific key from all the engines.

**empty** (*shape\_or\_dist*, *dtype=<type 'float'>*)

Create an empty Distarray.

**Parameters**

- **shape\_or\_dist** (*shape tuple or Distribution object*) –
- **dtype** (*NumPy dtype, optional (default float)*) –

**Returns** A DistArray distributed as specified, with uninitialized values.

**Return type** *DistArray*

**fromarray** (*arr*, *distribution=None*)

Create a DistArray from an ndarray.

**Parameters** **distribution** (*Distribution object, optional*) – If a Distribution object is not provided, one is created with *Distribution(arr.shape)*.

**Returns** A DistArray distributed as specified, using the values and dtype from *arr*.

**Return type** *DistArray*

**fromfunction** (*function*, *shape*, *\*\*kwargs*)

Create a DistArray from a function over global indices.

Unlike numpy's *fromfunction*, the result of distarray's *fromfunction* is restricted to the same Distribution as the index array generated from *shape*.

See numpy.fromfunction for more details.

**fromndarray** (*arr*, *distribution=None*)

Create a DistArray from an ndarray.

**Parameters** **distribution** (*Distribution object, optional*) – If a Distribution object is not provided, one is created with *Distribution(arr.shape)*.

**Returns** A DistArray distributed as specified, using the values and dtype from *arr*.

**Return type** *DistArray*

**load\_dnp** (*name*)

Load a distributed array from .dnp files.

The .dnp file format is a binary format inspired by NumPy's .npy format. The header of a particular .dnp file contains information about which portion of a DistArray is saved in it (using the metadata outlined in the Distributed Array Protocol), and the data portion contains the output of NumPy's *save* function for the local array data. See the module docstring for *distarray.localapi.format* for full details.

**Parameters** **name** (*str or list of str*) – If a str, this is used as the prefix for the filename used by each engine. Each engine will load a file named *<name>\_<rank>.dnp*. If a list of str, each engine will use the name at the index corresponding to its rank. An exception is raised if the length of this list is not the same as the context's communicator's size.

**Returns** **result** – A DistArray encapsulating the file loaded on each engine.

**Return type** *DistArray*



**Raises** `TypeError` – If *name* is an iterable whose length is different from the context’s communicator’s size.

**See also:**

[`save\_dnpv\(\)`](#) Saving files to load with `load_dnpv`.

**`load_hdf5`** (*filename*, *distribution*, *key*=‘buffer’)

Load a DistArray from a dataset in an `.hdf5` file.

**Parameters**

- **`filename`** (*str*) – Filename to load.
- **`distribution`** (*Distribution object*) –
- **`key`** (*str*, *optional*) – The identifier for the group to load the DistArray from (the default is ‘buffer’).

**Returns** **result** – A DistArray encapsulating the file loaded.

**Return type** [\*DistArray\*](#)

**`load_npy`** (*filename*, *distribution*)

Load a DistArray from a dataset in a `.npv` file.

**Parameters**

- **`filename`** (*str*) – Filename to load.
- **`distribution`** (*Distribution object*) –

**Returns** **result** – A DistArray encapsulating the file loaded.

**Return type** [\*DistArray\*](#)

**`make_subcomm`** (*new\_targets*)

**`ones`** (*shape\_or\_dist*, *dtype*=<type ‘float’>)

Create a Distarray filled with ones.

**Parameters**

- **`shape_or_dist`** (*shape tuple or Distribution object*) –
- **`dtype`** (*NumPy dtype*, *optional* (default float)) –

**Returns** A DistArray distributed as specified, filled with ones.

**Return type** [\*DistArray\*](#)

**`push_function`** (*key*, *func*)

**`register`** (*func*)

Associate a function with this Context. Allows access to the local process and local data associated with each DistArray.

After registering a function with a context, the function can be called as `context.func(...)`. Doing so will call the function locally on target processes determined from the arguments passed in using `Context.apply(...)`. The function can take non-proxied Python objects, DistArrays, or other proxied objects as arguments. Non-proxied Python objects will be broadcasted to all local processes; proxied objects will be dereferenced before calling the function on the local process.

**`save_dnpv`** (*name*, *da*)

Save a distributed array to files in the `.dnpv` format.

The `.dnpy` file format is a binary format inspired by NumPy's `.npy` format. The header of a particular `.dnpy` file contains information about which portion of a DistArray is saved in it (using the metadata outlined in the Distributed Array Protocol), and the data portion contains the output of NumPy's `save` function for the local array data. See the module docstring for `distarray.localapi.format` for full details.

#### Parameters

- **name** (*str or list of str*) – If a `str`, this is used as the prefix for the filename used by each engine. Each engine will save a file named `<name>_<rank>.dnpy`. If a list of `str`, each engine will use the name at the index corresponding to its rank. An exception is raised if the length of this list is not the same as the context's communicator's size.
- **da** (`DistArray`) – Array to save to files.

**Raises** `TypeError` – If `name` is an sequence whose length is different from the context's communicator's size.

See also:

[`load\_dnpy\(\)`](#) Loading files saved with `save_dnpy`.

**save\_hdf5** (*filename, da, key='buffer', mode='a'*)

Save a DistArray to a dataset in an `.hdf5` file.

#### Parameters

- **filename** (*str*) – Name of file to write to.
- **da** (`DistArray`) – Array to save to a file.
- **key** (*str, optional*) – The identifier for the group to save the DistArray to (the default is `'buffer'`).
- **mode** (*optional, {'w', 'w-', 'a'}, default 'a'*) –
  - 'w' Create file, truncate if exists
  - 'w-' Create file, fail if exists
  - 'a' Read/write if exists, create otherwise (default)

**zeros** (*shape\_or\_dist, dtype=<type 'float'>*)

Create a Distarray filled with zeros.

#### Parameters

- **shape\_or\_dist** (*shape tuple or Distribution object*) –
- **dtype** (*NumPy dtype, optional (default float)*) –

**Returns** A DistArray distributed as specified, filled with zeros.

**Return type** `DistArray`

`distarray.globalapi.context.Context` (*\*args, \*\*kwargs*)

**exception** `distarray.globalapi.context.ContextCreationError`

Bases: `exceptions.RuntimeError`

**class** `distarray.globalapi.context.IPythonContext` (*client=None, targets=None*)

Bases: `distarray.globalapi.context.BaseContext`

Context class that uses `IPython.parallel`.

See the docstring for `BaseContext` for more information about Contexts.

See also:

*BaseContext*

**apply** (*func, args=None, kwargs=None, targets=None, autoproxyize=False*)

Analogous to IPython.parallel.view.apply\_sync

#### Parameters

- **func** (*function*) –
- **args** (*tuple*) – positional arguments to func
- **kwargs** (*dict*) – key word arguments to func
- **targets** (*sequence of integers*) – engines func is to be run on.
- **autoproxyize** (*bool, default False*) – If True, implicitly return a Proxy object from the function.

#### Returns

**Return type** return a list of the results on the each engine.

**cleanup** ()

Delete keys that this context created from all the engines.

**close** ()

**make\_subcomm** (*new\_targets*)

**push\_function** (*key, func, targets=None*)

**class** distarray.globalapi.context.**MPIContext** (*targets=None*)

Bases: *distarray.globalapi.context.BaseContext*

Context class that uses MPI only (no IPython.parallel).

See the docstring for *BaseContext* for more information about Contexts.

See also:

*BaseContext*

**INTERCOMM = None**

**apply** (*func, args=None, kwargs=None, targets=None, autoproxyize=False*)

Analogous to IPython.parallel.view.apply\_sync

#### Parameters

- **func** (*function*) –
- **args** (*tuple*) – positional arguments to func
- **kwargs** (*dict*) – keyword arguments to func
- **targets** (*sequence of integers*) – engines func is to be run on.
- **autoproxyize** (*bool, default False*) – If True, implicitly return a Proxy object from the function.

**Returns** result from each engine.

**Return type** list

**cleanup** ()

Delete keys that this context created from all the engines.

```
close()
delete_key(key, targets=None)
make_subcomm(targets)
push_function(key, func, targets=None)
```

## distarray Module

The *DistArray* data structure.

*DistArray* objects are proxies for collections of *LocalArray* objects. They are meant to roughly emulate NumPy *ndarrays*.

```
class distarray.globalapi.distarray.DistArray(distribution, dtype=<type 'float'>)
    Bases: object
```

**context**

**dist**

**dtype**

**fill**(value)

```
classmethod from_localarrays(key, context=None, targets=None, distribution=None,
                             dtype=None)
```

The caller has already created the *LocalArray* objects. *key* is their name on the engines. This classmethod creates a *DistArray* that refers to these *LocalArrays*.

Either a *context* or a *distribution* must also be provided. If *context* is provided, a *dim\_data\_per\_rank* will be pulled from the existing *LocalArrays* and a *Distribution* will be created from it. If *distribution* is provided, it should accurately reflect the distribution of the existing *LocalArrays*.

If *dtype* is not provided, it will be fetched from the engines.

```
get_localarrays()
```

Pull the *LocalArray* objects from the engines.

**Returns** one *localarray* per process

**Return type** list of *localarrays*

```
get_ndarrays()
```

Pull the local *ndarrays* from the engines.

**Returns** one *ndarray* per process

**Return type** list of *ndarrays*

**global\_size**

**grid\_shape**

**itemsizes**

```
localshapes()
```

```
max(axis=None, dtype=None, out=None)
```

Return the maximum of array elements over the given axis.

```
mean(axis=None, dtype=<type 'float'>, out=None)
```

Return the mean of array elements over the given axis.

**min** (*axis=None, dtype=None, out=None*)

Return the minimum of array elements over the given axis.

**nbytes**

**ndim**

**shape**

**std** (*axis=None, dtype=<type 'float'>, out=None*)

Return the standard deviation of array elements over the given axis.

**sum** (*axis=None, dtype=None, out=None*)

Return the sum of array elements over the given axis.

**targets**

**toarray** ()

Returns the distributed array as an ndarray.

**tondarray** ()

Returns the distributed array as an ndarray.

**var** (*axis=None, dtype=<type 'float'>, out=None*)

Return the variance of array elements over the given axis.

**view** (*dtype=None*)

New view of array with the same data.

**Parameters** **dtype** (*numpy dtype, optional*) – Data-type descriptor of the returned view, e.g., float32 or int16. The default, None, results in the view having the same data-type as the original array.

**Returns** A view on the original DistArray, optionally with the underlying memory interpreted as a different dtype.

**Return type** *DistArray*

---

**Note:** No redistribution is done. The sizes of all *LocalArrays* must be compatible with the new view.

---

## functions Module

Distributed ufuncs for *DistArrays*.

`distarray.globalapi.functions.absolute(a, *args, **kwargs)`

`distarray.globalapi.functions.arccos(a, *args, **kwargs)`

`distarray.globalapi.functions.arccosh(a, *args, **kwargs)`

`distarray.globalapi.functions.arcsin(a, *args, **kwargs)`

`distarray.globalapi.functions.arcsinh(a, *args, **kwargs)`

`distarray.globalapi.functions.arctan(a, *args, **kwargs)`

`distarray.globalapi.functions.arctanh(a, *args, **kwargs)`

`distarray.globalapi.functions.conjugate(a, *args, **kwargs)`

`distarray.globalapi.functions.cos(a, *args, **kwargs)`

`distarray.globalapi.functions.cosh(a, *args, **kwargs)`

```
distarray.globalapi.functions.exp(a, *args, **kwargs)
distarray.globalapi.functions.expm1(a, *args, **kwargs)
distarray.globalapi.functions.invert(a, *args, **kwargs)
distarray.globalapi.functions.log(a, *args, **kwargs)
distarray.globalapi.functions.log10(a, *args, **kwargs)
distarray.globalapi.functions.log1p(a, *args, **kwargs)
distarray.globalapi.functions.negative(a, *args, **kwargs)
distarray.globalapi.functions.reciprocal(a, *args, **kwargs)
distarray.globalapi.functions rint(a, *args, **kwargs)
distarray.globalapi.functions.sign(a, *args, **kwargs)
distarray.globalapi.functions.sin(a, *args, **kwargs)
distarray.globalapi.functions.sinh(a, *args, **kwargs)
distarray.globalapi.functions.sqrt(a, *args, **kwargs)
distarray.globalapi.functions.square(a, *args, **kwargs)
distarray.globalapi.functions.tan(a, *args, **kwargs)
distarray.globalapi.functions.tanh(a, *args, **kwargs)
distarray.globalapi.functions.add(a, b, *args, **kwargs)
distarray.globalapi.functions.arctan2(a, b, *args, **kwargs)
distarray.globalapi.functions.bitwise_and(a, b, *args, **kwargs)
distarray.globalapi.functions.bitwise_or(a, b, *args, **kwargs)
distarray.globalapi.functions.bitwise_xor(a, b, *args, **kwargs)
distarray.globalapi.functions.divide(a, b, *args, **kwargs)
distarray.globalapi.functions.floor_divide(a, b, *args, **kwargs)
distarray.globalapi.functions.fmod(a, b, *args, **kwargs)
distarray.globalapi.functions.hypot(a, b, *args, **kwargs)
distarray.globalapi.functions.left_shift(a, b, *args, **kwargs)
distarray.globalapi.functions.mod(a, b, *args, **kwargs)
distarray.globalapi.functions.multiply(a, b, *args, **kwargs)
distarray.globalapi.functions.power(a, b, *args, **kwargs)
distarray.globalapi.functions.remainder(a, b, *args, **kwargs)
distarray.globalapi.functions.right_shift(a, b, *args, **kwargs)
distarray.globalapi.functions.subtract(a, b, *args, **kwargs)
distarray.globalapi.functions.true_divide(a, b, *args, **kwargs)
distarray.globalapi.functions.less(a, b, *args, **kwargs)
distarray.globalapi.functions.less_equal(a, b, *args, **kwargs)
distarray.globalapi.functions.equal(a, b, *args, **kwargs)
```

```
distarray.globalapi.functions.not_equal (a, b, *args, **kwargs)
distarray.globalapi.functions.greater (a, b, *args, **kwargs)
distarray.globalapi.functions.greater_equal (a, b, *args, **kwargs)
```

### ipython\_cleanup Module

Functions for cleaning up *DistArray* objects from IPython parallel engines.

```
distarray.globalapi.ipython_cleanup.cleanup (view, module_name, prefix)
    Delete Context object with the given name from the given module

distarray.globalapi.ipython_cleanup.cleanup_all (module_name, prefix)
    Connects to all engines and runs cleanup() on them.

distarray.globalapi.ipython_cleanup.clear (view)
    Removes all distarray-related modules from engines' sys.modules.

distarray.globalapi.ipython_cleanup.clear_all ()

distarray.globalapi.ipython_cleanup.get_local_keys (view, prefix)
    Returns a dictionary of keyname -> target_list mapping for all names that start with prefix on engines in view.
```

### ipython\_utils Module

The single IPython entry point.

