array$_{s}plit Documentation$

Release 0.1.0

Shane J. Latham

October 27, 2016

Contents

1 array_split Read Me **3**
- 1.1 Quick Start Example . . . 3
- 1.2 Installation . . . 4
- 1.3 Requirements . . . 4
- 1.4 Testing . . . 4
- 1.5 Documentation . . . 4
- 1.6 Latest source code . . . 5
- 1.7 License information . . . 5

2 array_split Examples **7**
- 2.1 Terminology . . . 7
- 2.2 Parameter Categories . . . 7
- 2.3 Import statements for the examples . . . 8
- 2.4 Comparison between array_split, shape_split and ShapeSplitter . . . 8
- 2.5 Splitting by number of tiles . . . 9
  - 2.5.1 Single axis number of tiles . . . 9
  - 2.5.2 Multiple axes number of tiles . . . 9
- 2.6 Splitting by per-axis cut indices . . . 10
  - 2.6.1 Single axis cut indices . . . 10
  - 2.6.2 Multiple axes cut indices . . . 11
- 2.7 Splitting by tile shape . . . 11
- 2.8 Splitting by maximum bytes per tile . . . 12
  - 2.8.1 Tile shape upper bound constraint . . . 13
  - 2.8.2 Sub-tile shape constraint . . . 13
- 2.9 The *array_start* parameter . . . 14
- 2.10 The *halo* parameter . . . 14

3 API Reference **17**
- 3.1 The array_split Package . . . 17
  - 3.1.1 Classes and Functions . . . 17
    - array_split.shape_split . . . 18
    - array_split.array_split . . . 19
    - array_split.ShapeSplitter . . . 19
  - 3.1.2 Attributes . . . 27
- 3.2 The array_split.split Module . . . 27
  - 3.2.1 Classes and Functions . . . 27
    - array_split.split.shape_factors . . . 28
    - array_split.split.calculate_num_slices_per_axis . . . 28

array_split.split.calculate_tile_shape_for_max_bytes . . . 29
array_split.split.ShapeSplitter . . . 30
array_split.split.shape_split . . . 37
array_split.split.array_split . . . 38
3.2.2 Attributes . . . 39
3.3 The array_split.split_test Module . . . 39
3.3.1 Classes . . . 40
array_split.split_test.SplitTest . . . 41
3.4 The array_split.tests Module . . . 53
3.5 The array_split.logging Module . . . 53
3.5.1 Classes and Functions . . . 53
array_split.logging.SplitStreamHandler . . . 54
array_split.logging.initialise_loggers . . . 57
array_split.logging.get_formatter . . . 57
3.6 The array_split.unittest Module . . . 58
3.6.1 Classes and Functions . . . 58
array_split.unittest.main . . . 58
array_split.unittest.TestCase . . . 58
3.7 The array_split.license Module . . . 69
3.7.1 License . . . 69
3.7.2 Copyright . . . 69
3.7.3 Functions . . . 69
array_split.license.license . . . 69
array_split.license.copyright . . . 70

Python Module Index **71**

Release 0.1.0

Version 0.1.0

Date October 27, 2016

CHAPTER 1

array_split Read Me

The array_split python package is a modest enhancement to the numpy.array_split function for sub-dividing multi-dimensional arrays into sub-arrays (slices). The main motivation comes from parallel processing where one desires to split (decompose) a large array (or multiple arrays) into smaller sub-arrays which can be processed concurrently by other processes (multiprocessing or mpi4py) or other memory-limited hardware (e.g. GPGPU using pyopencl, pycuda, etc).

1.1 Quick Start Example

```
>>> from array_split import array_split, shape_split
>>> import numpy as np
>>>
>>> ary = np.arange(0, 4*9)
>>>
>>> array_split(ary, 4) # 1D split into 4 sections (like numpy.array_split)
[array([0, 1, 2, 3, 4, 5, 6, 7, 8]),
 array([ 9, 10, 11, 12, 13, 14, 15, 16, 17]),
 array([18, 19, 20, 21, 22, 23, 24, 25, 26]),
 array([27, 28, 29, 30, 31, 32, 33, 34, 35])]
>>>
>>> shape_split(ary.shape, 4) # 1D split into 4 sections, slice objects instead of numpy.ndarray view
array([(slice(0, 9, None),), (slice(9, 18, None),), (slice(18, 27, None),), (slice(27, 36, None),)],
      dtype=[('0', 'O')])
>>>
>>> ary = ary.reshape(4, 9) # Make ary 2D
>>> split = shape_split(ary.shape, axis=(2, 3)) # 2D split into 2*3=6 sections
>>> split.shape
(2, 3)
>>> split
array([[(slice(0, 2, None), slice(0, 3, None)),
        (slice(0, 2, None), slice(3, 6, None)),
        (slice(0, 2, None), slice(6, 9, None))],
       [(slice(2, 4, None), slice(0, 3, None)),
        (slice(2, 4, None), slice(3, 6, None)),
        (slice(2, 4, None), slice(6, 9, None))]],
      dtype=[('0', 'O'), ('1', 'O')])
>>> sub_arys = [ary[tup] for tup in split.flatten()] # Split ary into sub-array views using the slice
>>> sub_arys
[array([[ 0,  1,  2], [ 9, 10, 11]]),
 array([[ 3,  4,  5], [12, 13, 14]]),
 array([[ 6,  7,  8], [15, 16, 17]]),
```

```
 array([[18, 19, 20], [27, 28, 29]]),
 array([[21, 22, 23], [30, 31, 32]]),
 array([[24, 25, 26], [33, 34, 35]])]
```

Latest sphinx documentation examples at http://array-split.readthedocs.io/en/latest/examples/.

1.2 Installation

Using `pip`:

```
pip install array_split # with root access
```

or:

```
pip install --user array_split # no root/sudo permissions required
```

From latest github source:

```
git clone https://github.com/array-split/array_split.git
cd array_split
python setup.py install --user
```

1.3 Requirements

Requires numpy version >= *1.6*, python-2 version >= *2.6* or python-3 version >= *3.2*.

1.4 Testing

Run tests (unit-tests and doctest module docstring tests) using:

```
python -m array_split.tests
```

or, from the source tree, run:

```
python setup.py test
```

Travis CI at:

https://travis-ci.org/array-split/array_split/

1.5 Documentation

Latest sphinx generated documentation is at:

http://array-split.readthedocs.io/en/latest

1.6 Latest source code

Source at github:

https://github.com/array-split/array_split

1.7 License information

See the file LICENSE.txt for terms & conditions, for usage and a DISCLAIMER OF ALL WARRANTIES.

CHAPTER 2

`array_split` Examples

2.1 Terminology

Definitions:

tile A multi-dimensional *sub-array* of an array (e.g. `numpy.ndarray`) decomposition.

slice A `tuple` of `slice` elements defining the extents of a tile/sub-array.

cut A *division* along an axis to form tiles or slices.

split The sub-division (tiling) of an array (or an array shape) resulting from cuts.

halo Per-axis number of elements which specifies the expansion of a tile (in the negative and positive axis directions) to form an *overlap* of elements with neighbouring tiles. The *overlaps* are often referred to as *ghost cells* or *ghost elements*.

sub-tile A sub-array of a tile.

2.2 Parameter Categories

There are four categories of parameters for specifying a split:

Number of tiles The total number of tiles and/or the number of slices per axis. The *`indices_or_sections`* parameter can specify the number of tiles in the resulting split (as an `int`).

Per-axis split indices The per-axis indices specifying where the array (shape) is to be cut. The *`indices_or_sections`* parameter doubles up to indicate the indices at which cuts are to occur.

Tile shape Explicitly specify the shape of the tile in a split. The *`tile_shape`* parameter (typically as a lone *keyword argument*) indicates the tile shape.

Tile maximum number of bytes Given the number of bytes per array element, a tile shape is calculated such that all tiles (including halo extension) of the resulting split do not exceed a specified (maximum) number of bytes. The *`array_itemsize`* parameter gives the number of bytes per array element and the *`max_tile_bytes`* parameter constrains the maximum number of bytes per tile.

The subsequent sections provides examples from each of these categories.

2.3 Import statements for the examples

In the examples of the following sections, we assume that the following statement has been issued to `import` the relevant functions:

```
>>> import numpy
>>> from array_split import array_split, shape_split, ShapeSplitter
```

2.4 Comparison between `array_split`, `shape_split` and `ShapeSplitter`

The *`array_split.array_split()`* function is analogous to the `numpy.array_split()` function. It takes a `numpy.ndarray` object as an argument and returns a `list` of tile (`numpy.ndarray` sub-array objects) elements:

```
>>> numpy.array_split(numpy.arange(0, 10), 3)
[array([0, 1, 2, 3]), array([4, 5, 6]), array([7, 8, 9])]
>>> array_split(numpy.arange(0, 10), 3) # array_split.array_split
[array([0, 1, 2, 3]), array([4, 5, 6]), array([7, 8, 9])]
```

The *`array_split.shape_split()`* function takes an array *shape* as an argument instead of an actual array, and returns a `numpy` structured array of `tuple` elements. The tuple elements can then be used to generate the tiles from a `numpy.ndarray` of an equivalent shape:

```
>>> ary = numpy.arange(0, 10)
>>> split = shape_split(ary.shape, 3) # returns array of tuples
>>> split
array([(slice(0, 4, None),), (slice(4, 7, None),), (slice(7, 10, None),)],
      dtype=[('0', 'O')])
>>> [ary[slyce] for slyce in split.flatten()] # generates tile views of ary
[array([0, 1, 2, 3]), array([4, 5, 6]), array([7, 8, 9])]
```

Each `tuple` array element, of the returned split, has length equal to the dimension of the multi-dimensional shape, i.e. `N = len(`*`array_shape`*`)`. Each `tuple` indicates the indexing extent of a tile.

The *`array_split.ShapeSplitter`* class contains the bulk of the split implementation for the *`array_split.shape_split()`*. The *`array_split.ShapeSplitter.__init__()`* constructor takes the same arguments as the *`array_split.shape_split()`* function and the *`array_split.ShapeSplitter.calculate_split()`* method computes the split. After the split computation, some state information is preserved in the *`array_split.ShapeSplitter`* data attributes:

```
>>> ary = numpy.arange(0, 10)
>>> splitter = ShapeSplitter(ary.shape, 3)
>>> split = splitter.calculate_split()
>>> split.shape
(3,)
>>> split
array([(slice(0, 4, None),), (slice(4, 7, None),), (slice(7, 10, None),)],
      dtype=[('0', 'O')])
>>> [ary[slyce] for slyce in split.flatten()]
[array([0, 1, 2, 3]), array([4, 5, 6]), array([7, 8, 9])]
>>>
>>> splitter.split_shape # equivalent to split.shape above
array([3])
>>> splitter.split_begs  # start indices for tile extents
[array([0, 4, 7])]
```

```
>>> splitter.split_ends  # stop indices for tile extents
[array([ 4,  7, 10])]
```

Methods of the *array_split.ShapeSplitter* class can be over-ridden in sub-classes in order to customise the splitting behaviour.

The examples of the following section explicitly illustrate the behaviour for the *array_split.shape_split()* function, but with minor modifications, the examples are also relevant for the *array_split.array_split()* function and for instances of the *array_split.ShapeSplitter* class.

2.5 Splitting by number of tiles

2.5.1 Single axis number of tiles

When the *indices_or_sections* parameter is specified as an integer (scalar), it specifies the number of tiles in the returned split:

```
>>> split = shape_split([20,], 4)  # 1D, array_shape=[20,], number of tiles=4, default axis=0
>>> split.shape
(4,)
>>> split
array([(slice(0, 5, None),), (slice(5, 10, None),), (slice(10, 15, None),),
       (slice(15, 20, None),)],
      dtype=[('0', 'O')])
```

By default, cuts are made along the *axis* = 0 axis. In the multi-dimensional case, one can over-ride the axis using the *axis* parameter, e.g. for a 2D shape:

```
>>> split = shape_split([20,10], 4, axis=1)  # Split along axis=1
>>> split.shape
(1, 4)
>>> split
array([[(slice(0, 20, None), slice(0, 3, None)),
        (slice(0, 20, None), slice(3, 6, None)),
        (slice(0, 20, None), slice(6, 8, None)),
        (slice(0, 20, None), slice(8, 10, None))]],
      dtype=[('0', 'O'), ('1', 'O')])
```

2.5.2 Multiple axes number of tiles

The *axis* parameter can also be used to specify the number of slices (sections) per-axis:

```
>>> split = shape_split([20, 10], axis=[3, 2])  # Cut into 3*2=6 tiles
>>> split.shape
(3, 2)
>>> split
array([[(slice(0, 7, None), slice(0, 5, None)),
        (slice(0, 7, None), slice(5, 10, None))],
       [(slice(7, 14, None), slice(0, 5, None)),
        (slice(7, 14, None), slice(5, 10, None))],
       [(slice(14, 20, None), slice(0, 5, None)),
        (slice(14, 20, None), slice(5, 10, None))]],
      dtype=[('0', 'O'), ('1', 'O')])
```

The array axis 0 has been cut into three sections and axis 1 has been cut into two sections for a total of `3*2 = 6` tiles. In general, if *`axis`* is an integer (scalar) it indicates the single axis which is to be cut to form slices. When *`axis`* is a sequence, then *`axis`*`[i]` indicates the number of sections into which axis `i` is to be cut.

In addition, one can also specify a total number of tiles and use the *`axis`* parameter to limit which axes are to be cut by specifying non-positive values for elements of the *`axis`* sequence. For example, in 3D, cut into 8 tiles, but only cut the `axis=1` and `axis=2` axes:

```
>>> split = shape_split([20, 10, 15], 8, axis=[1, 0, 0])  # Cut into 1*?*?=8 tiles
>>> split.shape
(1, 4, 2)
>>> split
array([[[(slice(0, 20, None), slice(0, 3, None), slice(0, 8, None)),
         (slice(0, 20, None), slice(0, 3, None), slice(8, 15, None))],
        [(slice(0, 20, None), slice(3, 6, None), slice(0, 8, None)),
         (slice(0, 20, None), slice(3, 6, None), slice(8, 15, None))],
        [(slice(0, 20, None), slice(6, 8, None), slice(0, 8, None)),
         (slice(0, 20, None), slice(6, 8, None), slice(8, 15, None))],
        [(slice(0, 20, None), slice(8, 10, None), slice(0, 8, None)),
         (slice(0, 20, None), slice(8, 10, None), slice(8, 15, None))]]],
      dtype=[('0', 'O'), ('1', 'O'), ('2', 'O')])
```

In the above, non-positive elements of `axis` are replaced with positive values such that `numpy.product(axis)` equals the number of requested tiles (`= 8` above). Raises `ValueError` if the impossible is attempted:

```
>>> try:
...     split = shape_split([20, 10, 15], 8, axis=[1, 3, 0])  # Impossible to cut into 1*3*?=8 tiles
... except (ValueError,) as e:
...     e
...
ValueError('Unable to construct grid of num_slices=8 elements from num_slices_per_axis=[1, 3, 0] (wit
```

2.6 Splitting by per-axis cut indices

2.6.1 Single axis cut indices

The *`indices_or_sections`* parameter can also be used to specify the location (index values) of cuts:

```
>>> split = shape_split([20,], [5, 7, 9])  # 1D, split into 4 tiles, default cut axis=0
>>> split.shape
(4,)
>>> split
array([(slice(0, 5, None),), (slice(5, 7, None),), (slice(7, 9, None),),
       (slice(9, 20, None),)],
      dtype=[('0', 'O')])
```

Here, three cuts have been made to form `4` slices, cuts at index `5`, index `7` and index `9`.