### maps Module

Distribution class and auxiliary ClientMap classes.

The *Distribution* is a multi-dimensional map class that manages the one-dimensional maps for each *DistArray* dimension. The *Distribution* class represents the *distribution* information for a distributed array, independent of the distributed array's *data*. Distributions allow *DistArrays* to reduce overall communication when indexing and slicing by determining which processes own (or may possibly own) the indices in question. Two *DistArray* objects can share the same *Distribution* if they have the exact same distribution.

The one-dimensional *ClientMap* classes keep track of which process owns which index in that dimension. This class has several subclasses for specific distribution types, including *BlockMap*, *CyclicMap*, *NoDistMap*, and *UnstructuredMap*.

```
class distarray.globalapi.maps.BlockCyclicMap (size, grid_size, block_size=1)
    Bases: distarray.globalapi.maps.MapBase

    dist = 'c'

    classmethod from_axis_dim_dicts (axis_dim_dicts)

    classmethod from_global_dim_dict (glb_dim_dict)

    get_dimdicts ()

    index_owners (idx)

    is_compatible (other)
```

```
class distarray.globalapi.maps.BlockMap (size, grid_size, bounds=None, comm_padding=None,
                                         boundary_padding=None)
```

Bases: *distarray.globalapi.maps.MapBase*

**dist** = 'b'

**classmethod from\_axis\_dim\_dicts** (axis\_dim\_dicts)

**classmethod from\_global\_dim\_dict** (glb\_dim\_dict)

**get\_dimdicts** ()

**index\_owners** (idx)

**is\_compatible** (other)

**slice** (idx)

Make a new Map from a slice.

**slice\_owners** (idx)

**view** (new\_dimsize)

Scale this map for the view method.

```
class distarray.globalapi.maps.Distribution
```

Bases: object

Governs the mapping between global indices and process ranks for multi-dimensional objects.

**classmethod from\_dim\_data\_per\_rank** (context, dim\_data\_per\_rank, targets=None)

Create a Distribution from a sequence of *dim\_data* tuples.

#### Parameters

- **context** (*Context object*) –
- **dim\_data\_per\_rank** (*Sequence of dim\_data tuples, one per rank*) – See the “Distributed Array Protocol” for a description of *dim\_data* tuples.
- **targets** (*Sequence of int, optional*) – Sequence of engine target numbers. Default: all available

#### Returns

Return type *Distribution*

**classmethod from\_global\_dim\_data** (context, global\_dim\_data, targets=None)

Make a Distribution from a *global\_dim\_data* structure.

#### Parameters

- **context** (*Context object*) –
- **global\_dim\_data** (*tuple of dict*) – A global dimension dictionary per dimension. See following *Note* section.
- **targets** (*Sequence of int, optional*) – Sequence of engine target numbers. Default: all available

#### Returns

Return type *Distribution*

---

**Note:** The *global\_dim\_data* tuple is a simple, straightforward data structure that allows full control over all aspects of a DistArray’s distribution information. It does not contain any of the array’s *data*, only the *metadata* needed to specify how the array is to be distributed. Each dimension of the array is represented by corresponding dictionary in the tuple, one per dimension. All dictionaries have a *dist\_type* key that



specifies whether the array is block, cyclic, or unstructured. The other keys in the dictionary are dependent on the *dist\_type* key.

### Block

- *dist\_type* is 'b'.
- *bounds* is a sequence of integers, at least two elements.

The *bounds* sequence always starts with 0 and ends with the global *size* of the array. The other elements indicate the local array global index boundaries, such that successive pairs of elements from *bounds* indicates the *start* and *stop* indices of the corresponding local array.

- *comm\_padding* integer, greater than or equal to zero.
- *boundary\_padding* integer, greater than or equal to zero.

These integer values indicate the communication or boundary padding, respectively, for the local arrays. Currently only a single value for both *boundary\_padding* and *comm\_padding* is allowed for the entire dimension.

### Cyclic

- *dist\_type* is 'c'
- *proc\_grid\_size* integer, greater than or equal to one.

The size of the process grid in this dimension. Equivalent to the number of local arrays in this dimension and determines the number of array sections.

- *size* integer, greater than or equal to zero.

The global size of the array in this dimension.

- *block\_size* integer, optional. Greater than or equal to one.

If not present, equivalent to being present with value of one.

### Unstructured

- *dist\_type* is 'u'
- *indices* sequence of one-dimensional numpy integer arrays or buffers.

The `len(indices)` is the number of local unstructured arrays in this dimension.

To compute the global size of the array in this dimension, compute `sum(len(ii) for ii in indices)`.

### Not-distributed

The 'n' distribution type is a convenience to specify that an array is not distributed along this dimension.

- *dist\_type* is 'n'
- *size* integer, greater than or equal to zero.

The global size of the array in this dimension.

---

**classmethod `from_maps`** (*context*, *maps*, *targets=None*)

Create a Distribution from a sequence of *Maps*.

#### Parameters

- **context** (*Context object*) –
- **maps** (*Sequence of Map objects*) –

- **targets** (*Sequence of int, optional*) – Sequence of engine target numbers. Default: all available

### Returns

Return type *Distribution*

**get\_dim\_data\_per\_rank** ()

**has\_precise\_index**

Does the client-side Distribution know precisely who owns all indices?

This can be used to determine whether one needs to use the *checked* version of `__getitem__` or `__setitem__` on LocalArrays.

**is\_compatible** (*o*)

**localshapes** ()

**owning\_ranks** (*idxs*)

Returns a list of ranks that may *possibly* own the location in the *idxs* tuple.

For many distribution types, the owning rank is precisely known; for others, it is only probably known. When the rank is precisely known, *owning\_ranks()* returns a list of exactly one rank. Otherwise, returns a list of more than one rank.

If the *idxs* tuple is out of bounds, raises *IndexError*.

**owning\_targets** (*idxs*)

Like *owning\_ranks()* but returns a list of targets rather than ranks.

Convenience method meant for IPython parallel usage.

**reduce** (*axes*)

Returns a new Distribution reduced along *axis*, i.e., the new distribution has one fewer dimension than *self*.

**slice** (*index\_tuple*)

Make a new Distribution from a slice.

**view** (*new\_dimsize=None*)

Generate a new Distribution for use with *DistArray.view*.

**class** *distarray.globalapi.maps.MapBase*

Bases: *object*

Base class for one-dimensional client-side maps.

Maps keep track of the relevant distribution information for a single dimension of a distributed array. Maps allow distributed arrays to keep track of which process to talk to when indexing and slicing.

Classes that inherit from *MapBase* must implement the *index\_owners()* abstractmethod.

**classmethod** *from\_axis\_dim\_dicts* (*axis\_dim\_dicts*)

Make a Map from a sequence of process-local dimension dictionaries.

There should be one such dimension dictionary per process.

**classmethod** *from\_global\_dim\_dict* (*glb\_dim\_dict*)

Make a Map from a global dimension dictionary.

**get\_dimdicts** ()

Return a *dim\_dict* per process in this dimension.

**index\_owners** (*idx*)

Returns a list of process IDs in this dimension that might possibly own *idx*.

Raises *IndexError* if *idx* is out of bounds.

**is\_compatible** (*map*)

**class** `distarray.globalapi.maps.NoDistMap` (*size, grid\_size*)

Bases: `distarray.globalapi.maps.MapBase`

**dist** = 'n'

**classmethod** `from_axis_dim_dicts` (*axis\_dim\_dicts*)

**classmethod** `from_global_dim_dict` (*glb\_dim\_dict*)

**get\_dimdicts** ()

**index\_owners** (*idx*)

**is\_compatible** (*other*)

**slice** (*idx*)

Make a new Map from a slice.

**slice\_owners** (*idx*)

**view** (*new\_dimsize*)

Scale this map for the *view* method.

**class** `distarray.globalapi.maps.UnstructuredMap` (*size, grid\_size, indices=None*)

Bases: `distarray.globalapi.maps.MapBase`

**dist** = 'u'

**classmethod** `from_axis_dim_dicts` (*axis\_dim\_dicts*)

**classmethod** `from_global_dim_dict` (*glb\_dim\_dict*)

**get\_dimdicts** ()

**index\_owners** (*idx*)

`distarray.globalapi.maps.asdistribution` (*context*, *shape\_or\_dist*, *dist=None*,  
*grid\_shape=None, targets=None*)

`distarray.globalapi.maps.choose_map` (*dist\_type*)

Choose a map class given one of the distribution types.

`distarray.globalapi.maps.map_from_global_dim_dict` (*global\_dim\_dict*)

Given a *global\_dim\_dict* return map.

`distarray.globalapi.maps.map_from_sizes` (*size, dist\_type, grid\_size*)

Returns an instance of the appropriate subclass of *MapBase*.

## random Module

Contains the *Random* class that emulates *numpy.random* for *DistArray*.

**class** `distarray.globalapi.random.Random` (*context*)

Bases: `object`

**normal** (*shape\_or\_dist, loc=0.0, scale=1.0*)

Draw random samples from a normal (Gaussian) distribution.

The probability density function of the normal distribution, first derived by De Moivre and 200 years later by both Gauss and Laplace independently <sup>1</sup>, is often called the bell curve because of its characteristic shape (see the example below).

The normal distributions occurs often in nature. For example, it describes the commonly occurring distribution of samples influenced by a large number of tiny, random disturbances, each with its own unique distribution <sup>2</sup>.

#### Parameters

- **loc** (*float*) – Mean (“centre”) of the distribution.
- **scale** (*float*) – Standard deviation (spread or “width”) of the distribution.
- **shape\_or\_dist** (*shape tuple or Distribution object*) –

#### Notes

The probability density for the Gaussian distribution is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where  $\mu$  is the mean and  $\sigma$  the standard deviation. The square of the standard deviation,  $\sigma^2$ , is called the variance.

The function has its peak at the mean, and its “spread” increases with the standard deviation (the function reaches 0.607 times its maximum at  $x + \sigma$  and  $x - \sigma$ ). This implies that `numpy.random.normal` is more likely to return samples lying close to the mean, rather than those far away.

#### References

**rand** (*shape\_or\_dist*)

Random values over a given distribution.

Create a distarray of the given shape and propagate it with random samples from a uniform distribution over  $[0, 1)$ .

**Parameters** **shape\_or\_dist** (*shape tuple or Distribution object*) –

**Returns** **out** – Random values.

**Return type** *DistArray*

**randint** (*shape\_or\_dist, low, high=None*)

Return random integers from *low* (inclusive) to *high* (exclusive).

Return random integers from the “discrete uniform” distribution in the “half-open” interval  $[low, high)$ . If *high* is None (the default), then results are from  $[0, low)$ .

#### Parameters

- **shape\_or\_dist** (*shape tuple or Distribution object*) –
- **low** (*int*) – Lowest (signed) integer to be drawn from the distribution (unless *high=None*, in which case this parameter is the *highest* such integer).
- **high** (*int, optional*) – if provided, one above the largest (signed) integer to be drawn from the distribution (see above for behavior if *high=None*).

---

<sup>1</sup> P. R. Peebles Jr., “Central Limit Theorem” in “Probability, Random Variables and Random Signal Principles”, 4th ed., 2001, pp. 51, 51, 125.

**Returns out** – DistArray of random integers from the appropriate distribution.

**Return type** DistArray of ints

**randn** (*shape\_or\_dist*)

Return samples from the “standard normal” distribution.

**Parameters shape\_or\_dist** (*shape tuple or Distribution object*) –

**Returns out** – A DistArray of floating-point samples from the standard normal distribution.

**Return type** *DistArray*

**seed** (*seed=None*)

Seed the random number generators on each engine.

**Parameters seed** (*None, int, or array of integers*) – Base random number seed to use on each engine. If *None*, then a non-deterministic seed is obtained from the operating system. Otherwise, the seed is used as passed, and the sequence of random numbers will be deterministic.

Each individual engine has its state adjusted so that it is different from each other engine. Thus, each engine will compute a different sequence of random numbers.

## localapi Package

### localapi Package

Modules dealing with the local index-space view of *DistArrays*.

In other words, the view from an engine.

### construct Module

`distarray.localapi.construct.init_base_comm(comm)`

Sanitize an MPI.comm instance or create one.

`distarray.localapi.construct.init_comm(base_comm, grid_shape)`

Create an MPI communicator with a cartesian topology.

### error Module

**exception** `distarray.localapi.error.IncompatibleArrayError`

Bases: `distarray.error.DistArrayError`

Exception class when arrays are incompatible.

**exception** `distarray.localapi.error.InvalidBaseCommError`

Bases: `distarray.error.DistArrayError`

Exception class when an object expected to be an MPI.Comm object is not one.

**exception** `distarray.localapi.error.InvalidDimensionError`

Bases: `distarray.error.DistArrayError`

Exception class when a specified dimension is invalid.

**exception** `distarray.localapi.error.NullCommError`

Bases: `distarray.error.DistArrayError`

Exception class when an MPI communicator is NULL.

## format Module

Define a simple format for saving LocalArrays to disk with full information about them. This format, `.dnpy`, draws heavily from the `.npy` format specification from NumPy and from the data structure defined in the Distributed Array Protocol.

**Version numbering** The version numbering of this format is independent of DistArray’s and the Distributed Array Protocol’s version numberings.

**Format Version 1.0** The first 6 bytes are a magic string: exactly `\x93DARRAY`.

The next 1 byte is an unsigned byte: the major version number of the file format, e.g. `\x01`.

The next 1 byte is an unsigned byte: the minor version number of the file format, e.g. `\x00`. Note: the version of the file format is not tied to the version of the DistArray package.

The next 2 bytes form a little-endian unsigned short int: the length of the header data `HEADER_LEN`.

The next `HEADER_LEN` bytes form the header data describing the distribution of this chunk of the LocalArray. It is an ASCII string which contains a Python literal expression of a dictionary. It is terminated by a newline (`\n`) and padded with spaces (`\x20`) to make the total length of `magic string + 4 + HEADER_LEN` be evenly divisible by 16 for alignment purposes.

The dictionary contains two keys, both described in the Distributed Array Protocol:

“**\_\_version\_\_**” [str] Version of the Distributed Array Protocol used in this header.

“**dim\_data**” [tuple of dict] One dictionary per array dimension; see the Distributed Array Protocol for the details of this data structure.

For repeatability and readability, the dictionary keys are sorted in alphabetic order. This is for convenience only. A writer SHOULD implement this if possible. A reader MUST NOT depend on this.

Following this header is the output of `numpy.save` for the underlying data buffer. This contains the full output of `save`, beginning with the magic number for `.npy` files, followed by the `.npy` header and array data.

The `.npy` format, including reasons for creating it and a comparison of alternatives, is described fully in the “`numpy-format`” NEP and in the module docstring for `numpy.lib.format`.

```
distarray.localapi.format.magic(major, minor, prefix=<MagicMock name='mock.asbytes()'
                                id='139799651562512'>)
```

Return the magic string for the given file format version.

### Parameters

- **major** (int in [0, 255]) –
- **minor** (int in [0, 255]) –

### Returns magic

### Return type str

**Raises** `ValueError` – if the version cannot be formatted.

```
distarray.localapi.format.read_array_header_1_0(fp)
```

Read an array header from a filelike object using the 1.0 file format version.

This will leave the file object located just after the header.

**Parameters** **fp** (filelike object) – A file object or something with a `.read()` method like a file.

### Returns

- **\_\_version\_\_** (*str*) – Version of the Distributed Array Protocol used.
- **dim\_data** (*tuple*) – A tuple containing a dictionary for each dimension of the underlying array, as described in the Distributed Array Protocol.

**Raises** `ValueError` – If the data is invalid.

`distarray.localapi.format.read_localarray(fp)`

Read a LocalArray from an .dnpy file.

**Parameters** **fp** (*file\_like object*) – If this is not a real file object, then this may take extra memory and time.

**Returns** **distbuffer** – The Distributed Array Protocol structure created from the data on disk.

**Return type** `dict`

**Raises** `ValueError` – If the data is invalid.

`distarray.localapi.format.read_magic(fp)`

Read the magic string to get the version of the file format.

**Parameters** **fp** (*filelike object*) –

**Returns**

- **major** (*int*)
- **minor** (*int*)

`distarray.localapi.format.write_localarray(fp, arr, version=(1, 0))`

Write a LocalArray to a .dnpy file, including a header.

The `__version__` and `dim_data` keys from the Distributed Array Protocol are written to a header, then `numpy.save` is used to write the value of the `buffer` key.

**Parameters**

- **fp** (*file\_like object*) – An open, writable file object, or similar object with a `.write()` method.
- **arr** (`LocalArray`) – The array to write to disk.
- **version** (*((int, int), optional)*) – The version number of the file format. Default: (1, 0)

**Raises**

- `ValueError` – If the array cannot be persisted.
- *Various other errors* – If the underlying numpy array contains Python objects as part of its dtype, the process of pickling them may raise various errors if the objects are not picklable.

## localarray Module

The *LocalArray* data structure.

*DistArray* objects are proxies for collections of *LocalArray* objects (that usually reside on engines).

**class** `distarray.localapi.localarray.GlobalIndex(distribution, ndarray)`

Bases: `object`

Object which provides access to global indexing on LocalArrays.

**checked\_getitem** (*global\_inds*)

**checked\_setitem** (*global\_inds, value*)

**get\_slice** (*global\_inds, new\_distribution*)

**class** `distarray.localapi.localarray.GlobalIterator` (*arr*)  
Bases: `distarray.externals.six.Iterator`

**class** `distarray.localapi.localarray.LocalArray` (*distribution, dtype=None, buf=None*)  
Bases: `object`

Distributed memory Python arrays.

**\_\_array\_wrap\_\_** (*obj, context=None*)  
Return a LocalArray based on *obj*.

This method constructs a new LocalArray object using the distribution from self and the buffer from *obj*.

This is used to construct return arrays for ufuncs.

**\_\_distarray\_\_** ()  
Returns the data structure required by the DAP.

DAP = Distributed Array Protocol

See the project's documentation for the Protocol's specification.

**\_\_getitem\_\_** (*index*)  
Get a local item.

**\_\_setitem\_\_** (*index, value*)  
Set a local item.

**asdist\_like** (*other*)  
Return a version of self that has shape, dist and grid\_shape like *other*.

**astype** (*newdtype*)  
Return a copy of this LocalArray with a new underlying dtype.

**cart\_coords**

**comm**

**comm\_rank**

**comm\_size**

**compatibility\_hash** ()

**coords\_from\_rank** (*rank*)

**copy** ()  
Return a copy of this LocalArray.

**dim\_data**

**dist**

**dtype**

**fill** (*scalar*)

**classmethod from\_distarray** (*comm, obj*)  
Make a LocalArray from Distributed Array Protocol data structure.

An object that supports the Distributed Array Protocol will have a `__distarray__` method that returns the data structure described here:

<https://github.com/enthought/distributed-array-protocol>



**Parameters** `obj` (an object with a `__distarray__` method or a dict) – If a dict, it must conform to the structure defined by the distributed array protocol.

**Returns** A LocalArray encapsulating the buffer of the original data. No copy is made.

**Return type** *LocalArray*

`global_from_local` (*local\_ind*)

`global_limits` (*dim*)

`global_shape`

`global_size`

`grid_shape`

`itemsize`

`local_data`

`local_from_global` (*global\_ind*)

`local_shape`

`local_size`

`local_view` (*dtype=None*)

`nbytes`

`ndarray`

`ndim`

`pack_index` (*inds*)

`rank_from_coords` (*coords*)

`sync` ()

`unpack_index` (*packed\_ind*)

`view` (*distribution, dtype*)

Return a new LocalArray whose underlying *ndarray* is a view on *self.ndarray*.

**class** `distarray.localapi.localarray.LocalArrayBinaryOperation` (*numpy\_ufunc*)

Bases: object

**class** `distarray.localapi.localarray.LocalArrayUnaryOperation` (*numpy\_ufunc*)

Bases: object

`distarray.localapi.localarray.arecompatible` (*a, b*)

Do these arrays have the same compatibility hash?

`distarray.localapi.localarray.compact_indices` (*dim\_data*)

Given a *dim\_data* structure, return a tuple of compact indices.

For every dimension in *dim\_data*, return a representation of the indices indicated by that *dim\_dict*; return a slice if possible, else, return the list of global indices.

**Parameters** `dim_data` (*tuple of dict*) – A dict for each dimension, with the data described here: <https://github.com/enthought/distributed-array-protocol> we use only the indexing related keys from this structure here.

**Returns** `index` – Efficient structure usable for indexing into a numpy-array-like data structure.

**Return type** tuple of slices and/or lists of int

```
distarray.localapi.localarray.empty(distribution, dtype=<type 'float'>)
```

Create an empty LocalArray.

```
distarray.localapi.localarray.empty_like(arr, dtype=None)
```

Create an empty LocalArray with a distribution like *arr*.

```
distarray.localapi.localarray.fromfunction(function, distribution, **kwargs)
```

```
distarray.localapi.localarray.fromndarray_like(ndarray, like_arr)
```

Create a new LocalArray like *like\_arr* with buffer set to *ndarray*.

```
distarray.localapi.localarray.get_printoptions()
```

```
distarray.localapi.localarray.load_dnpz(comm, file)
```

Load a LocalArray from a *.dnpz* file.

**Parameters** *file* (file-like object or str) – The file to read. It must support `seek()` and `read()` methods.

**Returns** *result* – A LocalArray encapsulating the data loaded.

**Return type** *LocalArray*

```
distarray.localapi.localarray.load_hdf5(comm, filename, dim_data, key='buffer')
```

Load a LocalArray from an *.hdf5* file.

**Parameters**

- **filename** (str) – The filename to read.
- **dim\_data** (tuple of dict) – A dict for each dimension, with the data described here: <https://github.com/enthought/distributed-array-protocol>, describing which portions of the HDF5 file to load into this LocalArray, and with what metadata.
- **comm** (MPI comm object) –
- **key** (str, optional) – The identifier for the group to load the LocalArray from (the default is 'buffer').

**Returns** *result* – A LocalArray encapsulating the data loaded.

**Return type** *LocalArray*

---

**Note:** For *dim\_data* dimension dictionaries containing unstructured ('u') distribution types, the indices selected by the 'indices' key must be in increasing order. This is a limitation of h5py / hdf5.

---

```
distarray.localapi.localarray.load_npy(comm, filename, dim_data)
```

Load a LocalArray from a *.npz* file.

**Parameters**

- **filename** (str) – The file to read.
- **dim\_data** (tuple of dict) – A dict for each dimension, with the data described here: <https://github.com/enthought/distributed-array-protocol>, describing which portions of the HDF5 file to load into this LocalArray, and with what metadata.
- **comm** (MPI comm object) –

**Returns** *result* – A LocalArray encapsulating the data loaded.

**Return type** *LocalArray*

```
distarray.localapi.localarray.local_reduction(out_comm, reducer, larr, ddpr, dtype, axes)
```

Entry point for reductions on local arrays.

**Parameters**

- **reducer** (*callable*) – Performs the core reduction operation.
- **out\_comm** (*MPI Comm instance.*) – The MPI communicator for the result of the reduction. Is equal to MPI.COMM\_NULL when this rank is not part of the output communicator.
- **larr** (*LocalArray*) – Input. Defined for all ranks.
- **ddpr** (*sequence of dim-data dictionaries.*) –
- **axes** (*Sequence of ints or None.*) –

**Returns** When out\_comm == MPI.COMM\_NULL, returns None. Otherwise, returns the LocalArray section of the reduction result.

**Return type** LocalArray or None

`distarray.localapi.localarray.max_reducer(reduce_comm, larr, out, axes, dtype)`  
Core reduction function for max.

`distarray.localapi.localarray.mean_reducer(reduce_comm, larr, out, axes, dtype)`  
Core reduction function for mean.

`distarray.localapi.localarray.min_reducer(reduce_comm, larr, out, axes, dtype)`  
Core reduction function for min.

`distarray.localapi.localarray.ndenumerate(arr)`

`distarray.localapi.localarray.ones(distribution, dtype=<type 'float'>)`  
Create a LocalArray filled with ones.

`distarray.localapi.localarray.save_dnpz(file, arr)`  
Save a LocalArray to a .dnpz file.

**Parameters**

- **file** (*file-like object or str*) – The file or filename to which the data is to be saved.
- **arr** (*LocalArray*) – Array to save to a file.

`distarray.localapi.localarray.save_hdf5(filename, arr, key='buffer', mode='a')`  
Save a LocalArray to a dataset in an .hdf5 file.

**Parameters**

- **filename** (*str*) – Name of file to write to.
- **arr** (*LocalArray*) – Array to save to a file.
- **key** (*str, optional*) – The identifier for the group to save the LocalArray to (the default is 'buffer').
- **mode** (*optional, {'w', 'w-', 'a'}, default 'a'*) –
  - 'w' Create file, truncate if exists
  - 'w-' Create file, fail if exists
  - 'a' Read/write if exists, create otherwise (default)

`distarray.localapi.localarray.set_printoptions(precision=None, threshold=None, edgeitems=None, linewidth=None, suppress=None)`

`distarray.localapi.localarray.std_reducer(reduce_comm, larr, out, axes, dtype)`  
Core reduction function for std.