Similarly, in 2D, the *`indices_or_sections`* cut indices can made along *`axis`* `= 1` only:

```
>>> split = shape_split([20, 13], [5, 7, 9], axis=1)  # 2D, cut into 4 tiles, cut axis=1
>>> split.shape
(1, 4)
>>> split
array([[(slice(0, 20, None), slice(0, 5, None)),
        (slice(0, 20, None), slice(5, 7, None)),
        (slice(0, 20, None), slice(7, 9, None)),
```

```
        (slice(0, 20, None), slice(9, 13, None))]],
      dtype=[('0', 'O'), ('1', 'O')])
```

2.6.2 Multiple axes cut indices

The *indices_or_sections* parameter can also be used to cut along multiple axes. In this case, the *indices_or_sections* parameter is specified as a *sequence of sequence*, so that *indices_or_sections*[i] specifies the cut indices along axis i. For example, in 3D, cut along axis=1 and axis=2 only:

```
>>> split = shape_split([20, 13, 64], [[], [7], [15, 30, 45]])  # 3D, split into 8 tiles, no cuts on
>>> split.shape
(1, 2, 4)
>>> split
array([[[(slice(0, 20, None), slice(0, 7, None), slice(0, 15, None)),
         (slice(0, 20, None), slice(0, 7, None), slice(15, 30, None)),
         (slice(0, 20, None), slice(0, 7, None), slice(30, 45, None)),
         (slice(0, 20, None), slice(0, 7, None), slice(45, 64, None))],
        [(slice(0, 20, None), slice(7, 13, None), slice(0, 15, None)),
         (slice(0, 20, None), slice(7, 13, None), slice(15, 30, None)),
         (slice(0, 20, None), slice(7, 13, None), slice(30, 45, None)),
         (slice(0, 20, None), slice(7, 13, None), slice(45, 64, None))]]],
      dtype=[('0', 'O'), ('1', 'O'), ('2', 'O')])
```

The *indices_or_sections*=[[], [7], [15, 30, 45]] parameter indicates that the cut indices for axis=0 are [] (i.e. no cuts), the cut indices for axis=1 are [7] (a single cut at index 7) and the cut indices for axis=2 are [15, 30, 45] (three cuts).

2.7 Splitting by tile shape

The tile shape can be explicitly set with the *tile_shape* parameter, e.g. in 1D:

```
>>> split = shape_split([20,], tile_shape=[6,])  # Cut into (6,) shaped tiles
>>> split.shape
(4,)
>>> split
array([(slice(0, 6, None),), (slice(6, 12, None),), (slice(12, 18, None),),
       (slice(18, 20, None),)],
      dtype=[('0', 'O')])
```

and 2D:

```
>>> split = shape_split([20, 32], tile_shape=[6, 16])  # Cut into (6, 16) shaped tiles
>>> split.shape
(4, 2)
>>> split
array([[(slice(0, 6, None), slice(0, 16, None)),
        (slice(0, 6, None), slice(16, 32, None))],
       [(slice(6, 12, None), slice(0, 16, None)),
        (slice(6, 12, None), slice(16, 32, None))],
       [(slice(12, 18, None), slice(0, 16, None)),
        (slice(12, 18, None), slice(16, 32, None))],
       [(slice(18, 20, None), slice(0, 16, None)),
        (slice(18, 20, None), slice(16, 32, None))]],
      dtype=[('0', 'O'), ('1', 'O')])
```

2.8 Splitting by maximum bytes per tile

Tile shape can constrained by specifying a maximum number of bytes per tile by specifying the `array_itemsize` and the `max_tile_bytes` parameters. In 1D:

```
>>> split = shape_split(
...   array_shape=[512,],
...   array_itemsize=1,
...   max_tile_bytes=512 # Equals number of array bytes
... )
...
>>> split.shape
(1,)
>>> split
array([(slice(0, 512, None),)],
      dtype=[('0', 'O')])
```

Double the array per-element number of bytes:

```
>>> split = shape_split(
...   array_shape=[512,],
...   array_itemsize=2,
...   max_tile_bytes=512 # Equals half the number of array bytes
... )
...
>>> split.shape
(2,)
>>> split
array([(slice(0, 256, None),), (slice(256, 512, None),)],
      dtype=[('0', 'O')])
```

Decrement *`max_tile_bytes`* to `511` to split into 3 tiles:

```
>>> split = shape_split(
...   array_shape=[512,],
...   array_itemsize=2,
...   max_tile_bytes=511 # Less than half the number of array bytes
... )
...
>>> split.shape
(3,)
>>> split
array([(slice(0, 171, None),), (slice(171, 342, None),),
       (slice(342, 512, None),)],
      dtype=[('0', 'O')])
```

Note that the split is calculated so that tiles are approximately equal in size.

In 2D:

```
>>> split = shape_split(
...   array_shape=[512, 1024],
...   array_itemsize=1,
...   max_tile_bytes=512*512
... )
...
>>> split.shape
(2, 1)
>>> split
```

```
array([[(slice(0, 256, None), slice(0, 1024, None))],
       [(slice(256, 512, None), slice(0, 1024, None))]],
      dtype=[('0', 'O'), ('1', 'O')])
```

and increasing *array_itemsize* to 4:

```
>>> split = shape_split(
...   array_shape=[512, 1024],
...   array_itemsize=4,
...   max_tile_bytes=512*512
... )
...
>>> split.shape
(8, 1)
>>> split
array([[(slice(0, 64, None), slice(0, 1024, None))],
       [(slice(64, 128, None), slice(0, 1024, None))],
       [(slice(128, 192, None), slice(0, 1024, None))],
       [(slice(192, 256, None), slice(0, 1024, None))],
       [(slice(256, 320, None), slice(0, 1024, None))],
       [(slice(320, 384, None), slice(0, 1024, None))],
       [(slice(384, 448, None), slice(0, 1024, None))],
       [(slice(448, 512, None), slice(0, 1024, None))]],
      dtype=[('0', 'O'), ('1', 'O')])
```

The preference is to cut into (`'C'` order) contiguous memory tiles.

2.8.1 Tile shape upper bound constraint

The split can be influenced by specifying the *max_tile_shape* parameter. For the previous 2D example, cuts can for forced along `axis=1` by constraining the tile shape:

```
>>> split = shape_split(
...   array_shape=[512, 1024],
...   array_itemsize=4,
...   max_tile_bytes=512*512,
...   max_tile_shape=[numpy.inf, 256]
... )
...
>>> split.shape
(2, 4)
>>> split
array([[(slice(0, 256, None), slice(0, 256, None)),
        (slice(0, 256, None), slice(256, 512, None)),
        (slice(0, 256, None), slice(512, 768, None)),
        (slice(0, 256, None), slice(768, 1024, None))],
       [(slice(256, 512, None), slice(0, 256, None)),
        (slice(256, 512, None), slice(256, 512, None)),
        (slice(256, 512, None), slice(512, 768, None)),
        (slice(256, 512, None), slice(768, 1024, None))]],
      dtype=[('0', 'O'), ('1', 'O')])
```

2.8.2 Sub-tile shape constraint

The split can also be influenced by specifying the *sub_tile_shape* parameter which forces the tile shape to be an even multiple of the *sub_tile_shape*:

```
>>> split = shape_split(
...   array_shape=[512, 1024],
...   array_itemsize=4,
...   max_tile_bytes=512*512,
...   max_tile_shape=[numpy.inf, 256],
...   sub_tile_shape=(15, 10)
... )
...
>>> split.shape
(3, 5)
>>> split
array([[(slice(0, 180, None), slice(0, 210, None)),
        (slice(0, 180, None), slice(210, 420, None)),
        (slice(0, 180, None), slice(420, 630, None)),
        (slice(0, 180, None), slice(630, 840, None)),
        (slice(0, 180, None), slice(840, 1024, None))],
       [(slice(180, 360, None), slice(0, 210, None)),
        (slice(180, 360, None), slice(210, 420, None)),
        (slice(180, 360, None), slice(420, 630, None)),
        (slice(180, 360, None), slice(630, 840, None)),
        (slice(180, 360, None), slice(840, 1024, None))],
       [(slice(360, 512, None), slice(0, 210, None)),
        (slice(360, 512, None), slice(210, 420, None)),
        (slice(360, 512, None), slice(420, 630, None)),
        (slice(360, 512, None), slice(630, 840, None)),
        (slice(360, 512, None), slice(840, 1024, None))]],
      dtype=[('0', 'O'), ('1', 'O')])
```

2.9 The *array_start* parameter

The *array_start* argument to the *array_split.shape_split()* function and the *array_split.ShapeSplitter.__init__()* constructor specifies an index offset for the slices in the returned `tuple` of `slice` objects:

```
>>> split = shape_split((15,), 3)
>>> split
array([(slice(0, 5, None),), (slice(5, 10, None),), (slice(10, 15, None),)],
      dtype=[('0', 'O')])
>>> split = shape_split((15,), 3, array_start=(20,))
>>> split
array([(slice(20, 25, None),), (slice(25, 30, None),),
       (slice(30, 35, None),)],
      dtype=[('0', 'O')])
```

2.10 The *halo* parameter

The *halo* parameter can be used to generate tiles which overlap with neighbouring tiles by a specified number of array elements (in each axis direction):

```
>>> from array_split import ARRAY_BOUNDS, NO_BOUNDS
>>> split = shape_split([16,], 4) # No halo
>>> split.shape
(4,)
```

```
>>> split
array([(slice(0, 4, None),), (slice(4, 8, None),), (slice(8, 12, None),),
       (slice(12, 16, None),)],
      dtype=[('0', 'O')])
>>> split = shape_split([16,], 4, halo=2, tile_bounds_policy=ARRAY_BOUNDS) # halo width = 2
>>> split.shape
(4,)
>>> split
array([(slice(0, 6, None),), (slice(2, 10, None),), (slice(6, 14, None),),
       (slice(10, 16, None),)],
      dtype=[('0', 'O')])
>>> split = shape_split(
... [16,],
... 4,
... halo=2,
... tile_bounds_policy=NO_BOUNDS  # halo width = 2 and tile halos extend outside array_shape bounds
... )
>>> split.shape
(4,)
>>> split
array([(slice(-2, 6, None),), (slice(2, 10, None),), (slice(6, 14, None),),
       (slice(10, 18, None),)],
      dtype=[('0', 'O')])
```

The `tile_bounds_policy` parameter specifies whether the *`halo`* extended tiles can extend beyond the bounding box defined by the *start* index *`array_start`* and the *stop* index *`array_start + array_shape`*.

Asymmetric halo extensions can also be specified:

```
>>> split = shape_split(
... [16,],
... 4,
... halo=((1,2),),
... tile_bounds_policy=NO_BOUNDS
... )
>>> split.shape
(4,)
>>> split
array([(slice(-1, 6, None),), (slice(3, 10, None),), (slice(7, 14, None),),
       (slice(11, 18, None),)],
      dtype=[('0', 'O')])
```

For an `N` dimensional split (i.e. `N = len(array_shape)`), the *`halo`* parameter can be either a

scalar Tiles are extended by *`halo`* voxels in the negative and positive directions for all axes.

1D sequence Tiles are extended by *`halo[i]`* voxels in the negative and positive directions for axis `i`.

2D sequence Tiles are extended by *`halo[i][0]`* voxels in the negative direction and *`halo[i][1]`* in the positive direction for axis `i`.

For example, in 3D:

```
>>> split = shape_split(
... [16, 8, 8],
... 2,
... halo=1,  # halo=1 in +ve and -ve directions for all axes
... tile_bounds_policy=NO_BOUNDS
... )
>>> split.shape
```

```
(2, 1, 1)
>>> split
array([[[(slice(-1, 9, None), slice(-1, 9, None), slice(-1, 9, None))]],

       [[(slice(7, 17, None), slice(-1, 9, None), slice(-1, 9, None))]]],
      dtype=[('0', 'O'), ('1', 'O'), ('2', 'O')])
>>> split = shape_split(
... [16, 8, 8],
... 2,
... halo=(1, 2, 3),  # halo=1 for axis 0, halo=2 for axis 1, halo=3 for axis=2
... tile_bounds_policy=NO_BOUNDS
... )
>>> split.shape
(2, 1, 1)
>>> split
array([[[(slice(-1, 9, None), slice(-2, 10, None), slice(-3, 11, None))]],

       [[(slice(7, 17, None), slice(-2, 10, None), slice(-3, 11, None))]]],
      dtype=[('0', 'O'), ('1', 'O'), ('2', 'O')])
>>> split = shape_split(
... [16, 8, 8],
... 2,
... halo=((1, 2), (3, 4), (5, 6)),  # halo=1 for -ve axis 0, halo=2 for +ve axis 0
...                                 # halo=3 for -ve axis 1, halo=4 for +ve axis 1
...                                 # halo=5 for -ve axis 2, halo=6 for +ve axis 2
... tile_bounds_policy=NO_BOUNDS
... )
>>> split.shape
(2, 1, 1)
>>> split
array([[[(slice(-1, 10, None), slice(-3, 12, None), slice(-5, 14, None))]],

       [[(slice(7, 18, None), slice(-3, 12, None), slice(-5, 14, None))]]],
      dtype=[('0', 'O'), ('1', 'O'), ('2', 'O')])
```

CHAPTER 3

API Reference

3.1 The `array_split` Package

Small python package for splitting a `numpy.ndarray` (or just an array shape) into a number of sub-arrays.

The two main functions are:

array_split.array_split() Similar to `numpy.array_split()`, returns a list of *views* of sub-arrays of the input `numpy.ndarray`. Can split along multiple axes and has more splitting criteria (parameters) than `numpy.array_split()`.

array_split.shape_split() Instead taking an `numpy.ndarray` as an argument, it takes the array *shape* and returns tuples of `slice` objects which indicate the extents of the sub-arrays.

These two functions use an instance of the *`array_split.ShapeSplitter`* class which contains the bulk of the *split* implementation and maintains some state related to the computed split.

Splitting of multi-dimensional arrays can be performed according to several criteria:

- Per-axis indicies indicating the *cut* positions.
- Per-axis number of sub-arrays.
- Total number of sub-arrays (with optional per-axis *number of sections* constraints).
- Specific sub-array shape.
- Maximum number of bytes for a sub-array with constraints:
  - sub-arrays are an even multiple of a specified sub-tile shape
  - upper limit on the per-axis sub-array shape

The usage documentation is given in the *array_split Examples* section.

3.1.1 Classes and Functions

`shape_split`(array_shape, *args, **kwargs)	Splits specified *`array_shape`* in tiles, returns array of `slice` tuples.
`array_split`(ary[, indices_or_sections, ...])	Splits the specified array *`ary`* into sub-arrays, returns list of `numpy.ndarray`.
`ShapeSplitter`(array_shape[, ...])	Implements array shape splitting.

array_split.shape_split

array_split.**shape_split**(*array_shape*, **args*, ***kwargs*)

Splits specified *array_shape* in tiles, returns array of slice tuples.

Parameters

- **array_shape** (sequence of int) – The shape to be *split*.
- **indices_or_sections** (None, int or sequence of int) – If an integer, indicates the number of elements in the calculated *split* array. If a sequence, indicates the indicies (per axis) at which the splits occur. See *Splitting by number of tiles* examples.
- **axis** (None, int or sequence of int) – If an integer, indicates the axis which is to be split. If a sequence integers, indicates the number of slices per axis, i.e. if *axis* = [3, 5] then axis 0 is cut into 3 slices and axis 1 is cut into 5 slices for a total of 15 (3*5) rectangular slices in the returned (3, 5) shaped split. See *Splitting by number of tiles* examples and *Splitting by per-axis cut indices* examples.
- **array_start** (None or sequence of int) – The start index. Defaults to [0,]*len(array_shape). The array indexing extents are assumed to range from *array_start* to *array_start + array_shape*. See *The array_start parameter* examples.
- **array_itemsize** (int or sequence of int) – Number of bytes per array element. Only relevant when *max_tile_bytes* is specified. See *Splitting by maximum bytes per tile* examples.
- **tile_shape** (None or sequence of int) – When not None, specifies explicit shape for tiles. Should be same length as *array_shape*. See *Splitting by tile shape* examples.
- **max_tile_bytes** (None or int) – The maximum number of bytes for calculated tile_shape. See *Splitting by maximum bytes per tile* examples.
- **max_tile_shape** (None or sequence of int) – Per axis maximum shapes for the calculated tile_shape. Only relevant when *max_tile_bytes* is specified. Should be same length as *array_shape*. See *Splitting by maximum bytes per tile* examples.
- **sub_tile_shape** (None or sequence of int) – When not None, the calculated tile_shape will be an even multiple of this sub-tile shape. Only relevant when *max_tile_bytes* is specified. Should be same length as *array_shape*. See *Splitting by maximum bytes per tile* examples.
- **halo** (None, int, sequence of int, or (len(*array_shape*), 2) shaped numpy.ndarray) – How tiles are extended per axis in -ve and +ve directions with *halo* elements. See *The halo parameter* examples.
- **tile_bounds_policy** (str) – Specifies whether tiles can extend beyond the array boundaries. Only relevant for halo values greater than one. If *tile_bounds_policy* is *ARRAY_BOUNDS* then the calculated tiles will not extend beyond the array extents *array_start* and *array_start + array_shape*. If *tile_bounds_policy* is *NO_BOUNDS* then the returned tiles will extend beyond the *array_start* and *array_start + array_shape* extend for positive *halo* values. See *The halo parameter* examples.

Return type numpy.ndarray

Returns Array of tuple objects. Each tuple element is a slice object so that each tuple defines a multi-dimensional slice of an array of shape *array_shape*.

See also:

array_split.array_split(), *array_split.ShapeSplitter()*, *array_split Examples*

array_split.array_split

array_split.**array_split**(*ary*, *indices_or_sections=None*, *axis=None*, *tile_shape=None*, *max_tile_bytes=None*, *max_tile_shape=None*, *sub_tile_shape=None*, *halo=None*)

Splits the specified array *ary* into sub-arrays, returns list of numpy.ndarray.

Parameters

- **ary** (numpy.ndarray) – Array which is split into sub-arrays.
- **indices_or_sections** (None, int or sequence of int) – If an integer, indicates the number of elements in the calculated *split* array. If a sequence, indicates the indicies (per axis) at which the splits occur. See *Splitting by number of tiles* examples.
- **axis** (None, int or sequence of int) – If an integer, indicates the axis which is to be split. If a sequence integers, indicates the number of slices per axis, i.e. if *axis = [3, 5]* then axis 0 is cut into 3 slices and axis 1 is cut into 5 slices for a total of 15 (3*5) rectangular slices in the returned (3, 5) shaped split. See *Splitting by number of tiles* examples and *Splitting by per-axis cut indices* examples.
- **tile_shape** (None or sequence of int) – When not None, specifies explicit shape for tiles. Should be same length as *array_shape*. See *Splitting by tile shape* examples.
- **max_tile_bytes** (None or int) – The maximum number of bytes for calculated tile_shape. See *Splitting by maximum bytes per tile* examples.
- **max_tile_shape** (None or sequence of int) – Per axis maximum shapes for the calculated tile_shape. Only relevant when *max_tile_bytes* is specified. Should be same length as *array_shape*. See *Splitting by maximum bytes per tile* examples.
- **sub_tile_shape** (None or sequence of int) – When not None, the calculated tile_shape will be an even multiple of this sub-tile shape. Only relevant when *max_tile_bytes* is specified. Should be same length as *array_shape*. See *Splitting by maximum bytes per tile* examples.
- **halo** (None, int, sequence of int, or (len(*ary*.shape), 2) shaped numpy.ndarray) – How tiles are extended per axis in -ve and +ve directions with *halo* elements. See *The halo parameter* examples.

Return type list

Returns List of numpy.ndarray elements, where each element is a *slice* from *ary* (potentially an empty slice).

See also:

array_split.shape_split(), *array_split.ShapeSplitter()*, *array_split Examples*

array_split.ShapeSplitter

class array_split.**ShapeSplitter**(*array_shape*, *indices_or_sections=None*, *axis=None*, *array_start=None*, *array_itemsize=1*, *tile_shape=None*, *max_tile_bytes=None*, *max_tile_shape=None*, *sub_tile_shape=None*, *halo=None*, *tile_bounds_policy=<property object>*)

Implements array shape splitting. There are three main (top-level) methods:

__init__() Initialisation of parameters which define the split.

set_split_extents() Calculates the per-axis indices for the cuts. Sets the *split_shape*, *split_begs* and *split_ends* attributes.

calculate_split() Calls *set_split_extents()* followed by *calculate_split_from_extents()* to return the numpy.ndarray of tuple elements (slices).

Methods

__init__(array_shape[, indices_or_sections, ...])	Initialises parameters which define a split.
calculate_axis_split_extents(num_sections, size)	Divides `range(0, size)` into (approximately) equal sized interv
calculate_split()	Computes the split.
calculate_split_by_indices_per_axis()	Returns split calculated using extents obtained from *indices_per*
calculate_split_by_split_size()	Returns split calculated using extents obtained from *split_size*
calculate_split_by_tile_max_bytes()	Returns split calculated using extents obtained from *max_tile_by*
calculate_split_by_tile_shape()	Returns split calculated using extents obtained from *tile_shape*.
calculate_split_from_extents()	Returns split calculated using extents obtained from *split_begs*
check_consistent_parameter_dimensions()	Ensure that all parameter dimensions are consistent with the *array*
check_consistent_parameter_grouping()	Ensures this object does not have conflicting groups of parameters.
check_halo()	Raises `ValueError` if there is an inconsistency between shapes of
check_split_parameters()	Ensures this object has a state consistent with evaluating a split.
check_tile_bounds_policy()	Raises `ValueError` if *tile_bounds_policy* is not in `[self`
set_split_extents()	Sets split extents (*split_begs* and *split_ends*) calculated usi
set_split_extents_by_indices_per_axis()	Sets split shape *split_shape* and split extents (*split_begs* an
set_split_extents_by_split_size()	Sets split shape *split_shape* and split extents (*split_begs* an
set_split_extents_by_tile_max_bytes()	Sets split extents (*split_begs* and *split_ends*) calculated usi
set_split_extents_by_tile_shape()	Sets split shape *split_shape* and split extents (*split_begs* an
update_tile_extent_bounds()	Updates the *tile_beg_min* and *tile_end_max* data members

array_split.ShapeSplitter.__init__

ShapeSplitter.**__init__**(*array_shape*, *indices_or_sections=None*, *axis=None*, *array_start=None*, *array_itemsize=1*, *tile_shape=None*, *max_tile_bytes=None*, *max_tile_shape=None*, *sub_tile_shape=None*, *halo=None*, *tile_bounds_policy=<property object>*)

Initialises parameters which define a split.

Parameters

- **array_shape** (sequence of `int`) – The shape to be *split*.
- **indices_or_sections** (`None`, `int` or sequence of `int`) – If an integer, indicates the number of elements in the calculated *split* array. If a sequence, indicates the indicies (per axis) at which the splits occur. See *Splitting by number of tiles* examples.
- **axis** (`None`, `int` or sequence of `int`) – If an integer, indicates the axis which is to be split. If a sequence integers, indicates the number of slices per axis, i.e. if `axis = [3, 5]` then axis `0` is cut into 3 slices and axis `1` is cut into 5 slices for a total of 15 (`3*5`) rectangular slices in the returned `(3, 5)` shaped split. See *Splitting by number of tiles* examples and *Splitting by per-axis cut indices* examples.
- **array_start** (`None` or sequence of `int`) – The start index. Defaults to `[0,]*len(array_shape)`. The array indexing extents are assumed to range from

array_start to array_start + array_shape. See *The array_start parameter* examples.

- **array_itemsize** (int or sequence of int) – Number of bytes per array element. Only relevant when *max_tile_bytes* is specified. See *Splitting by maximum bytes per tile* examples.
- **tile_shape** (None or sequence of int) – When not None, specifies explicit shape for tiles. Should be same length as *array_shape*. See *Splitting by tile shape* examples.
- **max_tile_bytes** (None or int) – The maximum number of bytes for calculated tile_shape. See *Splitting by maximum bytes per tile* examples.
- **max_tile_shape** (None or sequence of int) – Per axis maximum shapes for the calculated tile_shape. Only relevant when *max_tile_bytes* is specified. Should be same length as *array_shape*. See *Splitting by maximum bytes per tile* examples.
- **sub_tile_shape** (None or sequence of int) – When not None, the calculated tile_shape will be an even multiple of this sub-tile shape. Only relevant when *max_tile_bytes* is specified. Should be same length as *array_shape*. See *Splitting by maximum bytes per tile* examples.
- **halo** (None, int, sequence of int, or (len(*array_shape*), 2) shaped numpy.ndarray) – How tiles are extended per axis in -ve and +ve directions with *halo* elements. See *The halo parameter* examples.
- **tile_bounds_policy** (str) – Specifies whether tiles can extend beyond the array boundaries. Only relevant for halo values greater than one. If *tile_bounds_policy* is *ARRAY_BOUNDS* then the calculated tiles will not extend beyond the array extents *array_start* and *array_start + array_shape*. If *tile_bounds_policy* is *NO_BOUNDS* then the returned tiles will extend beyond the *array_start* and *array_start + array_shape* extend for positive *halo* values. See *The halo parameter* examples.

See also:

array_split Examples

array_split.ShapeSplitter.calculate_axis_split_extents

ShapeSplitter.**calculate_axis_split_extents**(*num_sections*, *size*)

Divides range(0, *size*) into (approximately) equal sized intervals. Returns (begs, ends) where slice(begs[i], ends[i]) define the intervals for i in range(0, *num_sections*).

Parameters

- **num_sections** (int) – Divide range(0, *size*) into this many intervals (approximately) equal sized intervals.
- **size** (int) – Range for the subdivision.

Return type tuple

Returns Two element tuple (begs, ends) such that slice(begs[i], ends[i]) define the intervals for i in range(0, *num_sections*).

array_split.ShapeSplitter.calculate_split

ShapeSplitter.**calculate_split**()
Computes the split.

Return type numpy.ndarray

Returns A numpy structured array of dimension len(*self*.array_shape). Each element of the returned array is a tuple containing len(*self*.array_shape) elements, with each element being a slice object. Each tuple defines a slice within the bounds *self*.array_start - *self*.halo[:, 0] to *self*.array_start + *self*.array_shape + *self*.halo[:, 1].

array_split.ShapeSplitter.calculate_split_by_indices_per_axis

ShapeSplitter.**calculate_split_by_indices_per_axis**()
Returns split calculated using extents obtained from *indices_per_axis*.

Return type numpy.ndarray

Returns A numpy structured array where each element is a tuple of slice objects.

array_split.ShapeSplitter.calculate_split_by_split_size

ShapeSplitter.**calculate_split_by_split_size**()
Returns split calculated using extents obtained from *split_size* and *split_num_slices_per_axis*.

Return type numpy.ndarray

Returns A numpy structured array where each element is a tuple of slice objects.

array_split.ShapeSplitter.calculate_split_by_tile_max_bytes

ShapeSplitter.**calculate_split_by_tile_max_bytes**()
Returns split calculated using extents obtained from *max_tile_bytes* (and *max_tile_shape*, *sub_tile_shape*, *halo*).

Return type numpy.ndarray

Returns A numpy structured array where each element is a tuple of slice objects.

array_split.ShapeSplitter.calculate_split_by_tile_shape

ShapeSplitter.**calculate_split_by_tile_shape**()
Returns split calculated using extents obtained from *tile_shape*.

Return type numpy.ndarray

Returns A numpy structured array where each element is a tuple of slice objects.

array_split.ShapeSplitter.calculate_split_from_extents

ShapeSplitter.**calculate_split_from_extents**()
Returns split calculated using extents obtained from *split_begs* and *split_ends*.

Return type numpy.ndarray

Returns A numpy structured array where each element is a tuple of slice objects.

array_split.ShapeSplitter.check_consistent_parameter_dimensions

ShapeSplitter.**check_consistent_parameter_dimensions**()
Ensure that all parameter dimensions are consistent with the *array_shape* dimension.

Raises **ValueError** – For inconsistent parameter dimensions.

array_split.ShapeSplitter.check_consistent_parameter_grouping

ShapeSplitter.**check_consistent_parameter_grouping**()
Ensures this object does not have conflicting groups of parameters.

Raises **ValueError** – For conflicting or absent parameters.

array_split.ShapeSplitter.check_halo

ShapeSplitter.**check_halo**()
Raises ValueError if there is an inconsistency between shapes of *array_shape* and *halo*.

array_split.ShapeSplitter.check_split_parameters

ShapeSplitter.**check_split_parameters**()
Ensures this object has a state consistent with evaluating a split.