`distarray.localapi.localarray.sum_reducer(reduce_comm, larr, out, axes, dtype)`

Core reduction function for sum.

`distarray.localapi.localarray.var_reducer(reduce_comm, larr, out, axes, dtype)`

Core reduction function for var.

`distarray.localapi.localarray.zeros(distribution, dtype=<type 'float'>)`

Create a LocalArray filled with zeros.

`distarray.localapi.localarray.zeros_like(arr, dtype=<type 'float'>)`

Create a LocalArray of zeros with a distribution like *arr*.

## maps Module

Classes to manage the distribution-specific aspects of a LocalArray.

The Distribution class is the main entry point and is meant to be used by LocalArrays to help translate between local and global index spaces. It manages *ndim* one-dimensional map objects.

The one-dimensional map classes BlockMap, CyclicMap, BlockCyclicMap, and UnstructuredMap all manage the mapping tasks for their particular dimension. All are subclasses of MapBase. The reason for the several subclasses is to allow more compact and efficient operations.

**class** `distarray.localapi.maps.BlockCyclicMap(global_size, grid_size, grid_rank, start, block_size)`

Bases: `distarray.localapi.maps.MapBase`

One-dimensional block cyclic map class.

**dim\_dict**

**dist** = 'c'

**global\_from\_local\_index**(lidx)

**global\_iter**

**global\_slice**

Return a slice representing the global index space of this dimension; only possible for `block_size == 1`.

**local\_from\_global\_index**(gidx)

**size**

**class** `distarray.localapi.maps.BlockMap(global_size, grid_size, grid_rank, start, stop)`

Bases: `distarray.localapi.maps.MapBase`

One-dimensional block map class.

**dim\_dict**

**dist** = 'b'

**global\_from\_local\_index**(lidx)

**global\_from\_local\_slice**(lidx)

**global\_iter**

**global\_slice**

Return a slice representing the global index space of this dimension.

**local\_from\_global\_index**(gidx)

**local\_from\_global\_slice**(gidx)

**size**

**class** `distarray.localapi.maps.CyclicMap(global_size, grid_size, grid_rank, start)`

Bases: `distarray.localapi.maps.MapBase`

One-dimensional cyclic map class.

**dim\_dict**

**dist** = 'c'

**global\_from\_local\_index**(*lidx*)

**global\_iter**

**global\_slice**

Return a slice representing the global index space of this dimension.

**local\_from\_global\_index**(*gidx*)

**size**

**class** `distarray.localapi.maps.Distribution(comm, dim_data)`

Bases: `object`

Multi-dimensional Map class.

Manages one or more one-dimensional map classes.

**cart\_coords**

**comm\_rank**

**comm\_size**

**coords\_from\_rank**(*rank*)

**dim\_data**

**dist**

**classmethod** **from\_shape**(*comm, shape, dist=None, grid\_shape=None*)

Create a Distribution from a *shape* and optional arguments.

**global\_from\_local**(*local\_ind*)

Given *local\_ind* indices, translate into global indices.

**global\_shape**

**global\_size**

**global\_slice**

Return a slice representing the global index space of this dimension.

**grid\_shape**

**local\_from\_global**(*global\_ind*)

Given *global\_ind* indices, translate into local indices.

**local\_shape**

**local\_size**

**ndim**

**rank\_from\_coords**(*coords*)

**class** `distarray.localapi.maps.MapBase`

Bases: `object`

Base class for all one dimensional Map classes.

**class** `distarray.localapi.maps.UnstructuredMap` (*global\_size, grid\_size, grid\_rank, indices*)

Bases: `distarray.localapi.maps.MapBase`

One-dimensional unstructured map class.

**dim\_dict**

**dist** = 'u'

**global\_from\_local\_index** (*lidx*)

**global\_iter**

**local\_from\_global\_index** (*gidx*)

**size**

`distarray.localapi.maps.map_from_dim_dict` (*dd*)

Factory function that returns a 1D map for a given dimension dictionary.

### **mpiutils Module**

Entry point for MPI.

`distarray.localapi.mpiutils.create_comm_of_size` (*size=4*)

Create a subcommunicator of COMM\_PRIVATE of given size.

`distarray.localapi.mpiutils.create_comm_with_list` (*nodes, base\_comm=None*)

Create a subcommunicator of base\_comm with a list of ranks.

If base\_comm is not specified, defaults to COMM\_PRIVATE.

`distarray.localapi.mpiutils.get_comm_private` ()

`distarray.localapi.mpiutils.mpi_type_for_ndarray` (*a*)

### **proxyize Module**

**class** `distarray.localapi.proxyize.Proxy` (*name, obj, module\_name*)

Bases: `object`

**cleanup** ()

**dereference** ()

Callable only on the engines.

**class** `distarray.localapi.proxyize.Proxyize`

Bases: `object`

**next\_name** ()

**set\_state** (*state*)

**str\_counter** ()

## random Module

Pseudo-random number generation routines for local arrays.

This module provides a number of routines for generating random numbers, from a variety of probability distributions.

`distarray.localapi.random.beta(a, b, distribution=None)`

Return an array with random numbers from the beta probability distribution.

### Parameters

- **a** (*float*) – Parameter that describes the beta probability distribution.
- **b** (*float*) – Parameter that describes the beta probability distribution.
- **distribution** (*The desired distribution of the array.*) – If None, then a normal NumPy array is returned. Otherwise, a LocalArray with this distribution is returned.

### Returns

**Return type** An array with random numbers.

`distarray.localapi.random.label_state(comm)`

Label/personalize the random generator state for the local rank.

This ensures that each separate engine, when using the same global seed, will generate a different sequence of pseudo-random numbers.

`distarray.localapi.random.normal(loc=0.0, scale=1.0, distribution=None)`

Return an array with random numbers from a normal (Gaussian) probability distribution.

### Parameters

- **loc** (*float*) – The mean (or center) of the probability distribution.
- **scale** (*float*) – The standard deviation (or width) of the probability distribution.
- **distribution** (*The desired distribution of the array.*) – If None, then a normal NumPy array is returned. Otherwise, a LocalArray with this distribution is returned.

### Returns

**Return type** An array with random numbers.

`distarray.localapi.random.rand(distribution=None)`

Return an array with random numbers distributed over the interval [0, 1).

**Parameters** **distribution** (*The desired distribution of the array.*) – If None, then a normal NumPy array is returned. Otherwise, a LocalArray with this distribution is returned.

### Returns

**Return type** An array with random numbers.

`distarray.localapi.random.randint(low, high=None, distribution=None)`

Return random integers from low (inclusive) to high (exclusive).

Return random integers from the “discrete uniform” distribution in the “half-open” interval [low, high). If high is None (the default), then results are from [0, low).

### Parameters

- **low** (*int*) – Lowest (signed) integer to be drawn from the distribution (unless high=None, in which case this parameter is the highest such integer).

- **high** (*int, optional*) – If provided, one above the largest (signed) integer to be drawn from the distribution (see above for behavior if high=None).
- **distribution** (*The desired distribution of the array.*) – If None, then a normal NumPy array is returned. Otherwise, a LocalArray with this distribution is returned.

#### Returns

**Return type** An array with random numbers.

`distarray.localapi.random.randn (distribution=None)`

Return a sample (or samples) from the “standard normal” distribution.

**Parameters** **distribution** (*The desired distribution of the array.*) – If None, then a normal NumPy array is returned. Otherwise, a LocalArray with this distribution is returned.

#### Returns

**Return type** An array with random numbers.

## plotting Package

### plotting Package

Plotting functions for distarrays.

### plotting Module

Plotting functions for distarrays.

`distarray.plotting.plotting.cmap_discretize (cmap, N)`

Create a discrete colormap from the continuous colormap cmap.

#### Parameters

- **cmap** (*colormap instance, or string*) – The continuous colormap, as object or name, to make discrete. For example, `matplotlib.cm.jet`, or ‘jet’.
- **N** (*int*) – The number of discrete colors desired.

**Returns** The desired discrete colormap.

**Return type** colormap

#### Example

```
>>> x = resize(arange(100), (5,100))
>>> djet = cmap_discretize(cm.jet, 5)
>>> pyplot.imshow(x, cmap=djet)
```

`distarray.plotting.plotting.create_discrete_colormaps (num_values)`

Create colormap objects for a discrete colormap.

**Parameters** **num\_values** (*The number of distinct colors to use.*) –

**Returns** **cmap, norm, text\_colors** – The matplotlib colormap, norm, and recommended text colors. `text_colors` is an array of length `num_values`, with each entry being a nice color for text drawn on top of the colormap selection.



**Return type** tuple

`distarray.plotting.plotting.get_ranks(arr)`

Given a distarray `arr`, return a distarray with the same shape, but with the elements equal to the rank of the process the element is on.

`distarray.plotting.plotting.plot_array_distribution(darray, process_coords, title=None, xlabel=None, ylabel=None, yflip=False, cell_label=True, legend=False, global_plot_filename=None, local_plot_filename=None, *args, **kwargs)`

Plot a distarray's memory layout. It can be 1D or 2D. Elements are colored according to the process they are on.

**Parameters**

- **darray** (`DistArray`) – The distributed array to plot.
- **process\_coords** (*List of tuples.*) – The process grid coordinates.
- **title** (*string*) – Text label for the plot title, or `None`.
- **xlabel** (*string*) – Text label for the x-axis, or `None`.
- **ylabel** (*string*) – Text label for the y-axis, or `None`.
- **yflip** (*bool*) – If `True`, then the y-axis increases downwards, to match the layout when printing the array itself.
- **cell\_label** (*bool*) – If `True`, then each cell in the plot is labeled with the array value. This can look cluttered for large arrays.
- **legend** (*bool*) – If `True`, then a colorbar legend is drawn to label the colors.
- **global\_plot\_filename** (*string*) – Output filename for the global array plot image.
- **local\_plot\_filename** (*string*) – Output filename for the local array plot image.

**Returns** The process assignment array, as a `DistArray`.

**Return type** out

`distarray.plotting.plotting.plot_local_array_subfigure(subfig, local_array, process_coord, colormap_objects, *args, **kwargs)`

Plot a single `local_array` into a matplotlib subfigure.

`distarray.plotting.plotting.plot_local_arrays(darray, process_coords, colormap_objects, filename)`

Plot the local arrays as a multi-figure matplotlib plot.

## 5.2 Building HDF5 and h5py for DistArray

These are notes from trying to build HDF5 1.8.12 and h5py 2.2.1 against mpi4py 1.3 and openmpi-1.6.5 on OS X 10.8.5.

### 5.2.1 HDF5

Download the HDF5 source (1.8.12) and configure it with parallel support. From the source directory:

```
$ CFLAGS=-O0 CC=/Users/robertgrant/localroot/bin/mpicc ./configure --enable-shared --enable-parallel
```

The `CFLAGS` setting is to get around a known problem with the tests on OS X 10.8 ([http://www.hdfgroup.org/HDF5/release/known\\_problems/](http://www.hdfgroup.org/HDF5/release/known_problems/)).

Build it:

```
$ make
```

Test it:

```
$ make check
```

This produced some errors related to `ph5diff`, which the website claims are “not valid errors”, so I ignored them (<http://www.hdfgroup.org/HDF5/faq/parallel.html#ph5difftest>).

Install HDF5:

```
$ make install
```

### 5.2.2 h5py

Build `h5py` against this version of HDF5. Without setting `HDF5_DIR`, on my system the build found Canopy’s serial version of HDF5. In the `h5py` source directory:

```
$ HDF5_DIR=/Users/robertgrant/localroot/ CC=mpicc python setup.py build --mpi
```

This gives me an error about “MPI Message” addressed here:

```
https://github.com/h5py/h5py/issues/401
```

After patching `api_compat.h` as suggested, it builds. One could also use the master version of `h5py` from GitHub instead of the latest release.

Run the tests:

```
$ python setup.py test
```

and install `h5py`:

```
$ python setup.py install
```

You should now be able to run the example code listed here:

```
http://docs.h5py.org/en/latest/mpi.html#using-parallel-hdf5-from-h5py
```

## 5.3 Notes on building environment-modules

`environment-modules` is a tool, written with Tcl, that makes it convenient to switch environment settings. It is not required to use `DistArray`, but we find it useful in development. It is a difficult name to google. I had to build it from source, and made some notes of my steps, which will hopefully be helpful for others that build this.

There seems to be some version available to `apt-get` for Debian. But I read suggestions not to mix Debian and Ubuntu packages, and as I have Ubuntu, I did not try and configure my `apt-get` to look at the Debian packages. So I installed from source, with notes as follows.

These specific notes are from an installation from source for Linux Mint (Ubuntu), done by Mark Kness. These actions were based on the `INSTALL` document in the modules source and the Geoghegan link.

```
$ sudo apt-get install tcl tcl8.4-dev
```

This seemed to run ok.

```
$ tar xvvf modules-3.2.10.tar.gz
```

I had already downloaded this. Double v means extra verbose.

```
$ cd modules-3.2.10
$ gedit README
$ gedit INSTALL
$ gedit INSTALL.RH7x
```

Read the installation notes!

```
$ ./configure
```

First step is to run this and see how far it gets. Tcl is the likely problem here.

I got the following messages from ./configure...:

```
checking for Tcl configuration (tclConfig.sh)... found /usr/lib/tcl8.4/tclConfig.sh
checking for existence of tclConfig.sh... loading
checking for Tcl version... 8.5
checking TCL_VERSION... 8.5
checking TCL_LIB_SPEC... -L/usr/lib -ltcl8.4
checking TCL_INCLUDE_SPEC... -I/usr/include/tcl8.4
checking for TclX configuration (tclxConfig.sh)... not found
checking for TclX version... using 8.5
checking TCLX_VERSION... 8.5
checking TCLX_LIB_SPEC... TCLX_LIB_SPEC not found, need to use --with-tclx-lib
checking TCLX_INCLUDE_SPEC... TCLX_INCLUDE_SPEC not found, need to use --with-tclx-inc
configure: WARNING: will use MODULEPATH=/usr/local/Modules/modulefiles : rerun configure using --with-
configure: WARNING: will use VERSIONPATH=/usr/local/Modules/versions : rerun configure using --with-v
```

It seems that TCL\_VERSION, TCL\_LIB\_SPEC, and TCL\_INCLUDE\_SPEC were all found ok. (The TCLX variants are not found but that is different and not a problem.) Generally it seems like Tcl is ok, except perhaps for some 8.4 vs 8.5 version inconsistency. A non-default path for the module files themselves seems recommended, so...

```
$ cd ~
$ mkdir modules
```

This created /home/mkness/modules on my machine. The install notes suggest that one make a non-default location for these. This directory name was an arbitrary choice.

```
$ cd modules-3.2.10
$ ./configure --with-module-path=~/modules
```

Seemed ok. I ignored the version and prefix path options.

```
$ make
```

Seemed basically ok, a few warnings.

```
$ ./modulecmd sh
```

I got the usage instructions, and NOT any Tcl messages. Ok!

```
$ sudo make install
```

Seemed to run ok. Got permission errors without sudo.

```
$ cd /usr/local/Modules
```

```
$ sudo ln -s 3.2.10 default
```

Setup symbolic link named ‘default’ to point to the installed version.

```
$ cd ~
```

```
$ /usr/local/Modules/default/bin/add.modules
```

This script is supposed to update my local `.bashrc` and similar files to have access to the Modules stuff. For me, it modified `.bashrc` and `.profile`. But if I say ‘module’, I get an error about an invalid path. It seems that `MODULE_VERSION` is not defined, so I added `export MODULE_VERSION=default` to the top of my `.bashrc`.

At this point I can say ‘module’ at the command line and I get the usage instructions. But ‘module avail’ dislikes the lack of an environment variable `MODULEPATH`. So I also add `export MODULEPATH=~/.modules` to my `.bashrc`. This path matches the `-with-module-path` argument to `./configure`.

Now it works!

### 5.3.1 References

<http://modules.sourceforge.net/> The main page for the modules package. It provides a source download: `modules-3.2.10.tar.gz`

<http://sourceforge.net/p/modules/wiki/FAQ/> FAQ for the modules package.

<http://nickgeoghegan.net/linux/installing-environment-modules> Build instructions for environment-modules. I partially followed these but with several changes.

<http://packages.debian.org/wheezy/environment-modules> <http://packages.debian.org/wheezy/amd64/environment-modules/download> <http://packages.debian.org/unstable/main/environment-modules> Debian package for environment-modules. Note that this is two different places.

<http://packages.debian.org/search?keywords=tcl&searchon=names&suite=stable&section=all> Debian package for Tcl.

## 5.4 Licence for *six.py* version 1.5.2

Copyright (c) 2010-2014 Benjamin Peterson

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



---

## Release Notes

---

### 6.1 DistArray 0.2: development release

Documentation: <http://distarray.readthedocs.org> License: Three-clause BSD Python versions: 2.7 and 3.3 OS support: \*nix and Mac OS X

DistArray aims to bring the strengths of NumPy to data-parallel high-performance computing. It provides distributed multi-dimensional NumPy-like arrays and distributed ufuncs, distributed IO capabilities, and can integrate with external distributed libraries, like Trilinos. DistArray works with NumPy and builds on top of it in a flexible and natural way.

Brian Granger started DistArray as a NASA-funded SBIR project in 2008. Enthought picked it up as part of a DOE Phase II SBIR [0] to provide a generally useful distributed array package. It builds on IPython, IPython.parallel, NumPy, MPI, and interfaces with the Trilinos suite of distributed HPC solvers (via PyTrilinos) [1].

Distarray:

- has a client-engine (or master-worker) process design – data resides on the worker processes, commands are initiated from master;
- allows full control over what is executed on the worker processes and integrates transparently with the master process;
- allows direct communication between workers bypassing the master process for scalability;
- integrates with IPython.parallel for interactive creation and exploration of distributed data;
- supports distributed ufuncs (currently without broadcasting);
- builds on and leverages MPI via MPI4Py in a transparent and user-friendly way;
- supports NumPy-like structured multidimensional arrays;
- has basic support for unstructured arrays;
- supports user-controllable array distributions across workers (block, cyclic, block-cyclic, and unstructured) on a per-axis basis;
- has a straightforward API to control how an array is distributed;
- has basic plotting support for visualization of array distributions;
- separates the array's distribution from the array's data – useful for slicing, reductions, redistribution, broadcasting, all of which will be implemented in coming releases;
- implements distributed random arrays;

- supports .npy-like flat-file IO and hdf5 parallel IO (via h5py); leverages MPI-based IO parallelism in an easy-to-use and transparent way; and
- supports the distributed array protocol [2], which allows independently developed parallel libraries to share distributed arrays without copying, analogous to the PEP-3118 new buffer protocol.
- This is the first public development release. DistArray is not ready for real-world use, but we want to get input from the larger scientific-Python community to help drive its development. The API is changing rapidly and we are adding many new features on a fast timescale. For that reason, DistArray is currently implemented in pure Python for maximal flexibility. Performance improvements are coming.

The 0.2 release's goals are to provide the components necessary to support upcoming features that are non-trivial to implement in a distributed environment.

Planned features for upcoming releases:

- Distributed reductions
- Distributed slicing
- Distributed broadcasting
- Distributed fancy indexing
- Re-distribution methods
- Integration with Trilinos [1] and other packages [3] that subscribe to the distributed array protocol [2]
- Lazy evaluation and deferred computation for latency hiding
- Out-of-core computations
- Extensive examples, tutorials, documentation
- Support for distributed sorting and other non-trivial distributed algorithms
- MPI-only communication for non-interactive deployment on clusters and supercomputers
- End-user control over communication and temporary array creation, and other performance aspects of distributed computations

[0] <http://www.sbir.gov/sbirsearch/detail/410257> [1] <http://trilinos.org/> [2] <http://distributed-array-protocol.readthedocs.org/en/rel-0.10.0/> [3] <http://www.mcs.anl.gov/petsc/>

## 6.2 DistArray 0.3: development release

**Documentation:** <http://distarray.readthedocs.org>

**License:** Three-clause BSD

**Python versions:** 2.7 and 3.3

**OS support:** \*nix and Mac OS X

### 6.2.1 What is DistArray?

DistArray aims to bring the strengths of NumPy to data-parallel high-performance computing. It provides distributed multi-dimensional NumPy-like arrays and distributed ufuncs, distributed IO capabilities, and can integrate with external distributed libraries, like Trilinos. DistArray works with NumPy and builds on top of it in a flexible and natural way.



### 6.2.2 0.3 Release

This is the second development release.

Noteworthy improvements in 0.3 include:

- support for distributions over a subset of processes;
- distributed reductions with a simple NumPy-like API: `da.sum(axis=3)` ;
- an `apply()` function for easier computation with process-local data;
- performance improvements and reduced communication overhead;
- cleanup, renamings, and refactorings;
- test suite improvements for parallel testing; and
- start of a more frequent release schedule.

DistArray is not ready for real-world use. We want to get input from the larger scientific-Python community to help drive its development. The API is changing rapidly and we are adding many new features on a fast timescale. DistArray is currently implemented in pure Python for maximal flexibility. Performance improvements are ongoing.

### 6.2.3 Existing features

Distarray:

- has a client-engine (or master-worker) process design – data resides on the worker processes, commands are initiated from master;
- allows full control over what is executed on the worker processes and integrates transparently with the master process;
- allows direct communication between workers bypassing the master process for scalability;
- integrates with IPython.parallel for interactive creation and exploration of distributed data;
- supports distributed ufuncs (currently without broadcasting);
- builds on and leverages MPI via MPI4Py in a transparent and user-friendly way;
- supports NumPy-like structured multidimensional arrays;
- has basic support for unstructured arrays;
- supports user-controllable array distributions across workers (block, cyclic, block-cyclic, and unstructured) on a per-axis basis;
- has a straightforward API to control how an array is distributed;
- has basic plotting support for visualization of array distributions;
- separates the array's distribution from the array's data – useful for slicing, reductions, redistribution, broadcasting, and other operations;
- implements distributed random arrays;
- supports `.npy`-like flat-file IO and hdf5 parallel IO (via `h5py`); leverages MPI-based IO parallelism in an easy-to-use and transparent way; and
- supports the distributed array protocol [[protocol](#)], which allows independently developed parallel libraries to share distributed arrays without copying, analogous to the PEP-3118 new buffer protocol.

## 6.2.4 Planned features and roadmap

- Distributed slicing
- Re-distribution methods
- Integration with Trilinos [*Trilinos*] and other packages [*petsc*] that subscribe to the distributed array protocol [*protocol*]
- Distributed broadcasting
- Distributed fancy indexing
- MPI-only communication for non-interactive deployment on clusters and supercomputers
- Lazy evaluation and deferred computation for latency hiding
- Out-of-core computations
- Extensive examples, tutorials, documentation
- Support for distributed sorting and other non-trivial distributed algorithms
- End-user control over communication and temporary array creation, and other performance aspects of distributed computations

## 6.2.5 History

Brian Granger started DistArray as a NASA-funded SBIR project in 2008. Enthought picked it up as part of a DOE Phase II SBIR [*SBIR*] to provide a generally useful distributed array package. It builds on IPython, IPython.parallel, NumPy, MPI, and interfaces with the Trilinos suite of distributed HPC solvers (via PyTrilinos [*Trilinos*]).

## 6.3 DistArray 0.4 development release

**Documentation:** <http://distarray.readthedocs.org>

**License:** Three-clause BSD

**Python versions:** 2.7 and 3.3

**OS support:** \*nix and Mac OS X

### 6.3.1 What is DistArray?

DistArray aims to bring the strengths of NumPy to data-parallel high-performance computing. It provides distributed multi-dimensional NumPy-like arrays and distributed ufuncs, distributed IO capabilities, and can integrate with external distributed libraries like Trilinos. DistArray works with NumPy and builds on top of it in a flexible and natural way.

### 6.3.2 0.4 Release

This is the third development release.

Noteworthy improvements in 0.4 include:

- basic slicing support;
- significant performance enhancements;

- reduction methods now support boolean arrays;
- an IPython notebook that demos basic functionality; and
- many bug fixes, API improvements, and refactorings.

DistArray is nearly ready for real-world use. The project is evolving rapidly and input from the larger scientific-Python community is very valuable and helps drive development.

### 6.3.3 Existing features

DistArray:

- has a client-engine (or master-worker) process design – data resides on the worker processes, and commands are initiated from master;
- allows full control over what is executed on the worker processes and integrates transparently with the master process;
- allows direct communication between workers, bypassing the master process for scalability;
- integrates with IPython.parallel for interactive creation and exploration of distributed data;
- supports distributed ufuncs (currently without broadcasting);
- builds on and leverages MPI via MPI4Py in a transparent and user-friendly way;
- supports NumPy-like multidimensional arrays;
- has basic support for unstructured arrays;
- supports user-controllable array distributions across workers (block, cyclic, block-cyclic, and unstructured) on a per-axis basis;
- has a straightforward API to control how an array is distributed;
- has basic plotting support for visualization of array distributions;
- separates the array’s distribution from the array’s data – useful for slicing, reductions, redistribution, broadcasting, and other operations;
- implements distributed random arrays;
- supports `.npy`-like flat-file IO and hdf5 parallel IO (via `h5py`); leverages MPI-based IO parallelism in an easy-to-use and transparent way; and
- supports the distributed array protocol [[protocol](#)], which allows independently developed parallel libraries to share distributed arrays without copying, analogous to the PEP-3118 new buffer protocol.

### 6.3.4 Planned features and roadmap

Near-term features and improvements include:

- MPI-only communication for performance and deployment on clusters and supercomputers;
- array re-distribution capabilities;
- interoperation with Trilinos [[Trilinos](#)];
- expanded tutorials, examples, and other introductory material; and
- distributed broadcasting support.

The longer-term roadmap includes:

- Lazy evaluation and deferred computation for latency hiding;
- Integration with other packages *[petsc]* that subscribe to the distributed array protocol *[protocol]*;
- Distributed fancy indexing;
- Out-of-core computations;
- Support for distributed sorting and other non-trivial distributed algorithms; and
- End-user control over communication and temporary array creation, and other performance aspects of distributed computations.

### 6.3.5 History and funding

Brian Granger started DistArray as a NASA-funded SBIR project in 2008. Enthought picked it up as part of a DOE Phase II SBIR *[SBIR]* to provide a generally useful distributed array package. It builds on NumPy, MPI, MPI4Py, IPython, IPython.parallel, and interfaces with the Trilinos suite of distributed HPC solvers (via PyTrilinos *[Trilinos]*).