Raises **ValueError** – For conflicting or absent parameters.

array_split.ShapeSplitter.check_tile_bounds_policy

ShapeSplitter.**check_tile_bounds_policy**()
Raises ValueError if *tile_bounds_policy* is not in [*self*.ARRAY_BOUNDS, *self*.NO_BOUNDS].

array_split.ShapeSplitter.set_split_extents

ShapeSplitter.**set_split_extents**()
Sets split extents (*split_begs* and *split_ends*) calculated using selected attributes set from *__init__()*.

array_split.ShapeSplitter.set_split_extents_by_indices_per_axis

ShapeSplitter.**set_split_extents_by_indices_per_axis**()
Sets split shape *split_shape* and split extents (*split_begs* and *split_ends*) from values in *indices_per_axis*.

array_split.ShapeSplitter.set_split_extents_by_split_size

ShapeSplitter.**set_split_extents_by_split_size**()
Sets split shape *split_shape* and split extents (*split_begs* and *split_ends*) from values in *split_size* and *split_num_slices_per_axis*.

array_split.ShapeSplitter.set_split_extents_by_tile_max_bytes

ShapeSplitter.**set_split_extents_by_tile_max_bytes**()
Sets split extents (*split_begs* and *split_ends*) calculated using from *max_tile_bytes* (and *max_tile_shape*, *sub_tile_shape*, *halo*).

array_split.ShapeSplitter.set_split_extents_by_tile_shape

ShapeSplitter.**set_split_extents_by_tile_shape**()
Sets split shape *split_shape* and split extents (*split_begs* and *split_ends*) from value of *tile_shape*.

array_split.ShapeSplitter.update_tile_extent_bounds

ShapeSplitter.**update_tile_extent_bounds**()
Updates the *tile_beg_min* and *tile_end_max* data members according to *tile_bounds_policy*.

Attributes

array_itemsize	The number of bytes per array element, see *max_tile_bytes*.
array_shape	The shape of the array which is to be split.
array_start	The start index.
halo	Per-axis -ve and +ve halo sizes for extending tiles to overlap with neighbouring tiles.
indices_per_axis	The per-axis indices indicating the cuts for the split.
logger	Class attribute for `logging.Logger` logging.
max_tile_bytes	The maximum number of bytes for any tile (including *halo*) in the returned split.
max_tile_shape	Per-axis maximum sizes for calculated tiles.
split_begs	The list of per-axis start indices for `slice` objects.
split_ends	The list of per-axis stop indices for `slice` objects.
split_num_slices_per_axis	Number of slices per axis.
split_shape	The shape of the calculated split array.
split_size	An `int` indicating the number of tiles in the calculated split.
sub_tile_shape	Calculated tile shape will be an integer multiple of this sub-tile shape.
tile_beg_min	The per-axis minimum index for `slice.start`.

Continued on next page

Table 3.3 – continued from previous page

tile_bounds_policy	A string indicating whether tile halo extents can extend beyond the array domain.
tile_end_max	The per-axis maximum index for `slice.stop`.
tile_shape	The shape of all tiles in the calculated split.
valid_tile_bounds_policies	Class attribute indicating list of valid values for `tile_bound_policy`.

array_split.ShapeSplitter.array_itemsize

ShapeSplitter.**array_itemsize**
The number of bytes per array element, see *max_tile_bytes*.

array_split.ShapeSplitter.array_shape

ShapeSplitter.**array_shape**
The shape of the array which is to be split. A sequence of `int` indicating the per-axis sizes which are to be split.

array_split.ShapeSplitter.array_start

ShapeSplitter.**array_start**
The start index. A sequence of `int` indicating the start of indexing for the tile slices. Defaults to `numpy.zeros_like(`*`self`*`.array_shape)`.

array_split.ShapeSplitter.halo

ShapeSplitter.**halo**
Per-axis -ve and +ve halo sizes for extending tiles to overlap with neighbouring tiles. A `(N, 2)` shaped array indicating the

array_split.ShapeSplitter.indices_per_axis

ShapeSplitter.**indices_per_axis**
The per-axis indices indicating the cuts for the split. A `list` of 1D `numpy.ndarray` objects such that *`self`*`.indices_per_axis[i]` indicates the cut positions for axis `i`.

array_split.ShapeSplitter.logger

ShapeSplitter.**logger = <logging.Logger object>**
Class attribute for `logging.Logger` logging.

array_split.ShapeSplitter.max_tile_bytes

ShapeSplitter.**max_tile_bytes**
The maximum number of bytes for any tile (including *halo*) in the returned split. An `int` which constrains the tile shape such that any tile from the computed split is no bigger than *max_tile_bytes*.

array_split.ShapeSplitter.max_tile_shape

ShapeSplitter.**max_tile_shape**
Per-axis maximum sizes for calculated tiles. A 1D `numpy.ndarray` of `int` indicating the per-axis maximum number of elements for tiles in the calculated split.

array_split.ShapeSplitter.split_begs

ShapeSplitter.**split_begs**
The list of per-axis start indices for `slice` objects. A `list` of 1D `numpy.ndarray` objects indicating the `slice.start` index for for tiles.

array_split.ShapeSplitter.split_ends

ShapeSplitter.**split_ends**
The list of per-axis stop indices for `slice` objects. A `list` of 1D `numpy.ndarray` objects indicating the `slice.stop` index for for tiles.

array_split.ShapeSplitter.split_num_slices_per_axis

ShapeSplitter.**split_num_slices_per_axis**
Number of slices per axis. A 1D `numpy.ndarray` of `int` indicating the number of slices (sections) per axis, so that *`self`*`.split_num_slices_per_axis[i]` is an integer indicating the number of sections along axis `i` in the calculated split.

array_split.ShapeSplitter.split_shape

ShapeSplitter.**split_shape**
The shape of the calculated split array. Indicates the per-axis number of sections in the calculated split. A 1D `numpy.ndarray`.

array_split.ShapeSplitter.split_size

ShapeSplitter.**split_size**
An `int` indicating the number of tiles in the calculated split.

array_split.ShapeSplitter.sub_tile_shape

ShapeSplitter.**sub_tile_shape**
Calculated tile shape will be an integer multiple of this sub-tile shape. i.e. `(self.tile_shape[i] % self.sub_tile_shape[i]) == 0`, for `i in range(0, len(self.tile_shape))`. A 1D `numpy.ndarray` of `int` indicating sub-tile shape.

array_split.ShapeSplitter.tile_beg_min

ShapeSplitter.**tile_beg_min**
The per-axis minimum index for `slice.start`. The per-axis lower bound for tile start indices. A 1D `numpy.ndarray`.

array_split.ShapeSplitter.tile_bounds_policy

ShapeSplitter.**tile_bounds_policy**
A string indicating whether tile halo extents can extend beyond the array domain. Valid values are indicated by *valid_tile_bounds_policies*.

array_split.ShapeSplitter.tile_end_max

ShapeSplitter.**tile_end_max**
The per-axis maximum index for `slice.stop`. The per-axis upper bound for tile stop indices. A 1D `numpy.ndarray`.

array_split.ShapeSplitter.tile_shape

ShapeSplitter.**tile_shape**
The shape of all tiles in the calculated split. A 1D `numpy.ndarray` of `int` indicating the per-axis number of elements for tiles in the calculated split.

array_split.ShapeSplitter.valid_tile_bounds_policies

ShapeSplitter.**valid_tile_bounds_policies = [<property object at 0x7f90c41ab0e8>, <property object at 0x7**
Class attribute indicating list of valid values for `tile_bound_policy`. See *ARRAY_BOUNDS* and *NO_BOUNDS*.

3.1.2 Attributes

array_split.**ARRAY_BOUNDS = <property object>**
See *array_split.split.ARRAY_BOUNDS*

array_split.**NO_BOUNDS = <property object>**
See *array_split.split.NO_BOUNDS*

3.2 The `array_split.split` Module

Defines array splitting functions and classes.

3.2.1 Classes and Functions

shape_factors(n[, dim])	Returns a `numpy.ndarray` of factors `f` such that `(len(f) == di`
calculate_num_slices_per_axis(...[, ...])	Returns a `numpy.ndarray` (`return_array` say) where non-positiv
calculate_tile_shape_for_max_bytes(...[, ...])	Returns a tile shape `tile_shape` such that `numpy.product(tile`
ShapeSplitter(array_shape[, ...])	Implements array shape splitting.
shape_split(array_shape, *args, **kwargs)	Splits specified *array_shape* in tiles, returns array of `slice` tuples.
array_split(ary[, indices_or_sections, ...])	Splits the specified array *ary* into sub-arrays, returns list of `numpy.n`

array_split.split.shape_factors

array_split.split.**shape_factors**(*n*, *dim=2*)

Returns a numpy.ndarray of factors f such that (len(f) == *dim*) and (numpy.product(f) == *n*). The returned factors are as *square* (*cubic*, etc) as possible. For example:

```
>>> shape_factors(24, 1)
array([24])
>>> shape_factors(24, 2)
array([4, 6])
>>> shape_factors(24, 3)
array([2, 3, 4])
>>> shape_factors(24, 4)
array([2, 2, 2, 3])
>>> shape_factors(24, 5)
array([1, 2, 2, 2, 3])
>>> shape_factors(24, 6)
array([1, 1, 2, 2, 2, 3])
```

Parameters

- **n** (int) – Integer which is factored into *dim* factors.
- **dim** (int) – Number of factors.

Return type numpy.ndarray

Returns A (*dim*,) shaped array of integers which are factors of *n*.

array_split.split.calculate_num_slices_per_axis

array_split.split.**calculate_num_slices_per_axis**(*num_slices_per_axis*, *num_slices*, *max_slices_per_axis=None*)

Returns a numpy.ndarray (return_array say) where non-positive elements of the *num_slices_per_axis* sequence have been replaced with positive integer values such that numpy.product(return_array) == num_slices and:

```
numpy.all(
    return_array[numpy.where(num_slices_per_axis <= 0)]
    <=
    max_slices_per_axis[numpy.where(num_slices_per_axis <= 0)]
) is True
```

Parameters

- **num_slices_per_axis** (sequence of int) – Constraint for per-axis sub-divisions. Non-positive elements indicate values to be replaced in the returned array. Positive values are identical to the corresponding element in the returned array.
- **num_slices** (*integer*) – Indicates the number of slices (rectangular sub-arrays) formed by performing sub-divisions per axis. The returned array return_array has elements assigned such that numpy.product(return_array) == *num_slices*.
- **max_slices_per_axis** (sequence of int (or None)) – Constraint specifying maximum number of per-axis sub-divisions. If None defaults to numpy.array([numpy.inf,]*len(*num_slices_per_axis*)).

Return type numpy.ndarray

Returns An array `return_array` such that `numpy.product(return_array) == num_slices`.

Examples:

```
>>> from array_split.split import calculate_num_slices_per_axis
>>>
>>> calculate_num_slices_per_axis([0, 0, 0], 16)
array([4, 2, 2])
>>> calculate_num_slices_per_axis([1, 0, 0], 16)
array([1, 4, 4])
>>> calculate_num_slices_per_axis([1, 0, 0], 16, [2, 2, 16])
array([1, 2, 8])
```

array_split.split.calculate_tile_shape_for_max_bytes

array_split.split.**calculate_tile_shape_for_max_bytes**(*array_shape*, *array_itemsize*, *max_tile_bytes*, *max_tile_shape=None*, *sub_tile_shape=None*, *halo=None*)

Returns a tile shape `tile_shape` such that `numpy.product(tile_shape)*numpy.sum(array_itemsize) <= max_tile_bytes`. Also, if `max_tile_shape is not None` then `numpy.all(tile_shape <= max_tile_shape) is True` and if `sub_tile_shape is not None` the `numpy.all((tile_shape % sub_tile_shape) == 0) is True`.

Parameters

- **array_shape** (sequence of `int`) – Shape of the array which is to be split into tiles.
- **array_itemsize** (`int`) – The number of bytes per element of the array to be tiled.
- **max_tile_bytes** (`int`) – The maximum number of bytes for the returned `tile_shape`.
- **max_tile_shape** (sequence of `int`) – Per axis maximum shapes for the returned `tile_shape`.
- **sub_tile_shape** (sequence of `int`) – The returned `tile_shape` will be an even multiple of this sub-tile shape.
- **halo** (`int`, sequence of `int`, or `(len(array_shape), 2)` shaped `numpy.ndarray`) – How tiles are extended in each axis direction with *halo* elements. See *The halo parameter* for meaning of `halo` values.

Return type `numpy.ndarray`

Returns A 1D array of shape `(len(array_shape),)` indicating a *tile shape* which will (approximately) uniformly divide the given `array_shape` into tiles (sub-arrays).

Examples:

```
>>> from array_split.split import calculate_tile_shape_for_max_bytes
>>> calculate_tile_shape_for_max_bytes(
... array_shape=[512,],
... array_itemsize=1,
... max_tile_bytes=512
... )
array([512])
>>> calculate_tile_shape_for_max_bytes(
```

```
    ... array_shape=[512,],
    ... array_itemsize=2,  # Doubling the itemsize halves the tile size.
    ... max_tile_bytes=512
    ... )
    array([256])
    >>> calculate_tile_shape_for_max_bytes(
    ... array_shape=[512,],
    ... array_itemsize=1,
    ... max_tile_bytes=512-1  # tile shape will now be halved
    ... )
    array([256])
```

array_split.split.ShapeSplitter

class array_split.split.**ShapeSplitter**(*array_shape*, *indices_or_sections=None*, *axis=None*, *array_start=None*, *array_itemsize=1*, *tile_shape=None*, *max_tile_bytes=None*, *max_tile_shape=None*, *sub_tile_shape=None*, *halo=None*, *tile_bounds_policy=<property object>*)

Implements array shape splitting. There are three main (top-level) methods:

__init__() Initialisation of parameters which define the split.

set_split_extents() Calculates the per-axis indices for the cuts. Sets the *split_shape*, *split_begs* and *split_ends* attributes.

calculate_split() Calls *set_split_extents()* followed by *calculate_split_from_extents()* to return the numpy.ndarray of tuple elements (slices).