This material is based upon work supported by the Department of Energy under Award Number DE-SC0007699.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

## 6.4 DistArray 0.5 release

**Documentation:** <http://distarray.readthedocs.org>

**License:** Three-clause BSD

**Python versions:** 2.7, 3.3, and 3.4

**OS support:** \*nix and Mac OS X

### 6.4.1 What is DistArray?

DistArray aims to bring the ease-of-use of NumPy to data-parallel high-performance computing. It provides distributed multi-dimensional NumPy arrays, distributed ufuncs, and distributed IO capabilities. It can efficiently interoperate with external distributed libraries like Trilinos. DistArray works with NumPy and builds on top of it in a flexible and natural way.

### 6.4.2 0.5 Release

Noteworthy improvements in this release include:

- closer alignment with NumPy's API,
- support for Python 3.4 (existing support for Python 2.7 and 3.3),
- a performance-oriented MPI-only mode for deployment on clusters and supercomputers,

- a way to register user-defined functions to be callable locally on worker processes,
- more consistent naming of sub-packages,
- testing with MPICH2 (already tested against OpenMPI),
- improved and expanded examples,
- installed version testable via `distarray.test()`, and
- performance and scaling improvements.

With this release, DistArray ready for real-world testing and deployment. The project is still evolving rapidly and we appreciate the continued input from the larger scientific-Python community.

### 6.4.3 Existing features

DistArray:

- supports NumPy-like slicing, reductions, and ufuncs on distributed multidimensional arrays;
- has a client-engine process design – data resides on the worker processes, commands are initiated from master;
- allows full control over what is executed on the worker processes and integrates transparently with the master process;
- allows direct communication between workers, bypassing the master process for scalability;
- integrates with IPython.parallel for interactive creation and exploration of distributed data;
- supports distributed ufuncs (currently without broadcasting);
- builds on and leverages MPI via MPI4Py in a transparent and user-friendly way;
- has basic support for unstructured arrays;
- supports user-controllable array distributions across workers (block, cyclic, block-cyclic, and unstructured) on a per-axis basis;
- has a straightforward API to control how an array is distributed;
- has basic plotting support for visualization of array distributions;
- separates the array's distribution from the array's data – useful for slicing, reductions, redistribution, broadcasting, and other operations;
- implements distributed random arrays;
- supports .numpy-like flat-file IO and hdf5 parallel IO (via `h5py`); leverages MPI-based IO parallelism in an easy-to-use and transparent way; and
- supports the distributed array protocol [\[protocol\]](#), which allows independently developed parallel libraries to share distributed arrays without copying, analogous to the PEP-3118 new buffer protocol.

### 6.4.4 Planned features and roadmap

Near-term features and improvements include:

- array re-distribution capabilities;
- lazy evaluation and deferred computation for latency hiding;
- interoperation with Trilinos [\[Trilinos\]](#); and
- distributed broadcasting support.

The longer-term roadmap includes:

- Integration with other packages [\[petsc\]](#) that subscribe to the distributed array protocol [\[protocol\]](#);
- Distributed fancy indexing;
- Out-of-core computations;
- Support for distributed sorting and other non-trivial distributed algorithms; and
- End-user control over communication and temporary array creation, and other performance aspects of distributed computations.

### 6.4.5 History and funding

Brian Granger started DistArray as a NASA-funded SBIR project in 2008. Enthought picked it up as part of a DOE Phase II SBIR [\[SBIR\]](#) to provide a generally useful distributed array package. It builds on NumPy, MPI, MPI4Py, IPython, IPython.parallel, and interfaces with the Trilinos suite of distributed HPC solvers (via PyTrilinos [\[Trilinos\]](#)).

This material is based upon work supported by the Department of Energy under Award Number DE-SC0007699.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`





[protocol] <http://distributed-array-protocol.readthedocs.org/en/rel-0.10.0/>  
[Trilinos] <http://trilinos.org/>  
[petsc] <http://www.mcs.anl.gov/petsc/>  
[SBIR] <http://www.sbir.gov/sbirsearch/detail/410257>  
[protocol] <http://distributed-array-protocol.readthedocs.org/en/rel-0.10.0/>  
[Trilinos] <http://trilinos.org/>  
[petsc] <http://www.mcs.anl.gov/petsc/>  
[SBIR] <http://www.sbir.gov/sbirsearch/detail/410257>  
[protocol] <http://distributed-array-protocol.readthedocs.org/en/rel-0.10.0/>  
[Trilinos] <http://trilinos.org/>  
[petsc] <http://www.mcs.anl.gov/petsc/>  
[SBIR] <http://www.sbir.gov/sbirsearch/detail/410257>



## d

- `distarray.__init__`, 11
- `distarray.__version__`, 11
- `distarray.apps.__init__`, 18
- `distarray.apps.dacluster`, 19
- `distarray.apps.engine`, 19
- `distarray.error`, 11
- `distarray.globalapi.__init__`, 19
- `distarray.globalapi.context`, 19
- `distarray.globalapi.distarray`, 24
- `distarray.globalapi.functions`, 25
- `distarray.globalapi.ipython_cleanup`, 27
- `distarray.globalapi.ipython_utils`, 27
- `distarray.globalapi.maps`, 27
- `distarray.globalapi.random`, 31
- `distarray.localapi.__init__`, 33
- `distarray.localapi.construct`, 33
- `distarray.localapi.error`, 33
- `distarray.localapi.format`, 34
- `distarray.localapi.localarray`, 35
- `distarray.localapi.maps`, 40
- `distarray.localapi.mpiutils`, 42
- `distarray.localapi.proxyize`, 42
- `distarray.localapi.random`, 43
- `distarray.metadata_utils`, 12
- `distarray.mpi_engine`, 14
- `distarray.mpionly_utils`, 15
- `distarray.plotting.__init__`, 44
- `distarray.plotting.plotting`, 44
- `distarray.run_tests`, 15
- `distarray.testing`, 15
- `distarray.utils`, 17



## Symbols

`__array_wrap__()` (distarray.localapi.localarray.LocalArray method), 36

`__distarray__()` (distarray.localapi.localarray.LocalArray method), 36

`__getitem__()` (distarray.localapi.localarray.LocalArray method), 36

`__setitem__()` (distarray.localapi.localarray.LocalArray method), 36

## A

`absolute()` (in module distarray.globalapi.functions), 25

`add()` (in module distarray.globalapi.functions), 26

`all_equal()` (in module distarray.utils), 17

`allclose()` (distarray.globalapi.context.BaseContext method), 19

`apply()` (distarray.globalapi.context.BaseContext method), 19

`apply()` (distarray.globalapi.context.IPythonContext method), 23

`apply()` (distarray.globalapi.context.MPIContext method), 23

`arccos()` (in module distarray.globalapi.functions), 25

`arccosh()` (in module distarray.globalapi.functions), 25

`arcsin()` (in module distarray.globalapi.functions), 25

`arcsinh()` (in module distarray.globalapi.functions), 25

`arctan()` (in module distarray.globalapi.functions), 25

`arctan2()` (in module distarray.globalapi.functions), 26

`arctanh()` (in module distarray.globalapi.functions), 25

`arecompatible()` (in module distarray.localapi.localarray), 37

`arg_kwarg_proxy_converter()` (distarray.mpi\_engine.Engine method), 14

`asdist_like()` (distarray.localapi.localarray.LocalArray method), 36

`asdistribution()` (in module distarray.globalapi.maps), 31

`assert_localarrays_allclose()` (in module distarray.testing), 16

`assert_localarrays_equal()` (in module distarray.testing),

16

`astype()` (distarray.localapi.localarray.LocalArray method), 36

## B

`BaseContext` (class in distarray.globalapi.context), 19

`BaseContextTestCase` (class in distarray.testing), 15

`beta()` (in module distarray.localapi.random), 43

`bitwise_and()` (in module distarray.globalapi.functions), 26

`bitwise_or()` (in module distarray.globalapi.functions), 26

`bitwise_xor()` (in module distarray.globalapi.functions), 26

`block_cyclic_size()` (in module distarray.metadata\_utils), 12

`block_size()` (in module distarray.metadata\_utils), 12

`BlockCyclicMap` (class in distarray.globalapi.maps), 27

`BlockCyclicMap` (class in distarray.localapi.maps), 40

`BlockMap` (class in distarray.globalapi.maps), 27

`BlockMap` (class in distarray.localapi.maps), 40

`builtin_call()` (distarray.mpi\_engine.Engine method), 14

## C

`c_or_bc_chooser()` (in module distarray.metadata\_utils), 12

`cart_coords` (distarray.localapi.localarray.LocalArray attribute), 36

`cart_coords` (distarray.localapi.maps.Distribution attribute), 41

`check_grid_shape_postconditions()` (in module distarray.metadata\_utils), 12

`check_grid_shape_preconditions()` (in module distarray.metadata\_utils), 12

`check_targets()` (in module distarray.testing), 16

`checked_getitem()` (distarray.localapi.localarray.GlobalIndex method), 35

`checked_setitem()` (distarray.localapi.localarray.GlobalIndex method), 35

[choose\\_map\(\)](#) (in module `distarray.globalapi.maps`), 31  
[cleanup\(\)](#) (`distarray.globalapi.context.BaseContext` method), 20  
[cleanup\(\)](#) (`distarray.globalapi.context.IPythonContext` method), 23  
[cleanup\(\)](#) (`distarray.globalapi.context.MPIContext` method), 23  
[cleanup\(\)](#) (`distarray.localapi.proxyize.Proxy` method), 42  
[cleanup\(\)](#) (in module `distarray.globalapi.ipython_cleanup`), 27  
[cleanup\\_all\(\)](#) (in module `distarray.globalapi.ipython_cleanup`), 27  
[clear\(\)](#) (in module `distarray.apps.dacluster`), 19  
[clear\(\)](#) (in module `distarray.globalapi.ipython_cleanup`), 27  
[clear\\_all\(\)](#) (in module `distarray.globalapi.ipython_cleanup`), 27  
[close\(\)](#) (`distarray.globalapi.context.BaseContext` method), 20  
[close\(\)](#) (`distarray.globalapi.context.IPythonContext` method), 23  
[close\(\)](#) (`distarray.globalapi.context.MPIContext` method), 23  
[cmap\\_discretize\(\)](#) (in module `distarray.plotting.plotting`), 44  
[comm](#) (`distarray.localapi.localarray.LocalArray` attribute), 36  
[comm\\_null\\_passes\(\)](#) (in module `distarray.testing`), 16  
[comm\\_rank](#) (`distarray.localapi.localarray.LocalArray` attribute), 36  
[comm\\_rank](#) (`distarray.localapi.maps.Distribution` attribute), 41  
[comm\\_size](#) (`distarray.localapi.localarray.LocalArray` attribute), 36  
[comm\\_size](#) (`distarray.localapi.maps.Distribution` attribute), 41  
[comm\\_size](#) (`distarray.testing.ParallelTestCase` attribute), 16  
[CommNullPasser](#) (class in `distarray.testing`), 15  
[compact\\_indices\(\)](#) (in module `distarray.localapi.localarray`), 37  
[compatibility\\_hash\(\)](#) (`distarray.localapi.localarray.LocalArray` method), 36  
[conjugate\(\)](#) (in module `distarray.globalapi.functions`), 25  
[context](#) (`distarray.globalapi.distarray.DistArray` attribute), 24  
[Context\(\)](#) (in module `distarray.globalapi.context`), 22  
[ContextCreationError](#), 22  
[ContextError](#), 11  
[coords\\_from\\_rank\(\)](#) (`distarray.localapi.localarray.LocalArray` method), 36  
[coords\\_from\\_rank\(\)](#) (`distarray.localapi.maps.Distribution` method), 41  
[copy\(\)](#) (`distarray.localapi.localarray.LocalArray` method), 36  
[cos\(\)](#) (in module `distarray.globalapi.functions`), 25  
[cosh\(\)](#) (in module `distarray.globalapi.functions`), 25  
[count\\_round\\_trips](#) (class in `distarray.utils`), 17  
[create\\_comm\\_of\\_size\(\)](#) (in module `distarray.localapi.mpiutils`), 42  
[create\\_comm\\_with\\_list\(\)](#) (in module `distarray.localapi.mpiutils`), 42  
[create\\_discrete\\_colormaps\(\)](#) (in module `distarray.plotting.plotting`), 44  
[cyclic\\_size\(\)](#) (in module `distarray.metadata_utils`), 12  
[CyclicMap](#) (class in `distarray.localapi.maps`), 41

## D

[DefaultContextTestCase](#) (class in `distarray.testing`), 16  
[delete\(\)](#) (`distarray.mpi_engine.Engine` method), 14  
[delete\\_key\(\)](#) (`distarray.globalapi.context.BaseContext` method), 20  
[delete\\_key\(\)](#) (`distarray.globalapi.context.MPIContext` method), 24  
[dereference\(\)](#) (`distarray.localapi.proxyize.Proxy` method), 42  
[dim\\_data](#) (`distarray.localapi.localarray.LocalArray` attribute), 36  
[dim\\_data](#) (`distarray.localapi.maps.Distribution` attribute), 41  
[dim\\_dict](#) (`distarray.localapi.maps.BlockCyclicMap` attribute), 40  
[dim\\_dict](#) (`distarray.localapi.maps.BlockMap` attribute), 40  
[dim\\_dict](#) (`distarray.localapi.maps.CyclicMap` attribute), 41  
[dim\\_dict](#) (`distarray.localapi.maps.UnstructuredMap` attribute), 42  
[dist](#) (`distarray.globalapi.distarray.DistArray` attribute), 24  
[dist](#) (`distarray.globalapi.maps.BlockCyclicMap` attribute), 27  
[dist](#) (`distarray.globalapi.maps.BlockMap` attribute), 28  
[dist](#) (`distarray.globalapi.maps.NoDistMap` attribute), 31  
[dist](#) (`distarray.globalapi.maps.UnstructuredMap` attribute), 31  
[dist](#) (`distarray.localapi.localarray.LocalArray` attribute), 36  
[dist](#) (`distarray.localapi.maps.BlockCyclicMap` attribute), 40  
[dist](#) (`distarray.localapi.maps.BlockMap` attribute), 40  
[dist](#) (`distarray.localapi.maps.CyclicMap` attribute), 41  
[dist](#) (`distarray.localapi.maps.Distribution` attribute), 41  
[dist](#) (`distarray.localapi.maps.UnstructuredMap` attribute), 42  
[DistArray](#) (class in `distarray.globalapi.distarray`), 24  
[distarray.\\_\\_init\\_\\_](#) (module), 11

[distarray.\\_\\_version\\_\\_ \(module\), 11](#)  
[distarray.apps.\\_\\_init\\_\\_ \(module\), 18](#)  
[distarray.apps.dacluster \(module\), 19](#)  
[distarray.apps.engine \(module\), 19](#)  
[distarray.error \(module\), 11](#)  
[distarray.globalapi.\\_\\_init\\_\\_ \(module\), 19](#)  
[distarray.globalapi.context \(module\), 19](#)  
[distarray.globalapi.distarray \(module\), 24](#)  
[distarray.globalapi.functions \(module\), 25](#)  
[distarray.globalapi.ipython\\_cleanup \(module\), 27](#)  
[distarray.globalapi.ipython\\_utils \(module\), 27](#)  
[distarray.globalapi.maps \(module\), 27](#)  
[distarray.globalapi.random \(module\), 31](#)  
[distarray.localapi.\\_\\_init\\_\\_ \(module\), 33](#)  
[distarray.localapi.construct \(module\), 33](#)  
[distarray.localapi.error \(module\), 33](#)  
[distarray.localapi.format \(module\), 34](#)  
[distarray.localapi.localarray \(module\), 35](#)  
[distarray.localapi.maps \(module\), 40](#)  
[distarray.localapi.mpiutils \(module\), 42](#)  
[distarray.localapi.proxyize \(module\), 42](#)  
[distarray.localapi.random \(module\), 43](#)  
[distarray.metadata\\_utils \(module\), 12](#)  
[distarray.mpi\\_engine \(module\), 14](#)  
[distarray.mpionly\\_utils \(module\), 15](#)  
[distarray.plotting.\\_\\_init\\_\\_ \(module\), 44](#)  
[distarray.plotting.plotting \(module\), 44](#)  
[distarray.run\\_tests \(module\), 15](#)  
[distarray.testing \(module\), 15](#)  
[distarray.utils \(module\), 17](#)  
[distarray\\_random\\_getstate\(\) \(in module distarray.utils\), 17](#)  
[distarray\\_random\\_setstate\(\) \(in module distarray.utils\), 17](#)  
[DistArrayError, 11](#)  
[distribute\\_block\\_indices\(\) \(in module distarray.metadata\\_utils\), 12](#)  
[distribute\\_cyclic\\_indices\(\) \(in module distarray.metadata\\_utils\), 12](#)  
[distribute\\_indices\(\) \(in module distarray.metadata\\_utils\), 12](#)  
[Distribution \(class in distarray.globalapi.maps\), 28](#)  
[Distribution \(class in distarray.localapi.maps\), 41](#)  
[DistributionError, 11](#)  
[divide\(\) \(in module distarray.globalapi.functions\), 26](#)  
[divisors\\_minmax\(\) \(in module distarray.utils\), 17](#)  
[dtype \(distarray.globalapi.distarray.DistArray attribute\), 24](#)  
[dtype \(distarray.localapi.localarray.LocalArray attribute\), 36](#)

## E

[empty\(\) \(distarray.globalapi.context.BaseContext method\), 20](#)

[empty\(\) \(in module distarray.localapi.localarray\), 37](#)  
[empty\\_like\(\) \(in module distarray.localapi.localarray\), 38](#)  
[Engine \(class in distarray.mpi\\_engine\), 14](#)  
[engine\\_make\\_targets\\_comm\(\) \(distarray.mpi\\_engine.Engine method\), 14](#)  
[equal\(\) \(in module distarray.globalapi.functions\), 26](#)  
[execute\(\) \(distarray.mpi\\_engine.Engine method\), 14](#)  
[exp\(\) \(in module distarray.globalapi.functions\), 25](#)  
[expm1\(\) \(in module distarray.globalapi.functions\), 26](#)

## F

[fill\(\) \(distarray.globalapi.distarray.DistArray method\), 24](#)  
[fill\(\) \(distarray.localapi.localarray.LocalArray method\), 36](#)  
[flatten\(\) \(in module distarray.utils\), 17](#)  
[floor\\_divide\(\) \(in module distarray.globalapi.functions\), 26](#)  
[fmod\(\) \(in module distarray.globalapi.functions\), 26](#)  
[free\\_comm\(\) \(distarray.mpi\\_engine.Engine method\), 14](#)  
[from\\_axis\\_dim\\_dicts\(\) \(distarray.globalapi.maps.BlockCyclicMap class method\), 27](#)  
[from\\_axis\\_dim\\_dicts\(\) \(distarray.globalapi.maps.BlockMap class method\), 28](#)  
[from\\_axis\\_dim\\_dicts\(\) \(distarray.globalapi.maps.MapBase class method\), 30](#)  
[from\\_axis\\_dim\\_dicts\(\) \(distarray.globalapi.maps.NoDistMap class method\), 31](#)  
[from\\_axis\\_dim\\_dicts\(\) \(distarray.globalapi.maps.UnstructuredMap class method\), 31](#)  
[from\\_dim\\_data\\_per\\_rank\(\) \(distarray.globalapi.maps.Distribution class method\), 28](#)  
[from\\_distarray\(\) \(distarray.localapi.localarray.LocalArray class method\), 36](#)  
[from\\_global\\_dim\\_data\(\) \(distarray.globalapi.maps.Distribution class method\), 28](#)  
[from\\_global\\_dim\\_dict\(\) \(distarray.globalapi.maps.BlockCyclicMap class method\), 27](#)  
[from\\_global\\_dim\\_dict\(\) \(distarray.globalapi.maps.BlockMap class method\), 28](#)  
[from\\_global\\_dim\\_dict\(\) \(distarray.globalapi.maps.MapBase class method\), 30](#)  
[from\\_global\\_dim\\_dict\(\) \(distarray.globalapi.maps.NoDistMap class method\), 30](#)

31  
from\_global\_dim\_dict() (distarray.globalapi.maps.UnstructuredMap class method), 31  
from\_localarrays() (distarray.globalapi.distarray.DistArray class method), 24  
from\_maps() (distarray.globalapi.maps.Distribution class method), 29  
from\_shape() (distarray.localapi.maps.Distribution class method), 41  
fromarray() (distarray.globalapi.context.BaseContext method), 20  
fromfunction() (distarray.globalapi.context.BaseContext method), 20  
fromfunction() (in module distarray.localapi.localarray), 38  
fromndarray() (distarray.globalapi.context.BaseContext method), 20  
fromndarray\_like() (in module distarray.localapi.localarray), 38  
func\_call() (distarray.mpi\_engine.Engine method), 14

## G

get\_comm\_private() (in module distarray.localapi.mpiutils), 42  
get\_comm\_world() (in module distarray.mpionly\_utils), 15  
get\_dim\_data\_per\_rank() (distarray.globalapi.maps.Distribution method), 30  
get\_dimdicts() (distarray.globalapi.maps.BlockCyclicMap method), 27  
get\_dimdicts() (distarray.globalapi.maps.BlockMap method), 28  
get\_dimdicts() (distarray.globalapi.maps.MapBase method), 30  
get\_dimdicts() (distarray.globalapi.maps.NoDistMap method), 31  
get\_dimdicts() (distarray.globalapi.maps.UnstructuredMap method), 31  
get\_from\_dotted\_name() (in module distarray.utils), 17  
get\_local\_keys() (in module distarray.globalapi.ipython\_cleanup), 27  
get\_localarrays() (distarray.globalapi.distarray.DistArray method), 24  
get\_ndarrays() (distarray.globalapi.distarray.DistArray method), 24  
get\_engines() (in module distarray.mpionly\_utils), 15  
get\_printoptions() (in module distarray.localapi.localarray), 38  
get\_ranks() (in module distarray.plotting.plotting), 45  
get\_slice() (distarray.localapi.localarray.GlobalIndex method), 35

get\_world\_rank() (in module distarray.mpionly\_utils), 15  
global\_from\_local() (distarray.localapi.localarray.LocalArray method), 37  
global\_from\_local() (distarray.localapi.maps.Distribution method), 41  
global\_from\_local\_index() (distarray.localapi.maps.BlockCyclicMap method), 40  
global\_from\_local\_index() (distarray.localapi.maps.BlockMap method), 40  
global\_from\_local\_index() (distarray.localapi.maps.CyclicMap method), 41  
global\_from\_local\_index() (distarray.localapi.maps.UnstructuredMap method), 42  
global\_from\_local\_slice() (distarray.localapi.maps.BlockMap method), 40  
global\_iter (distarray.localapi.maps.BlockCyclicMap attribute), 40  
global\_iter (distarray.localapi.maps.BlockMap attribute), 40  
global\_iter (distarray.localapi.maps.CyclicMap attribute), 41  
global\_iter (distarray.localapi.maps.UnstructuredMap attribute), 42  
global\_limits() (distarray.localapi.localarray.LocalArray method), 37  
global\_shape (distarray.localapi.localarray.LocalArray attribute), 37  
global\_shape (distarray.localapi.maps.Distribution attribute), 41  
global\_size (distarray.globalapi.distarray.DistArray attribute), 24  
global\_size (distarray.localapi.localarray.LocalArray attribute), 37  
global\_size (distarray.localapi.maps.Distribution attribute), 41  
global\_slice (distarray.localapi.maps.BlockCyclicMap attribute), 40  
global\_slice (distarray.localapi.maps.BlockMap attribute), 40  
global\_slice (distarray.localapi.maps.CyclicMap attribute), 41  
global\_slice (distarray.localapi.maps.Distribution attribute), 41  
GlobalIndex (class in distarray.localapi.localarray), 35  
GlobalIterator (class in distarray.localapi.localarray), 36  
greater() (in module distarray.globalapi.functions), 27  
greater\_equal() (in module distarray.globalapi.functions), 27  
grid\_shape (distarray.globalapi.distarray.DistArray attribute), 24  
grid\_shape (distarray.localapi.localarray.LocalArray at-



tribute), 37  
 grid\_shape (distarray.localapi.maps.Distribution attribute), 41  
 GridShapeError, 12

## H

has\_exactly\_one() (in module distarray.utils), 17  
 has\_precise\_index (distarray.globalapi.maps.Distribution attribute), 30  
 hypot() (in module distarray.globalapi.functions), 26

## I

import\_or\_skip() (in module distarray.testing), 16  
 IncompatibleArrayError, 33  
 index\_owners() (distarray.globalapi.maps.BlockCyclicMap method), 27  
 index\_owners() (distarray.globalapi.maps.BlockMap method), 28  
 index\_owners() (distarray.globalapi.maps.MapBase method), 30  
 index\_owners() (distarray.globalapi.maps.NoDistMap method), 31  
 index\_owners() (distarray.globalapi.maps.UnstructuredMap method), 31  
 init\_base\_comm() (in module distarray.localapi.construct), 33  
 init\_comm() (in module distarray.localapi.construct), 33  
 initial\_comm\_setup() (in module distarray.mpionly\_utils), 15  
 INTERCOMM (distarray.globalapi.context.MPICContext attribute), 23  
 INTERCOMM (distarray.mpi\_engine.Engine attribute), 14  
 InvalidBaseCommError, 33  
 InvalidCommSizeError, 11  
 InvalidDimensionError, 33  
 InvalidGridShapeError, 12  
 InvalidRankError, 11  
 invert() (in module distarray.globalapi.functions), 26  
 IPythonContext (class in distarray.globalapi.context), 22  
 IPythonContextTestCase (class in distarray.testing), 16  
 is\_compatible() (distarray.globalapi.maps.BlockCyclicMap method), 27  
 is\_compatible() (distarray.globalapi.maps.BlockMap method), 28  
 is\_compatible() (distarray.globalapi.maps.Distribution method), 30  
 is\_compatible() (distarray.globalapi.maps.MapBase method), 31  
 is\_compatible() (distarray.globalapi.maps.NoDistMap method), 31

is\_engine() (distarray.mpi\_engine.Engine method), 14  
 is\_solo\_mpi\_process() (in module distarray.mpionly\_utils), 15  
 itemsize (distarray.globalapi.distarray.DistArray attribute), 24  
 itemsize (distarray.localapi.localarray.LocalArray attribute), 37