Methods

__init__(array_shape[, indices_or_sections, ...])	Initialises parameters which define a split.
calculate_axis_split_extents(num_sections, size)	Divides range(0, *size*) into (approximately) equal sized interv
calculate_split()	Computes the split.
calculate_split_by_indices_per_axis()	Returns split calculated using extents obtained from *indices_per*
calculate_split_by_split_size()	Returns split calculated using extents obtained from *split_size*
calculate_split_by_tile_max_bytes()	Returns split calculated using extents obtained from *max_tile_by*
calculate_split_by_tile_shape()	Returns split calculated using extents obtained from *tile_shape*.
calculate_split_from_extents()	Returns split calculated using extents obtained from *split_begs*
check_consistent_parameter_dimensions()	Ensure that all parameter dimensions are consistent with the *array*
check_consistent_parameter_grouping()	Ensures this object does not have conflicting groups of parameters.
check_halo()	Raises ValueError if there is an inconsistency between shapes of
check_split_parameters()	Ensures this object has a state consistent with evaluating a split.
check_tile_bounds_policy()	Raises ValueError if *tile_bounds_policy* is not in [*self*
set_split_extents()	Sets split extents (*split_begs* and *split_ends*) calculated usi
set_split_extents_by_indices_per_axis()	Sets split shape *split_shape* and split extents (*split_begs* an
set_split_extents_by_split_size()	Sets split shape *split_shape* and split extents (*split_begs* an
set_split_extents_by_tile_max_bytes()	Sets split extents (*split_begs* and *split_ends*) calculated usi
set_split_extents_by_tile_shape()	Sets split shape *split_shape* and split extents (*split_begs* an
update_tile_extent_bounds()	Updates the *tile_beg_min* and *tile_end_max* data members

array_split.split.ShapeSplitter.__init__

ShapeSplitter.**__init__**(*array_shape*, *indices_or_sections=None*, *axis=None*, *array_start=None*, *array_itemsize=1*, *tile_shape=None*, *max_tile_bytes=None*, *max_tile_shape=None*, *sub_tile_shape=None*, *halo=None*, *tile_bounds_policy=<property object>*)

Initialises parameters which define a split.

Parameters

- **array_shape** (sequence of `int`) – The shape to be *split*.
- **indices_or_sections** (`None`, `int` or sequence of `int`) – If an integer, indicates the number of elements in the calculated *split* array. If a sequence, indicates the indicies (per axis) at which the splits occur. See *Splitting by number of tiles* examples.
- **axis** (`None`, `int` or sequence of `int`) – If an integer, indicates the axis which is to be split. If a sequence integers, indicates the number of slices per axis, i.e. if `axis = [3, 5]` then axis `0` is cut into 3 slices and axis `1` is cut into 5 slices for a total of 15 (`3*5`) rectangular slices in the returned `(3, 5)` shaped split. See *Splitting by number of tiles* examples and *Splitting by per-axis cut indices* examples.
- **array_start** (`None` or sequence of `int`) – The start index. Defaults to `[0,]*len(array_shape)`. The array indexing extents are assumed to range from `array_start` to `array_start + array_shape`. See *The array_start parameter* examples.
- **array_itemsize** (int or sequence of `int`) – Number of bytes per array element. Only relevant when `max_tile_bytes` is specified. See *Splitting by maximum bytes per tile* examples.
- **tile_shape** (`None` or sequence of `int`) – When not `None`, specifies explicit shape for tiles. Should be same length as `array_shape`. See *Splitting by tile shape* examples.
- **max_tile_bytes** (`None` or `int`) – The maximum number of bytes for calculated `tile_shape`. See *Splitting by maximum bytes per tile* examples.
- **max_tile_shape** (`None` or sequence of `int`) – Per axis maximum shapes for the calculated `tile_shape`. Only relevant when `max_tile_bytes` is specified. Should be same length as `array_shape`. See *Splitting by maximum bytes per tile* examples.
- **sub_tile_shape** (`None` or sequence of `int`) – When not `None`, the calculated `tile_shape` will be an even multiple of this sub-tile shape. Only relevant when `max_tile_bytes` is specified. Should be same length as `array_shape`. See *Splitting by maximum bytes per tile* examples.
- **halo** (`None`, `int`, sequence of `int`, or `(len(array_shape), 2)` shaped `numpy.ndarray`) – How tiles are extended per axis in -ve and +ve directions with *halo* elements. See *The halo parameter* examples.
- **tile_bounds_policy** (`str`) – Specifies whether tiles can extend beyond the array boundaries. Only relevant for halo values greater than one. If `tile_bounds_policy` is `ARRAY_BOUNDS` then the calculated tiles will not extend beyond the array extents `array_start` and `array_start + array_shape`. If `tile_bounds_policy` is `NO_BOUNDS` then the returned tiles will extend beyond the `array_start` and `array_start + array_shape` extend for positive `halo` values. See *The halo parameter* examples.

See also:

array_split Examples

array_split.split.ShapeSplitter.calculate_axis_split_extents

ShapeSplitter.**calculate_axis_split_extents**(*num_sections*, *size*)

Divides `range(0, size)` into (approximately) equal sized intervals. Returns `(begs, ends)` where `slice(begs[i], ends[i])` define the intervals for `i in range(0, num_sections)`.

Parameters

- **num_sections** (`int`) – Divide `range(0, size)` into this many intervals (approximately) equal sized intervals.
- **size** (`int`) – Range for the subdivision.

Return type `tuple`

Returns Two element tuple `(begs, ends)` such that `slice(begs[i], ends[i])` define the intervals for `i in range(0, num_sections)`.

array_split.split.ShapeSplitter.calculate_split

ShapeSplitter.**calculate_split**()

Computes the split.

Return type `numpy.ndarray`

Returns A `numpy` structured array of dimension `len(self.array_shape)`. Each element of the returned array is a `tuple` containing `len(self.array_shape)` elements, with each element being a `slice` object. Each `tuple` defines a slice within the bounds `self.array_start - self.halo[:, 0]` to `self.array_start + self.array_shape + self.halo[:, 1]`.

array_split.split.ShapeSplitter.calculate_split_by_indices_per_axis

ShapeSplitter.**calculate_split_by_indices_per_axis**()

Returns split calculated using extents obtained from *indices_per_axis*.

Return type `numpy.ndarray`

Returns A `numpy` structured array where each element is a `tuple` of `slice` objects.

array_split.split.ShapeSplitter.calculate_split_by_split_size

ShapeSplitter.**calculate_split_by_split_size**()

Returns split calculated using extents obtained from *split_size* and *split_num_slices_per_axis*.

Return type `numpy.ndarray`

Returns A `numpy` structured array where each element is a `tuple` of `slice` objects.

array_split.split.ShapeSplitter.calculate_split_by_tile_max_bytes

ShapeSplitter.**calculate_split_by_tile_max_bytes**()

Returns split calculated using extents obtained from *max_tile_bytes* (and *max_tile_shape*, *sub_tile_shape*, *halo*).

Return type numpy.ndarray

Returns A numpy structured array where each element is a tuple of slice objects.

array_split.split.ShapeSplitter.calculate_split_by_tile_shape

ShapeSplitter.**calculate_split_by_tile_shape**()
Returns split calculated using extents obtained from *tile_shape*.

Return type numpy.ndarray

Returns A numpy structured array where each element is a tuple of slice objects.

array_split.split.ShapeSplitter.calculate_split_from_extents

ShapeSplitter.**calculate_split_from_extents**()
Returns split calculated using extents obtained from *split_begs* and *split_ends*.

Return type numpy.ndarray

Returns A numpy structured array where each element is a tuple of slice objects.

array_split.split.ShapeSplitter.check_consistent_parameter_dimensions

ShapeSplitter.**check_consistent_parameter_dimensions**()
Ensure that all parameter dimensions are consistent with the *array_shape* dimension.

Raises **ValueError** – For inconsistent parameter dimensions.

array_split.split.ShapeSplitter.check_consistent_parameter_grouping

ShapeSplitter.**check_consistent_parameter_grouping**()
Ensures this object does not have conflicting groups of parameters.

Raises **ValueError** – For conflicting or absent parameters.

array_split.split.ShapeSplitter.check_halo

ShapeSplitter.**check_halo**()
Raises ValueError if there is an inconsistency between shapes of *array_shape* and *halo*.

array_split.split.ShapeSplitter.check_split_parameters

ShapeSplitter.**check_split_parameters**()
Ensures this object has a state consistent with evaluating a split.

Raises **ValueError** – For conflicting or absent parameters.

array_split.split.ShapeSplitter.check_tile_bounds_policy

ShapeSplitter.**check_tile_bounds_policy**()
Raises ValueError if *tile_bounds_policy* is not in [*self*.ARRAY_BOUNDS, *self*.NO_BOUNDS].

array_split.split.ShapeSplitter.set_split_extents

ShapeSplitter.**set_split_extents**()
Sets split extents (*split_begs* and *split_ends*) calculated using selected attributes set from *__init__()*.

array_split.split.ShapeSplitter.set_split_extents_by_indices_per_axis

ShapeSplitter.**set_split_extents_by_indices_per_axis**()
Sets split shape *split_shape* and split extents (*split_begs* and *split_ends*) from values in *indices_per_axis*.

array_split.split.ShapeSplitter.set_split_extents_by_split_size

ShapeSplitter.**set_split_extents_by_split_size**()
Sets split shape *split_shape* and split extents (*split_begs* and *split_ends*) from values in *split_size* and *split_num_slices_per_axis*.

array_split.split.ShapeSplitter.set_split_extents_by_tile_max_bytes

ShapeSplitter.**set_split_extents_by_tile_max_bytes**()
Sets split extents (*split_begs* and *split_ends*) calculated using from *max_tile_bytes* (and *max_tile_shape*, *sub_tile_shape*, *halo*).

array_split.split.ShapeSplitter.set_split_extents_by_tile_shape

ShapeSplitter.**set_split_extents_by_tile_shape**()
Sets split shape *split_shape* and split extents (*split_begs* and *split_ends*) from value of *tile_shape*.

array_split.split.ShapeSplitter.update_tile_extent_bounds

ShapeSplitter.**update_tile_extent_bounds**()
Updates the *tile_beg_min* and *tile_end_max* data members according to *tile_bounds_policy*.

Attributes

array_itemsize	The number of bytes per array element, see *max_tile_bytes*.

Continued on next page

Table 3.6 – continued from previous page

array_shape	The shape of the array which is to be split.
array_start	The start index.
halo	Per-axis -ve and +ve halo sizes for extending tiles to overlap with neighbouring tiles.
indices_per_axis	The per-axis indices indicating the cuts for the split.
logger	Class attribute for `logging.Logger` logging.
max_tile_bytes	The maximum number of bytes for any tile (including *halo*) in the returned split.
max_tile_shape	Per-axis maximum sizes for calculated tiles.
split_begs	The list of per-axis start indices for `slice` objects.
split_ends	The list of per-axis stop indices for `slice` objects.
split_num_slices_per_axis	Number of slices per axis.
split_shape	The shape of the calculated split array.
split_size	An `int` indicating the number of tiles in the calculated split.
sub_tile_shape	Calculated tile shape will be an integer multiple of this sub-tile shape.
tile_beg_min	The per-axis minimum index for `slice.start`.
tile_bounds_policy	A string indicating whether tile halo extents can extend beyond the array domain.
tile_end_max	The per-axis maximum index for `slice.stop`.
tile_shape	The shape of all tiles in the calculated split.
valid_tile_bounds_policies	Class attribute indicating list of valid values for `tile_bound_policy`.

array_split.split.ShapeSplitter.array_itemsize

ShapeSplitter.**array_itemsize**
The number of bytes per array element, see *max_tile_bytes*.

array_split.split.ShapeSplitter.array_shape

ShapeSplitter.**array_shape**
The shape of the array which is to be split. A sequence of `int` indicating the per-axis sizes which are to be split.

array_split.split.ShapeSplitter.array_start

ShapeSplitter.**array_start**
The start index. A sequence of `int` indicating the start of indexing for the tile slices. Defaults to `numpy.zeros_like(`*self*`.array_shape)`.

array_split.split.ShapeSplitter.halo

ShapeSplitter.**halo**
Per-axis -ve and +ve halo sizes for extending tiles to overlap with neighbouring tiles. A `(N, 2)` shaped array indicating the

array_split.split.ShapeSplitter.indices_per_axis

ShapeSplitter.**indices_per_axis**
The per-axis indices indicating the cuts for the split. A `list` of 1D `numpy.ndarray` objects such that *self*`.indices_per_axis[i]` indicates the cut positions for axis `i`.

array_split.split.ShapeSplitter.logger

ShapeSplitter.**logger = <logging.Logger object>**
Class attribute for `logging.Logger` logging.

array_split.split.ShapeSplitter.max_tile_bytes

ShapeSplitter.**max_tile_bytes**
The maximum number of bytes for any tile (including *halo*) in the returned split. An `int` which constrains the tile shape such that any tile from the computed split is no bigger than *max_tile_bytes*.

array_split.split.ShapeSplitter.max_tile_shape

ShapeSplitter.**max_tile_shape**
Per-axis maximum sizes for calculated tiles. A 1D `numpy.ndarray` of `int` indicating the per-axis maximum number of elements for tiles in the calculated split.

array_split.split.ShapeSplitter.split_begs

ShapeSplitter.**split_begs**
The list of per-axis start indices for `slice` objects. A `list` of 1D `numpy.ndarray` objects indicating the `slice.start` index for for tiles.

array_split.split.ShapeSplitter.split_ends

ShapeSplitter.**split_ends**
The list of per-axis stop indices for `slice` objects. A `list` of 1D `numpy.ndarray` objects indicating the `slice.stop` index for for tiles.

array_split.split.ShapeSplitter.split_num_slices_per_axis

ShapeSplitter.**split_num_slices_per_axis**
Number of slices per axis. A 1D `numpy.ndarray` of `int` indicating the number of slices (sections) per axis, so that *self*`.split_num_slices_per_axis[i]` is an integer indicating the number of sections along axis `i` in the calculated split.

array_split.split.ShapeSplitter.split_shape

ShapeSplitter.**split_shape**
The shape of the calculated split array. Indicates the per-axis number of sections in the calculated split. A 1D `numpy.ndarray`.

array_split.split.ShapeSplitter.split_size

ShapeSplitter.**split_size**
An `int` indicating the number of tiles in the calculated split.

ShapeSplitter.**sub_tile_shape**
Calculated tile shape will be an integer multiple of this sub-tile shape. i.e. `(self.tile_shape[i] % self.sub_tile_shape[i]) == 0`, for `i in range(0, len(self.tile_shape))`. A 1D `numpy.ndarray` of `int` indicating sub-tile shape.

array_split.split.ShapeSplitter.tile_beg_min

ShapeSplitter.**tile_beg_min**
The per-axis minimum index for `slice.start`. The per-axis lower bound for tile start indices. A 1D `numpy.ndarray`.

array_split.split.ShapeSplitter.tile_bounds_policy

ShapeSplitter.**tile_bounds_policy**
A string indicating whether tile halo extents can extend beyond the array domain. Valid values are indicated by *valid_tile_bounds_policies*.

array_split.split.ShapeSplitter.tile_end_max

ShapeSplitter.**tile_end_max**
The per-axis maximum index for `slice.stop`. The per-axis upper bound for tile stop indices. A 1D `numpy.ndarray`.

array_split.split.ShapeSplitter.tile_shape

ShapeSplitter.**tile_shape**
The shape of all tiles in the calculated split. A 1D `numpy.ndarray` of `int` indicating the per-axis number of elements for tiles in the calculated split.

array_split.split.ShapeSplitter.valid_tile_bounds_policies

ShapeSplitter.**valid_tile_bounds_policies = [<property object at 0x7f90c41ab0e8>, <property object at 0x7**
Class attribute indicating list of valid values for `tile_bound_policy`. See *ARRAY_BOUNDS* and *NO_BOUNDS*.

array_split.split.shape_split

array_split.split.**shape_split**(*array_shape*, **args*, ***kwargs*)
Splits specified *array_shape* in tiles, returns array of `slice` tuples.