## K

kill() (distarray.mpi\_engine.Engine method), 14

## L

label\_state() (in module distarray.localapi.random), 43  
 left\_shift() (in module distarray.globalapi.functions), 26  
 less() (in module distarray.globalapi.functions), 26  
 less\_equal() (in module distarray.globalapi.functions), 26  
 list\_or\_tuple() (in module distarray.utils), 17  
 load\_dnpv() (distarray.globalapi.context.BaseContext method), 20  
 load\_dnpv() (in module distarray.localapi.localarray), 38  
 load\_hdf5() (distarray.globalapi.context.BaseContext method), 21  
 load\_hdf5() (in module distarray.localapi.localarray), 38  
 load\_npy() (distarray.globalapi.context.BaseContext method), 21  
 load\_npy() (in module distarray.localapi.localarray), 38  
 local\_data (distarray.localapi.localarray.LocalArray attribute), 37  
 local\_from\_global() (distarray.localapi.localarray.LocalArray method), 37  
 local\_from\_global() (distarray.localapi.maps.Distribution method), 41  
 local\_from\_global\_index() (distarray.localapi.maps.BlockCyclicMap method), 40  
 local\_from\_global\_index() (distarray.localapi.maps.BlockMap method), 40  
 local\_from\_global\_index() (distarray.localapi.maps.CyclicMap method), 41  
 local\_from\_global\_index() (distarray.localapi.maps.UnstructuredMap method), 42  
 local\_from\_global\_slice() (distarray.localapi.maps.BlockMap method), 40  
 local\_reduction() (in module distarray.localapi.localarray), 38  
 local\_shape (distarray.localapi.localarray.LocalArray attribute), 37  
 local\_shape (distarray.localapi.maps.Distribution attribute), 41  
 local\_size (distarray.localapi.localarray.LocalArray attribute), 37

`local_size` (`distarray.localapi.maps.Distribution` attribute), 41  
`local_view()` (`distarray.localapi.localarray.LocalArray` method), 37  
`LocalArray` (class in `distarray.localapi.localarray`), 36  
`LocalArrayBinaryOperation` (class in `distarray.localapi.localarray`), 37  
`LocalArrayUnaryOperation` (class in `distarray.localapi.localarray`), 37  
`localshapes()` (`distarray.globalapi.distarray.DistArray` method), 24  
`localshapes()` (`distarray.globalapi.maps.Distribution` method), 30  
`log()` (in module `distarray.globalapi.functions`), 26  
`log10()` (in module `distarray.globalapi.functions`), 26  
`log1p()` (in module `distarray.globalapi.functions`), 26

## M

`magic()` (in module `distarray.localapi.format`), 34  
`main()` (in module `distarray.apps.dacluster`), 19  
`make_base_comm()` (in module `distarray.mpionly_utils`), 15  
`make_context()` (`distarray.testing.DefaultContextTestCase` class method), 16  
`make_context()` (`distarray.testing.IPythonContextTestCase` class method), 16  
`make_context()` (`distarray.testing.MPIContextTestCase` class method), 16  
`make_grid_shape()` (in module `distarray.metadata_utils`), 12  
`make_intercomm()` (in module `distarray.mpionly_utils`), 15  
`make_subcomm()` (`distarray.globalapi.context.BaseContext` method), 21  
`make_subcomm()` (`distarray.globalapi.context.IPythonContext` method), 23  
`make_subcomm()` (`distarray.globalapi.context.MPIContext` method), 24  
`make_targets_comm()` (in module `distarray.mpionly_utils`), 15  
`map_from_dim_dict()` (in module `distarray.localapi.maps`), 42  
`map_from_global_dim_dict()` (in module `distarray.globalapi.maps`), 31  
`map_from_sizes()` (in module `distarray.globalapi.maps`), 31  
`MapBase` (class in `distarray.globalapi.maps`), 30  
`MapBase` (class in `distarray.localapi.maps`), 41

`max()` (`distarray.globalapi.distarray.DistArray` method), 24  
`max_reducer()` (in module `distarray.localapi.localarray`), 39  
`mean()` (`distarray.globalapi.distarray.DistArray` method), 24  
`mean_reducer()` (in module `distarray.localapi.localarray`), 39  
`min()` (`distarray.globalapi.distarray.DistArray` method), 24  
`min_reducer()` (in module `distarray.localapi.localarray`), 39  
`mirror_sort()` (in module `distarray.utils`), 18  
`mod()` (in module `distarray.globalapi.functions`), 26  
`mpi_type_for_ndarray()` (in module `distarray.localapi.mpiutils`), 42  
`MPIContext` (class in `distarray.globalapi.context`), 23  
`MPIContextTestCase` (class in `distarray.testing`), 16  
`MPIDistArrayError`, 12  
`mult_partitions()` (in module `distarray.utils`), 18  
`mult_partitions_rekurs()` (in module `distarray.utils`), 18  
`multi_for()` (in module `distarray.utils`), 18  
`multiply()` (in module `distarray.globalapi.functions`), 26

## N

`nbytes` (`distarray.globalapi.distarray.DistArray` attribute), 25  
`nbytes` (`distarray.localapi.localarray.LocalArray` attribute), 37  
`ndarray` (`distarray.localapi.localarray.LocalArray` attribute), 37  
`ndenumerate()` (in module `distarray.localapi.localarray`), 39  
`ndim` (`distarray.globalapi.distarray.DistArray` attribute), 25  
`ndim` (`distarray.localapi.localarray.LocalArray` attribute), 37  
`ndim` (`distarray.localapi.maps.Distribution` attribute), 41  
`negative()` (in module `distarray.globalapi.functions`), 26  
`next_name()` (`distarray.localapi.proxyize.Proxyize` method), 42  
`NoDistMap` (class in `distarray.globalapi.maps`), 31  
`non_dist_size()` (in module `distarray.metadata_utils`), 13  
`normal()` (`distarray.globalapi.random.Random` method), 31  
`normal()` (in module `distarray.localapi.random`), 43  
`normalize_dim_dict()` (in module `distarray.metadata_utils`), 13  
`normalize_dist()` (in module `distarray.metadata_utils`), 13  
`normalize_grid_shape()` (in module `distarray.metadata_utils`), 13  
`normalize_reduction_axes()` (in module `distarray.metadata_utils`), 13  
`not_equal()` (in module `distarray.globalapi.functions`), 26

ntargets (distarray.testing.BaseContextTestCase attribute), 15  
 NullCommError, 33

## O

ones() (distarray.globalapi.context.BaseContext method), 21  
 ones() (in module distarray.localapi.localarray), 39  
 owning\_ranks() (distarray.globalapi.maps.Distribution method), 30  
 owning\_targets() (distarray.globalapi.maps.Distribution method), 30

## P

pack\_index() (distarray.localapi.localarray.LocalArray method), 37  
 ParallelTestCase (class in distarray.testing), 16  
 parse\_msg() (distarray.mpi\_engine.Engine method), 14  
 plot\_array\_distribution() (in module distarray.plotting.plotting), 45  
 plot\_local\_array\_subfigure() (in module distarray.plotting.plotting), 45  
 plot\_local\_arrays() (in module distarray.plotting.plotting), 45  
 positivify() (in module distarray.metadata\_utils), 13  
 power() (in module distarray.globalapi.functions), 26  
 Proxy (class in distarray.localapi.proxyize), 42  
 Proxyize (class in distarray.localapi.proxyize), 42  
 pull() (distarray.mpi\_engine.Engine method), 14  
 push() (distarray.mpi\_engine.Engine method), 14  
 push\_function() (distarray.globalapi.context.BaseContext method), 21  
 push\_function() (distarray.globalapi.context.IPythonContext method), 23  
 push\_function() (distarray.globalapi.context.MPICContext method), 24  
 push\_function() (in module distarray.mpionly\_utils), 15

## R

raise\_typeerror() (in module distarray.testing), 17  
 rand() (distarray.globalapi.random.Random method), 32  
 rand() (in module distarray.localapi.random), 43  
 randint() (distarray.globalapi.random.Random method), 32  
 randint() (in module distarray.localapi.random), 43  
 randn() (distarray.globalapi.random.Random method), 33  
 randn() (in module distarray.localapi.random), 44  
 Random (class in distarray.globalapi.random), 31  
 rank\_from\_coords() (distarray.localapi.localarray.LocalArray method), 37  
 rank\_from\_coords() (distarray.localapi.maps.Distribution method), 41

read\_array\_header\_1\_0() (in module distarray.localapi.format), 34  
 read\_localarray() (in module distarray.localapi.format), 35  
 read\_magic() (in module distarray.localapi.format), 35  
 reciprocal() (in module distarray.globalapi.functions), 26  
 reduce() (distarray.globalapi.maps.Distribution method), 30  
 register() (distarray.globalapi.context.BaseContext method), 21  
 remainder() (in module distarray.globalapi.functions), 26  
 remove\_elements() (in module distarray.utils), 18  
 restart() (in module distarray.apps.dacluster), 19  
 right\_shift() (in module distarray.globalapi.functions), 26  
 rint() (in module distarray.globalapi.functions), 26

## S

sanitize\_indices() (in module distarray.metadata\_utils), 13  
 save\_dnp() (distarray.globalapi.context.BaseContext method), 21  
 save\_dnp() (in module distarray.localapi.localarray), 39  
 save\_hdf5() (distarray.globalapi.context.BaseContext method), 22  
 save\_hdf5() (in module distarray.localapi.localarray), 39  
 seed() (distarray.globalapi.random.Random method), 33  
 set\_from\_dotted\_name() (in module distarray.utils), 18  
 set\_printoptions() (in module distarray.localapi.localarray), 39  
 set\_state() (distarray.localapi.proxyize.Proxyize method), 42  
 setup\_engine\_comm() (in module distarray.mpionly\_utils), 15  
 setUpClass() (distarray.testing.BaseContextTestCase class method), 15  
 setUpClass() (distarray.testing.IPythonContextTestCase class method), 16  
 setUpClass() (distarray.testing.ParallelTestCase class method), 16  
 shape (distarray.globalapi.distarray.DistArray attribute), 25  
 shapes\_from\_dim\_data\_per\_rank() (in module distarray.metadata\_utils), 14  
 sign() (in module distarray.globalapi.functions), 26  
 sin() (in module distarray.globalapi.functions), 26  
 sinh() (in module distarray.globalapi.functions), 26  
 size (distarray.localapi.maps.BlockCyclicMap attribute), 40  
 size (distarray.localapi.maps.BlockMap attribute), 40  
 size (distarray.localapi.maps.CyclicMap attribute), 41  
 size (distarray.localapi.maps.UnstructuredMap attribute), 42  
 size\_chooser() (in module distarray.metadata\_utils), 14

size\_from\_dim\_data() (in module distarray.metadata\_utils), 14

slice() (distarray.globalapi.maps.BlockMap method), 28

slice() (distarray.globalapi.maps.Distribution method), 30

slice() (distarray.globalapi.maps.NoDistMap method), 31

slice\_intersection() (in module distarray.utils), 18

slice\_owners() (distarray.globalapi.maps.BlockMap method), 28

slice\_owners() (distarray.globalapi.maps.NoDistMap method), 31

sqr() (in module distarray.globalapi.functions), 26

square() (in module distarray.globalapi.functions), 26

start() (in module distarray.apps.dacluster), 19

std() (distarray.globalapi.distarray.DistArray method), 25

std\_reducer() (in module distarray.localapi.localarray), 39

stop() (in module distarray.apps.dacluster), 19

str\_counter() (distarray.localapi.proxyize.Proxyize method), 42

subtract() (in module distarray.globalapi.functions), 26

sum() (distarray.globalapi.distarray.DistArray method), 25

sum\_reducer() (in module distarray.localapi.localarray), 39

sync() (distarray.localapi.localarray.LocalArray method), 37

## T

tan() (in module distarray.globalapi.functions), 26

tanh() (in module distarray.globalapi.functions), 26

targets (distarray.globalapi.distarray.DistArray attribute), 25

tearDownClass() (distarray.testing.BaseContextTestCase class method), 15

tearDownClass() (distarray.testing.ParallelTestCase class method), 16

temp\_filepath() (in module distarray.testing), 17

test() (in module distarray.run\_tests), 15

toarray() (distarray.globalapi.distarray.DistArray method), 25

tondarray() (distarray.globalapi.distarray.DistArray method), 25

true\_divide() (in module distarray.globalapi.functions), 26

tuple\_intersection() (in module distarray.metadata\_utils), 14

## U

uid() (in module distarray.utils), 18

unpack\_index() (distarray.localapi.localarray.LocalArray method), 37

unstructured\_size() (in module distarray.metadata\_utils), 14

UnstructuredMap (class in distarray.globalapi.maps), 31

UnstructuredMap (class in distarray.localapi.maps), 42

update\_count() (distarray.utils.count\_round\_trips method), 17

## V

var() (distarray.globalapi.distarray.DistArray method), 25

var\_reducer() (in module distarray.localapi.localarray), 40

view() (distarray.globalapi.distarray.DistArray method), 25

view() (distarray.globalapi.maps.BlockMap method), 28

view() (distarray.globalapi.maps.Distribution method), 30

view() (distarray.globalapi.maps.NoDistMap method), 31

view() (distarray.localapi.localarray.LocalArray method), 37

## W

write\_localarray() (in module distarray.localapi.format), 35

## Z

zeros() (distarray.globalapi.context.BaseContext method), 22

zeros() (in module distarray.localapi.localarray), 40

zeros\_like() (in module distarray.localapi.localarray), 40