Parameters

- **array_shape** (sequence of `int`) – The shape to be *split*.
- **indices_or_sections** (`None`, `int` or sequence of `int`) – If an integer, indicates the number of elements in the calculated *split* array. If a sequence, indicates the indicies (per axis) at which the splits occur. See *Splitting by number of tiles* examples.

- **axis** (None, int or sequence of int) – If an integer, indicates the axis which is to be split. If a sequence integers, indicates the number of slices per axis, i.e. if *axis* = [3, 5] then axis 0 is cut into 3 slices and axis 1 is cut into 5 slices for a total of 15 (3*5) rectangular slices in the returned (3, 5) shaped split. See *Splitting by number of tiles* examples and *Splitting by per-axis cut indices* examples.
- **array_start** (None or sequence of int) – The start index. Defaults to [0,]*len(array_shape). The array indexing extents are assumed to range from *array_start* to *array_start + array_shape*. See *The array_start parameter* examples.
- **array_itemsize** (int or sequence of int) – Number of bytes per array element. Only relevant when *max_tile_bytes* is specified. See *Splitting by maximum bytes per tile* examples.
- **tile_shape** (None or sequence of int) – When not None, specifies explicit shape for tiles. Should be same length as *array_shape*. See *Splitting by tile shape* examples.
- **max_tile_bytes** (None or int) – The maximum number of bytes for calculated tile_shape. See *Splitting by maximum bytes per tile* examples.
- **max_tile_shape** (None or sequence of int) – Per axis maximum shapes for the calculated tile_shape. Only relevant when *max_tile_bytes* is specified. Should be same length as *array_shape*. See *Splitting by maximum bytes per tile* examples.
- **sub_tile_shape** (None or sequence of int) – When not None, the calculated tile_shape will be an even multiple of this sub-tile shape. Only relevant when *max_tile_bytes* is specified. Should be same length as *array_shape*. See *Splitting by maximum bytes per tile* examples.
- **halo** (None, int, sequence of int, or (len(*array_shape*), 2) shaped numpy.ndarray) – How tiles are extended per axis in -ve and +ve directions with *halo* elements. See *The halo parameter* examples.
- **tile_bounds_policy** (str) – Specifies whether tiles can extend beyond the array boundaries. Only relevant for halo values greater than one. If *tile_bounds_policy* is *ARRAY_BOUNDS* then the calculated tiles will not extend beyond the array extents *array_start* and *array_start + array_shape*. If *tile_bounds_policy* is *NO_BOUNDS* then the returned tiles will extend beyond the *array_start* and *array_start + array_shape* extend for positive *halo* values. See *The halo parameter* examples.

Return type numpy.ndarray

Returns Array of tuple objects. Each tuple element is a slice object so that each tuple defines a multi-dimensional slice of an array of shape *array_shape*.

See also:

array_split.array_split(), *array_split.ShapeSplitter()*, *array_split Examples*

array_split.split.array_split

array_split.split.**array_split**(*ary*, *indices_or_sections=None*, *axis=None*, *tile_shape=None*, *max_tile_bytes=None*, *max_tile_shape=None*, *sub_tile_shape=None*, *halo=None*)

Splits the specified array *ary* into sub-arrays, returns list of numpy.ndarray.

Parameters

- **ary** (numpy.ndarray) – Array which is split into sub-arrays.
- **indices_or_sections** (None, int or sequence of int) – If an integer, indicates the number of elements in the calculated *split* array. If a sequence, indicates the indicies (per axis) at which the splits occur. See *Splitting by number of tiles* examples.
- **axis** (None, int or sequence of int) – If an integer, indicates the axis which is to be split. If a sequence integers, indicates the number of slices per axis, i.e. if *axis* = [3, 5] then axis 0 is cut into 3 slices and axis 1 is cut into 5 slices for a total of 15 (3*5) rectangular slices in the returned (3, 5) shaped split. See *Splitting by number of tiles* examples and *Splitting by per-axis cut indices* examples.
- **tile_shape** (None or sequence of int) – When not None, specifies explicit shape for tiles. Should be same length as *array_shape*. See *Splitting by tile shape* examples.
- **max_tile_bytes** (None or int) – The maximum number of bytes for calculated tile_shape. See *Splitting by maximum bytes per tile* examples.
- **max_tile_shape** (None or sequence of int) – Per axis maximum shapes for the calculated tile_shape. Only relevant when *max_tile_bytes* is specified. Should be same length as *array_shape*. See *Splitting by maximum bytes per tile* examples.
- **sub_tile_shape** (None or sequence of int) – When not None, the calculated tile_shape will be an even multiple of this sub-tile shape. Only relevant when *max_tile_bytes* is specified. Should be same length as *array_shape*. See *Splitting by maximum bytes per tile* examples.
- **halo** (None, int, sequence of int, or (len(*ary*.shape), 2) shaped numpy.ndarray) – How tiles are extended per axis in -ve and +ve directions with *halo* elements. See *The halo parameter* examples.

Return type list

Returns List of numpy.ndarray elements, where each element is a *slice* from *ary* (potentially an empty slice).

See also:

array_split.shape_split(), *array_split.ShapeSplitter()*, *array_split Examples*

3.2.2 Attributes

array_split.split.**ARRAY_BOUNDS = <property object>**
Indicates that tiles are always within the array bounds, resulting in tiles which have truncated halos. See *The halo parameter* examples.

array_split.split.**NO_BOUNDS = <property object>**
Indicates that tiles may have halos which extend beyond the array bounds. See *The halo parameter* examples.

3.3 The array_split.split_test Module

Module defining *array_split.split* unit-tests. Execute as:

```
python -m array_split.split_tests
```

3.3.1 Classes

SplitTest([methodName])	unittest.TestCase for *array_split.split* functions.

array_split.split_test.SplitTest

class array_split.split_test.**SplitTest**(*methodName='runTest'*)

unittest.TestCase for *array_split.split* functions.

Methods

__init__([methodName])	Create an instance of the class that will use the named test method
addCleanup(function, *args, **kwargs)	Add a function, with arguments, to be called when the test is comp
addTypeEqualityFunc(typeobj, function)	Add a type specific assertEqual style function to compare a type.
assertAlmostEqual(first, second[, places, ...])	Fail if the two objects are unequal as determined by their differenc
assertAlmostEquals(*args, **kwargs)	
assertArraySplitEqual(splt1, splt2)	Compares list of numpy.ndarray results returned by numpy
assertCountEqual(first, second[, msg])	An unordered sequence comparison asserting that the same elemer
assertDictContainsSubset(subset, dictionary)	Checks whether dictionary is a superset of subset.
assertDictEqual(d1, d2[, msg])	
assertEqual(first, second[, msg])	Fail if the two objects are unequal as determined by the '==' opera
assertEquals(*args, **kwargs)	
assertFalse(expr[, msg])	Check that the expression is false.
assertGreater(a, b[, msg])	Just like self.assertTrue(a > b), but with a nicer default message.
assertGreaterEqual(a, b[, msg])	Just like self.assertTrue(a >= b), but with a nicer default message.
assertIn(member, container[, msg])	Just like self.assertTrue(a in b), but with a nicer default message.
assertIs(expr1, expr2[, msg])	Just like self.assertTrue(a is b), but with a nicer default message.
assertIsInstance(obj, cls[, msg])	Same as self.assertTrue(isinstance(obj, cls)), with a nicer default m
assertIsNone(obj[, msg])	Same as self.assertTrue(obj is None), with a nicer default message
assertIsNot(expr1, expr2[, msg])	Just like self.assertTrue(a is not b), but with a nicer default messag
assertIsNotNone(obj[, msg])	Included for symmetry with assertIsNone.
assertItemsEqual(*args, **kwargs)	See unittest.TestCase.assertItemsEqual.
assertLess(a, b[, msg])	Just like self.assertTrue(a < b), but with a nicer default message.
assertLessEqual(a, b[, msg])	Just like self.assertTrue(a <= b), but with a nicer default message.
assertListEqual(*args, **kwargs)	See unittest.TestCase.assertListEqual.
assertLogs([logger, level])	Fail unless a log message of level *level* or higher is emitted on *log*
assertMultiLineEqual(first, second[, msg])	Assert that two multi-line strings are equal.
assertNotAlmostEqual(first, second[, ...])	Fail if the two objects are equal as determined by their difference r
assertNotAlmostEquals(*args, **kwargs)	
assertNotEqual(first, second[, msg])	Fail if the two objects are equal as determined by the '!=' operator
assertNotEquals(*args, **kwargs)	
assertNotIn(member, container[, msg])	Just like self.assertTrue(a not in b), but with a nicer default messag
assertNotIsInstance(obj, cls[, msg])	Included for symmetry with assertIsInstance.
assertNotRegex(text, unexpected_regex[, msg])	Fail the test if the text matches the regular expression.
assertRaises(excClass[, callableObj])	Fail unless an exception of class excClass is raised by callableObj
assertRaisesRegex(*args, **kwargs)	See unittest.TestCase.assertRaisesRegex.
assertRaisesRegexp(*args, **kwargs)	See unittest.TestCase.assertRaisesRegexp.
assertRegex(text, expected_regex[, msg])	Fail the test unless the text matches the regular expression.
assertRegexpMatches(*args, **kwargs)	
assertSequenceEqual(*args, **kwargs)	See unittest.TestCase.assertSequenceEqual.
assertSetEqual(*args, **kwargs)	See unittest.TestCase.assertSetEqual.

Table 3.

assertTrue(expr[, msg])	Check that the expression is true.
assertTupleEqual(*args, **kwargs)	See unittest.TestCase.assertTupleEqual.
assertWarns(expected_warning[, callable_obj])	Fail unless a warning of class warnClass is triggered by callable_o
assertWarnsRegex(*args, **kwargs)	See unittest.TestCase.assertWarnsRegex.
*assert_*(*args, **kwargs)	
countTestCases()	
debug()	Run the test without collecting errors in a TestResult
defaultTestResult()	
doCleanups()	Execute all cleanup functions.
fail([msg])	Fail immediately, with the given message.
failIf(*args, **kwargs)	
failIfAlmostEqual(*args, **kwargs)	
failIfEqual(*args, **kwargs)	
failUnless(*args, **kwargs)	
failUnlessAlmostEqual(*args, **kwargs)	
failUnlessEqual(*args, **kwargs)	
failUnlessRaises(*args, **kwargs)	
id()	
run([result])	
setUp()	Hook method for setting up the test fixture before exercising it.
setUpClass()	Hook method for setting up class fixture before running tests in the
shortDescription()	Returns a one-line description of the test, or None if no description
skipTest(reason)	Skip this test.
subTest([msg])	Return a context manager that will return the enclosed block of co
tearDown()	Hook method for deconstructing the test fixture after testing it.
tearDownClass()	Hook method for deconstructing the class fixture after running all
test_array_split()	Test for case for *array_split.split.array_split()*.
test_calculate_num_slices_per_axis()	Tests for *array_split.split.calculate_num_slices*
test_calculate_split_by_tile_max_bytes_1d()	
test_calculate_split_by_tile_shape_1d()	
test_calculate_split_by_tile_shape_2d()	
test_calculate_split_with_array_start_1d()	
test_calculate_split_with_array_start_2d()	
test_calculate_split_with_halo_1d()	
test_calculate_split_with_halo_2d()	
test_calculate_tile_shape_for_max_bytes_1d()	Test case for *array_split.split.calculate_tile_sha*
test_calculate_tile_shape_for_max_bytes_2d()	Test case for *array_split.split.calculate_tile_sha*
test_shape_factors()	Tests for *array_split.split.shape_factors()*.
test_split_by_num_slices()	Test for case for splitting by number of
test_split_by_per_axis_indices()	Test for case for splitting by specified

array_split.split_test.SplitTest.__init__

SplitTest.**__init__**(*methodName='runTest'*)

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

array_split.split_test.SplitTest.addCleanup

SplitTest.**addCleanup**(*function*, **args*, ***kwargs*)

Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

array_split.split_test.SplitTest.addTypeEqualityFunc

SplitTest.**addTypeEqualityFunc**(*typeobj*, *function*)

Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Args:

typeobj: The data type to call this function on when both values are of the same type in assertEqual().

function: The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

array_split.split_test.SplitTest.assertAlmostEqual

SplitTest.**assertAlmostEqual**(*first*, *second*, *places=None*, *msg=None*, *delta=None*)

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

array_split.split_test.SplitTest.assertAlmostEquals

SplitTest.**assertAlmostEquals**(**args*, ***kwargs*)

array_split.split_test.SplitTest.assertArraySplitEqual

SplitTest.**assertArraySplitEqual**(*splt1*, *splt2*)

Compares `list` of `numpy.ndarray` results returned by `numpy.array_split()` and *`array_split.split.array_split()`* functions.

Parameters

- **splt1** (`list` of `numpy.ndarray`) – First object in equality comparison.
- **splt2** (`list` of `numpy.ndarray`) – Second object in equality comparison.

Raises **unittest.AssertionError** – If any element of *`splt1`* is not equal to the corresponding element of `splt2`.

array_split.split_test.SplitTest.assertCountEqual

SplitTest.**assertCountEqual**(*first*, *second*, *msg=None*)

An unordered sequence comparison asserting that the same elements, regardless of order. If the same element occurs more than once, it verifies that the elements occur the same number of times.

self.assertEqual(Counter(list(first)), Counter(list(second)))

Example:

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

array_split.split_test.SplitTest.assertDictContainsSubset

SplitTest.**assertDictContainsSubset**(*subset*, *dictionary*, *msg=None*)

Checks whether dictionary is a superset of subset.

array_split.split_test.SplitTest.assertDictEqual

SplitTest.**assertDictEqual**(*d1*, *d2*, *msg=None*)

array_split.split_test.SplitTest.assertEqual

SplitTest.**assertEqual**(*first*, *second*, *msg=None*)

Fail if the two objects are unequal as determined by the '==' operator.

array_split.split_test.SplitTest.assertEquals

SplitTest.**assertEquals**(**args*, ***kwargs*)

array_split.split_test.SplitTest.assertFalse

SplitTest.**assertFalse**(*expr*, *msg=None*)

Check that the expression is false.

array_split.split_test.SplitTest.assertGreater

SplitTest.**assertGreater**(*a*, *b*, *msg=None*)

Just like self.assertTrue(a > b), but with a nicer default message.

array_split.split_test.SplitTest.assertGreaterEqual

SplitTest.**assertGreaterEqual**(*a*, *b*, *msg=None*)

Just like self.assertTrue(a >= b), but with a nicer default message.

array_split.split_test.SplitTest.assertIn

SplitTest.**assertIn**(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

array_split.split_test.SplitTest.assertIs

SplitTest.**assertIs**(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

array_split.split_test.SplitTest.assertIsInstance

SplitTest.**assertIsInstance**(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

array_split.split_test.SplitTest.assertIsNone

SplitTest.**assertIsNone**(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

array_split.split_test.SplitTest.assertIsNot

SplitTest.**assertIsNot**(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

array_split.split_test.SplitTest.assertIsNotNone

SplitTest.**assertIsNotNone**(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

array_split.split_test.SplitTest.assertItemsEqual

SplitTest.**assertItemsEqual**(**args*, ***kwargs*)
See unittest.TestCase.assertItemsEqual.

array_split.split_test.SplitTest.assertLess

SplitTest.**assertLess**(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

array_split.split_test.SplitTest.assertLessEqual

SplitTest.**assertLessEqual**(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

array_split.split_test.SplitTest.assertListEqual

SplitTest.**assertListEqual**(**args*, ***kwargs*)
See unittest.TestCase.assertListEqual.

array_split.split_test.SplitTest.assertLogs

SplitTest.**assertLogs**(*logger=None*, *level=None*)
Fail unless a log message of level *level* or higher is emitted on *logger_name* or its children. If omitted, *level* defaults to INFO and *logger* defaults to the root logger.

This method must be used as a context manager, and will yield a recording object with two attributes: *output* and *records*. At the end of the context manager, the *output* attribute will be a list of the matching formatted log messages and the *records* attribute will be a list of the corresponding LogRecord objects.

Example:

```
with self.assertLogs('foo', level='INFO') as cm:
    logging.getLogger('foo').info('first message')
    logging.getLogger('foo.bar').error('second message')
self.assertEqual(cm.output, ['INFO:foo:first message',
                             'ERROR:foo.bar:second message'])
```

array_split.split_test.SplitTest.assertMultiLineEqual

SplitTest.**assertMultiLineEqual**(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

array_split.split_test.SplitTest.assertNotAlmostEqual

SplitTest.**assertNotAlmostEqual**(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

array_split.split_test.SplitTest.assertNotAlmostEquals

SplitTest.**assertNotAlmostEquals**(**args*, ***kwargs*)

array_split.split_test.SplitTest.assertNotEqual

SplitTest.**assertNotEqual**(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

array_split.split_test.SplitTest.assertNotEquals

SplitTest.**assertNotEquals**(**args*, ***kwargs*)

array_split.split_test.SplitTest.assertNotIn

SplitTest.**assertNotIn**(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

array_split.split_test.SplitTest.assertNotIsInstance

SplitTest.**assertNotIsInstance**(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

array_split.split_test.SplitTest.assertNotRegex

SplitTest.**assertNotRegex**(*text*, *unexpected_regex*, *msg=None*)
Fail the test if the text matches the regular expression.

array_split.split_test.SplitTest.assertRaises

SplitTest.**assertRaises**(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

```
with self.assertRaises(SomeException):
    do_something()
```

An optional keyword argument 'msg' can be provided when assertRaises is used as a context object.

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

```
with self.assertRaises(SomeException) as cm:
    do_something()
the_exception = cm.exception
self.assertEqual(the_exception.error_code, 3)
```

array_split.split_test.SplitTest.assertRaisesRegex

SplitTest.**assertRaisesRegex**(**args*, ***kwargs*)
See unittest.TestCase.assertRaisesRegex.

array_split.split_test.SplitTest.assertRaisesRegexp

SplitTest.**assertRaisesRegexp**(**args*, ***kwargs*)
See unittest.TestCase.assertRaisesRegexp.

array_split.split_test.SplitTest.assertRegex

SplitTest.**assertRegex**(*text*, *expected_regex*, *msg=None*)
Fail the test unless the text matches the regular expression.

array_split.split_test.SplitTest.assertRegexpMatches

SplitTest.**assertRegexpMatches**(**args*, ***kwargs*)

array_split.split_test.SplitTest.assertSequenceEqual

SplitTest.**assertSequenceEqual**(**args*, ***kwargs*)
See unittest.TestCase.assertSequenceEqual.

array_split.split_test.SplitTest.assertSetEqual

SplitTest.**assertSetEqual**(**args*, ***kwargs*)
See unittest.TestCase.assertSetEqual.

array_split.split_test.SplitTest.assertTrue

SplitTest.**assertTrue**(*expr*, *msg=None*)
Check that the expression is true.

array_split.split_test.SplitTest.assertTupleEqual

SplitTest.**assertTupleEqual**(**args*, ***kwargs*)
See unittest.TestCase.assertTupleEqual.

array_split.split_test.SplitTest.assertWarns

SplitTest.**assertWarns**(*expected_warning*, *callable_obj=None*, **args*, ***kwargs*)
Fail unless a warning of class warnClass is triggered by callable_obj when invoked with arguments args and keyword arguments kwargs. If a different type of warning is triggered, it will not be handled: depending on the other warning filtering rules in effect, it might be silenced, printed out, or raised as an exception.

If called with callable_obj omitted or None, will return a context object used like this:

```
with self.assertWarns(SomeWarning):
    do_something()
```

An optional keyword argument 'msg' can be provided when assertWarns is used as a context object.

The context manager keeps a reference to the first matching warning as the 'warning' attribute; similarly, the 'filename' and 'lineno' attributes give you information about the line of Python code from which the warning was triggered. This allows you to inspect the warning after the assertion:

```
        with self.assertWarns(SomeWarning) as cm:
            do_something()
        the_warning = cm.warning
        self.assertEqual(the_warning.some_attribute, 147)
```

array_split.split_test.SplitTest.assertWarnsRegex

SplitTest.**assertWarnsRegex**(**args*, ***kwargs*)
See unittest.TestCase.assertWarnsRegex.

array_split.split_test.SplitTest.assert

SplitTest.**assert_**(**args*, ***kwargs*)

array_split.split_test.SplitTest.countTestCases

SplitTest.**countTestCases**()

array_split.split_test.SplitTest.debug

SplitTest.**debug**()
Run the test without collecting errors in a TestResult

array_split.split_test.SplitTest.defaultTestResult

SplitTest.**defaultTestResult**()

array_split.split_test.SplitTest.doCleanups

SplitTest.**doCleanups**()
Execute all cleanup functions. Normally called for you after tearDown.

array_split.split_test.SplitTest.fail

SplitTest.**fail**(*msg=None*)
Fail immediately, with the given message.

array_split.split_test.SplitTest.failIf

SplitTest.**failIf**(**args*, ***kwargs*)

array_split.split_test.SplitTest.failIfAlmostEqual

SplitTest.**failIfAlmostEqual**(**args*, ***kwargs*)

array_split.split_test.SplitTest.failIfEqual

SplitTest.**failIfEqual**(**args*, ***kwargs*)

array_split.split_test.SplitTest.failUnless

SplitTest.**failUnless**(**args*, ***kwargs*)

array_split.split_test.SplitTest.failUnlessAlmostEqual

SplitTest.**failUnlessAlmostEqual**(**args*, ***kwargs*)

array_split.split_test.SplitTest.failUnlessEqual

SplitTest.**failUnlessEqual**(**args*, ***kwargs*)

array_split.split_test.SplitTest.failUnlessRaises

SplitTest.**failUnlessRaises**(**args*, ***kwargs*)

array_split.split_test.SplitTest.id

SplitTest.**id**()

array_split.split_test.SplitTest.run

SplitTest.**run**(*result=None*)

array_split.split_test.SplitTest.setUp

SplitTest.**setUp**()
Hook method for setting up the test fixture before exercising it.

array_split.split_test.SplitTest.setUpClass

SplitTest.**setUpClass**()
Hook method for setting up class fixture before running tests in the class.

array_split.split_test.SplitTest.shortDescription

SplitTest.**shortDescription**()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

array_split.split_test.SplitTest.skipTest

SplitTest.**skipTest**(*reason*)
Skip this test.

array_split.split_test.SplitTest.subTest

SplitTest.**subTest**(*msg=None*, ***params*)
Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters. A failure in the subtest marks the test case as failed but resumes execution at the end of the enclosed block, allowing further test code to be executed.

array_split.split_test.SplitTest.tearDown

SplitTest.**tearDown**()
Hook method for deconstructing the test fixture after testing it.

array_split.split_test.SplitTest.tearDownClass

SplitTest.**tearDownClass**()
Hook method for deconstructing the class fixture after running all tests in the class.

array_split.split_test.SplitTest.test_array_split

SplitTest.**test_array_split**()
Test for case for *array_split.split.array_split()*.

array_split.split_test.SplitTest.test_calculate_num_slices_per_axis

SplitTest.**test_calculate_num_slices_per_axis**()
Tests for *array_split.split.calculate_num_slices_per_axis()*.

array_split.split_test.SplitTest.test_calculate_split_by_tile_max_bytes_1d

SplitTest.**test_calculate_split_by_tile_max_bytes_1d**()

array_split.split_test.SplitTest.test_calculate_split_by_tile_shape_1d

SplitTest.**test_calculate_split_by_tile_shape_1d**()

array_split.split_test.SplitTest.test_calculate_split_by_tile_shape_2d

SplitTest.**test_calculate_split_by_tile_shape_2d**()

array_split.split_test.SplitTest.test_calculate_split_with_array_start_1d

SplitTest.**test_calculate_split_with_array_start_1d**()

array_split.split_test.SplitTest.test_calculate_split_with_array_start_2d

SplitTest.**test_calculate_split_with_array_start_2d**()

array_split.split_test.SplitTest.test_calculate_split_with_halo_1d

SplitTest.**test_calculate_split_with_halo_1d**()

array_split.split_test.SplitTest.test_calculate_split_with_halo_2d

SplitTest.**test_calculate_split_with_halo_2d**()

array_split.split_test.SplitTest.test_calculate_tile_shape_for_max_bytes_1d

SplitTest.**test_calculate_tile_shape_for_max_bytes_1d**()
Test case for *array_split.split.calculate_tile_shape_for_max_bytes()*, where `array_shape` parameter is 1D, i.e. of the form `(N,)`.

array_split.split_test.SplitTest.test_calculate_tile_shape_for_max_bytes_2d

SplitTest.**test_calculate_tile_shape_for_max_bytes_2d**()
Test case for *array_split.split.calculate_tile_shape_for_max_bytes()*, where `array_shape` parameter is 2D, i.e. of the form `(H,W)`.

array_split.split_test.SplitTest.test_shape_factors

SplitTest.**test_shape_factors**()
Tests for *array_split.split.shape_factors()*.

array_split.split_test.SplitTest.test_split_by_num_slices

SplitTest.**test_split_by_num_slices**()
Test for case for splitting by number of slice elements:

```
ShapeSplitter(array_shape=(10, 13), indices_or_sections=3).calculate_split()
ShapeSplitter(array_shape=(10, 13), axis=[2, 3]).calculate_split()
```

array_split.split_test.SplitTest.test_split_by_per_axis_indices

SplitTest.**test_split_by_per_axis_indices**()
Test for case for splitting by specified indices:

```
        ShapeSplitter(array_shape=(10, 4), indices_or_sections=[[2, 6, 8], ]).calculate_split()
```

Attributes

logger	Class attribute for logging.Logger logging.
longMessage	
maxDiff	

array_split.split_test.SplitTest.logger

SplitTest.**logger** = <logging.Logger object>
Class attribute for logging.Logger logging.

array_split.split_test.SplitTest.longMessage

SplitTest.**longMessage** = True

array_split.split_test.SplitTest.maxDiff

SplitTest.**maxDiff** = 640

3.4 The array_split.tests Module

Module for running all *array_split* unit-tests, including unittest test-cases and doctest tests for module doc-strings and sphinx (RST) documentation. Execute as:

```
python -m array_split.tests
```

3.5 The array_split.logging Module

Default initialisation of python logging.

Some simple wrappers of python built-in logging module for *array_split* logging.

3.5.1 Classes and Functions

SplitStreamHandler([outstr, errstr, splitlevel])	A python logging.handlers Handler class for splitting logging messa
initialise_loggers(names[, log_level, ...])	Initialises specified loggers to generate output at the specified logging level.
get_formatter([prefix_string])	Returns logging.Formatter object which produces messages with *time* a

array_split.logging.SplitStreamHandler

class `array_split.logging.`**`SplitStreamHandler`**(*outstr=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *errstr=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*, *splitlevel=30*)

A python `logging.handlers` `Handler` class for splitting logging messages to different streams depending on the logging-level.

Methods

__init__([outstr, errstr, splitlevel])	Initialise with a pair of streams and a threshold level which determines the stream where the m
acquire()	Acquire the I/O thread lock.
addFilter(filter)	Add the specified filter to this handler.
close()	Tidy up any resources used by the handler.
createLock()	Acquire a thread lock for serializing access to the underlying I/O.
emit(record)	Emit a record.
filter(record)	Determine if a record is loggable by consulting all the filters.
flush()	Flushes the stream.
format(record)	Format the specified record.
get_name()	
handle(record)	Conditionally emit the specified logging record.
handleError(record)	Handle errors which occur during an emit() call.
release()	Release the I/O thread lock.
removeFilter(filter)	Remove the specified filter from this handler.
setFormatter(fmt)	Set the formatter for this handler.
setLevel(level)	Set the logging level of this handler.
set_name(name)	

array_split.logging.SplitStreamHandler.__init__

`SplitStreamHandler.`**`__init__`**(*outstr=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *errstr=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*, *splitlevel=30*)

Initialise with a pair of streams and a threshold level which determines the stream where the messages are writting.

Parameters

- **outstr** (*file-like*) – Logging messages are written to this stream if the message level is less than `self.splitLevel`.
- **errstr** (*stream*) – Logging messages are written to this stream if the message level is greater-than-or-equal-to `self.splitLevel`.
- **splitlevel** (*int*) – Logging level threshold determining split streams for log messages.

array_split.logging.SplitStreamHandler.acquire

SplitStreamHandler.**acquire**()
Acquire the I/O thread lock.

array_split.logging.SplitStreamHandler.addFilter

SplitStreamHandler.**addFilter**(*filter*)
Add the specified filter to this handler.

array_split.logging.SplitStreamHandler.close

SplitStreamHandler.**close**()
Tidy up any resources used by the handler.

This version removes the handler from an internal map of handlers, _handlers, which is used for handler lookup by name. Subclasses should ensure that this gets called from overridden close() methods.

array_split.logging.SplitStreamHandler.createLock

SplitStreamHandler.**createLock**()
Acquire a thread lock for serializing access to the underlying I/O.

array_split.logging.SplitStreamHandler.emit

SplitStreamHandler.**emit**(*record*)
Emit a record.

If a formatter is specified, it is used to format the record. The record is then written to the stream with a trailing newline. If exception information is present, it is formatted using traceback.print_exception and appended to the stream. If the stream has an 'encoding' attribute, it is used to determine how to do the output to the stream.

array_split.logging.SplitStreamHandler.filter

SplitStreamHandler.**filter**(*record*)
Determine if a record is loggable by consulting all the filters.

The default is to allow the record to be logged; any filter can veto this and the record is then dropped. Returns a zero value if a record is to be dropped, else non-zero.

array_split.logging.SplitStreamHandler.flush

SplitStreamHandler.**flush**()
Flushes the stream.

array_split.logging.SplitStreamHandler.format

SplitStreamHandler.**format**(*record*)
Format the specified record.

If a formatter is set, use it. Otherwise, use the default formatter for the module.

array_split.logging.SplitStreamHandler.get_name

SplitStreamHandler.**get_name**()

array_split.logging.SplitStreamHandler.handle

SplitStreamHandler.**handle**(*record*)
Conditionally emit the specified logging record.

Emission depends on filters which may have been added to the handler. Wrap the actual emission of the record with acquisition/release of the I/O thread lock. Returns whether the filter passed the record for emission.

array_split.logging.SplitStreamHandler.handleError

SplitStreamHandler.**handleError**(*record*)
Handle errors which occur during an emit() call.

This method should be called from handlers when an exception is encountered during an emit() call. If raiseExceptions is false, exceptions get silently ignored. This is what is mostly wanted for a logging system - most users will not care about errors in the logging system, they are more interested in application errors. You could, however, replace this with a custom handler if you wish. The record which was being processed is passed in to this method.

array_split.logging.SplitStreamHandler.release

SplitStreamHandler.**release**()
Release the I/O thread lock.

array_split.logging.SplitStreamHandler.removeFilter

SplitStreamHandler.**removeFilter**(*filter*)
Remove the specified filter from this handler.

array_split.logging.SplitStreamHandler.setFormatter

SplitStreamHandler.**setFormatter**(*fmt*)
Set the formatter for this handler.

array_split.logging.SplitStreamHandler.setLevel

SplitStreamHandler.**setLevel**(*level*)

Set the logging level of this handler. level must be an int or a str.

array_split.logging.SplitStreamHandler.set_name

SplitStreamHandler.**set_name**(*name*)

Attributes

name
terminator

array_split.logging.SplitStreamHandler.name

SplitStreamHandler.**name**

array_split.logging.SplitStreamHandler.terminator

SplitStreamHandler.**terminator** = '\n'

array_split.logging.initialise_loggers

array_split.logging.**initialise_loggers**(*names*, *log_level=30*, *handler_class=<class 'array_split.logging.SplitStreamHandler'>*)

Initialises specified loggers to generate output at the specified logging level. If the specified named loggers do not exist, they are created.

Parameters

- **names** (list of str) – List of logger names.
- **log_level** (int) – Log level for messages, typically one of logging.DEBUG, logging.INFO, logging.WARN, logging.ERROR or logging.CRITICAL. See Logging Levels.
- **handler_class** (One of the logging.handlers classes.) – The handler class for output of log messages, for example *SplitStreamHandler* or logging.StreamHandler.

array_split.logging.get_formatter

array_split.logging.**get_formatter**(*prefix_string='ARRSPLT\ '*)

Returns logging.Formatter object which produces messages with *time* and prefix_string prefix.

Parameters **prefix_string** (str or None) – Prefix for all logging messages.

Return type logging.Formatter

Returns Regular formatter for logging.

3.6 The array_split.unittest Module

Some simple wrappers of python built-in unittest module for *array_split* unit-tests.

3.6.1 Classes and Functions

main(module_name[, log_level, init_logger_names])	Small wrapper for unittest.main() which initialises logging.Logg
TestCase([methodName])	Extends unittest.TestCase with the assertArraySplitEqual()

array_split.unittest.main

array_split.unittest.**main**(*module_name*, *log_level=10*, *init_logger_names=None*)

Small wrapper for unittest.main() which initialises logging.Logger objects. Loads a set of tests from module and runs them; this is primarily for making test modules conveniently executable. The simplest use for this function is to include the following line at the end of a test module:

```
array_split.unittest.main(__name__)
```

If __name__ == "__main__", then *discoverable* unittest.TestCase test cases are executed. Logging level can be explicitly set for a group of modules using:

```
import logging

array_split.unittest.main(
    __name__,
    logging.DEBUG,
    [__name__, "module_name_0", "module_name_1", "package.module_name_2"]
)
```

Parameters

- **module_name** (str) – If *module_name* == "__main__" then unit-tests are *discovered* and run.
- **log_level** (int) – The default logging level for all array_split.logging.Logger objects.
- **init_logger_names** (sequence of str) – List of logger names to initialise (using *array_split.logging.initialise_loggers()*). If None, then the list defaults to [*module_name*, "array_split"]. If list is empty no loggers are initialised.

array_split.unittest.TestCase

class array_split.unittest.**TestCase**(*methodName='runTest'*)

Extends unittest.TestCase with the *assertArraySplitEqual()*.

Methods

__init__([methodName])	Create an instance of the class that will use the named test method when exe
addCleanup(function, *args, **kwargs)	Add a function, with arguments, to be called when the test is completed.

Table 3.14 –

addTypeEqualityFunc(typeobj, function)	Add a type specific assertEqual style function to compare a type.
assertAlmostEqual(first, second[, places, ...])	Fail if the two objects are unequal as determined by their difference rounded
assertAlmostEquals(*args, **kwargs)	
assertArraySplitEqual(splt1, splt2)	Compares `list` of `numpy.ndarray` results returned by `numpy.array`
assertCountEqual(first, second[, msg])	An unordered sequence comparison asserting that the same elements, regard
assertDictContainsSubset(subset, dictionary)	Checks whether dictionary is a superset of subset.
assertDictEqual(d1, d2[, msg])	
assertEqual(first, second[, msg])	Fail if the two objects are unequal as determined by the '==' operator.
assertEquals(*args, **kwargs)	
assertFalse(expr[, msg])	Check that the expression is false.
assertGreater(a, b[, msg])	Just like self.assertTrue(a > b), but with a nicer default message.
assertGreaterEqual(a, b[, msg])	Just like self.assertTrue(a >= b), but with a nicer default message.
assertIn(member, container[, msg])	Just like self.assertTrue(a in b), but with a nicer default message.
assertIs(expr1, expr2[, msg])	Just like self.assertTrue(a is b), but with a nicer default message.
assertIsInstance(obj, cls[, msg])	Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.
assertIsNone(obj[, msg])	Same as self.assertTrue(obj is None), with a nicer default message.
assertIsNot(expr1, expr2[, msg])	Just like self.assertTrue(a is not b), but with a nicer default message.
assertIsNotNone(obj[, msg])	Included for symmetry with assertIsNone.
assertItemsEqual(*args, **kwargs)	See `unittest.TestCase.assertItemsEqual`.
assertLess(a, b[, msg])	Just like self.assertTrue(a < b), but with a nicer default message.
assertLessEqual(a, b[, msg])	Just like self.assertTrue(a <= b), but with a nicer default message.
assertListEqual(*args, **kwargs)	See `unittest.TestCase.assertListEqual`.
assertLogs([logger, level])	Fail unless a log message of level *level* or higher is emitted on *logger_name*
assertMultiLineEqual(first, second[, msg])	Assert that two multi-line strings are equal.
assertNotAlmostEqual(first, second[, ...])	Fail if the two objects are equal as determined by their difference rounded to
assertNotAlmostEquals(*args, **kwargs)	
assertNotEqual(first, second[, msg])	Fail if the two objects are equal as determined by the '!=' operator.
assertNotEquals(*args, **kwargs)	
assertNotIn(member, container[, msg])	Just like self.assertTrue(a not in b), but with a nicer default message.
assertNotIsInstance(obj, cls[, msg])	Included for symmetry with assertIsInstance.
assertNotRegex(text, unexpected_regex[, msg])	Fail the test if the text matches the regular expression.
assertRaises(excClass[, callableObj])	Fail unless an exception of class excClass is raised by callableObj when inv
assertRaisesRegex(*args, **kwargs)	See `unittest.TestCase.assertRaisesRegex`.
assertRaisesRegexp(*args, **kwargs)	See `unittest.TestCase.assertRaisesRegexp`.
assertRegex(text, expected_regex[, msg])	Fail the test unless the text matches the regular expression.
assertRegexpMatches(*args, **kwargs)	
assertSequenceEqual(*args, **kwargs)	See `unittest.TestCase.assertSequenceEqual`.
assertSetEqual(*args, **kwargs)	See `unittest.TestCase.assertSetEqual`.
assertTrue(expr[, msg])	Check that the expression is true.
assertTupleEqual(*args, **kwargs)	See `unittest.TestCase.assertTupleEqual`.
assertWarns(expected_warning[, callable_obj])	Fail unless a warning of class warnClass is triggered by callable_obj when i
assertWarnsRegex(*args, **kwargs)	See `unittest.TestCase.assertWarnsRegex`.
*assert_*(*args, **kwargs)	
countTestCases()	
debug()	Run the test without collecting errors in a TestResult
defaultTestResult()	
doCleanups()	Execute all cleanup functions.
fail([msg])	Fail immediately, with the given message.
failIf(*args, **kwargs)	
failIfAlmostEqual(*args, **kwargs)	

Table 3.14 –

failIfEqual(*args, **kwargs)	
failUnless(*args, **kwargs)	
failUnlessAlmostEqual(*args, **kwargs)	
failUnlessEqual(*args, **kwargs)	
failUnlessRaises(*args, **kwargs)	
id()	
run([result])	
setUp()	Hook method for setting up the test fixture before exercising it.
setUpClass()	Hook method for setting up class fixture before running tests in the class.
shortDescription()	Returns a one-line description of the test, or None if no description has beer
skipTest(reason)	Skip this test.
subTest([msg])	Return a context manager that will return the enclosed block of code in a su
tearDown()	Hook method for deconstructing the test fixture after testing it.
tearDownClass()	Hook method for deconstructing the class fixture after running all tests in th

array_split.unittest.TestCase.__init__

TestCase.**__init__**(*methodName='runTest'*)

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

array_split.unittest.TestCase.addCleanup

TestCase.**addCleanup**(*function*, **args*, ***kwargs*)

Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

array_split.unittest.TestCase.addTypeEqualityFunc

TestCase.**addTypeEqualityFunc**(*typeobj*, *function*)

Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Args:

typeobj: The data type to call this function on when both values are of the same type in assertEqual().

function: The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

array_split.unittest.TestCase.assertAlmostEqual

TestCase.**assertAlmostEqual**(*first*, *second*, *places=None*, *msg=None*, *delta=None*)

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

array_split.unittest.TestCase.assertAlmostEquals

`TestCase.`**`assertAlmostEquals`**(**args*, ***kwargs*)

array_split.unittest.TestCase.assertArraySplitEqual

`TestCase.`**`assertArraySplitEqual`**(*splt1*, *splt2*)

Compares `list` of `numpy.ndarray` results returned by `numpy.array_split()` and *`array_split.split.array_split()`* functions.

Parameters

- **`splt1`** (`list` of `numpy.ndarray`) – First object in equality comparison.
- **`splt2`** (`list` of `numpy.ndarray`) – Second object in equality comparison.

Raises **`unittest.AssertionError`** – If any element of *`splt1`* is not equal to the corresponding element of `splt2`.

array_split.unittest.TestCase.assertCountEqual

`TestCase.`**`assertCountEqual`**(*first*, *second*, *msg=None*)

An unordered sequence comparison asserting that the same elements, regardless of order. If the same element occurs more than once, it verifies that the elements occur the same number of times.

self.assertEqual(Counter(list(first)), Counter(list(second)))

Example:

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

array_split.unittest.TestCase.assertDictContainsSubset

`TestCase.`**`assertDictContainsSubset`**(*subset*, *dictionary*, *msg=None*)

Checks whether dictionary is a superset of subset.

array_split.unittest.TestCase.assertDictEqual

`TestCase.`**`assertDictEqual`**(*d1*, *d2*, *msg=None*)

array_split.unittest.TestCase.assertEqual

`TestCase.`**`assertEqual`**(*first*, *second*, *msg=None*)

Fail if the two objects are unequal as determined by the '==' operator.

array_split.unittest.TestCase.assertEquals

TestCase.**assertEquals**(**args*, ***kwargs*)

array_split.unittest.TestCase.assertFalse

TestCase.**assertFalse**(*expr*, *msg=None*)
Check that the expression is false.

array_split.unittest.TestCase.assertGreater

TestCase.**assertGreater**(*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

array_split.unittest.TestCase.assertGreaterEqual

TestCase.**assertGreaterEqual**(*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

array_split.unittest.TestCase.assertIn

TestCase.**assertIn**(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

array_split.unittest.TestCase.assertIs

TestCase.**assertIs**(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

array_split.unittest.TestCase.assertIsInstance

TestCase.**assertIsInstance**(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

array_split.unittest.TestCase.assertIsNone

TestCase.**assertIsNone**(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

array_split.unittest.TestCase.assertIsNot

TestCase.**assertIsNot**(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

array_split.unittest.TestCase.assertIsNotNone

TestCase.**assertIsNotNone**(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

array_split.unittest.TestCase.assertItemsEqual

TestCase.**assertItemsEqual**(**args*, ***kwargs*)
See unittest.TestCase.assertItemsEqual.

array_split.unittest.TestCase.assertLess

TestCase.**assertLess**(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

array_split.unittest.TestCase.assertLessEqual

TestCase.**assertLessEqual**(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

array_split.unittest.TestCase.assertListEqual

TestCase.**assertListEqual**(**args*, ***kwargs*)
See unittest.TestCase.assertListEqual.

array_split.unittest.TestCase.assertLogs

TestCase.**assertLogs**(*logger=None*, *level=None*)
Fail unless a log message of level *level* or higher is emitted on *logger_name* or its children. If omitted, *level* defaults to INFO and *logger* defaults to the root logger.

This method must be used as a context manager, and will yield a recording object with two attributes: *output* and *records*. At the end of the context manager, the *output* attribute will be a list of the matching formatted log messages and the *records* attribute will be a list of the corresponding LogRecord objects.

Example:

```
with self.assertLogs('foo', level='INFO') as cm:
    logging.getLogger('foo').info('first message')
    logging.getLogger('foo.bar').error('second message')
self.assertEqual(cm.output, ['INFO:foo:first message',
                             'ERROR:foo.bar:second message'])
```

array_split.unittest.TestCase.assertMultiLineEqual

TestCase.**assertMultiLineEqual**(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

array_split.unittest.TestCase.assertNotAlmostEqual

TestCase.**assertNotAlmostEqual**(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

array_split.unittest.TestCase.assertNotAlmostEquals

TestCase.**assertNotAlmostEquals**(**args*, ***kwargs*)

array_split.unittest.TestCase.assertNotEqual

TestCase.**assertNotEqual**(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

array_split.unittest.TestCase.assertNotEquals

TestCase.**assertNotEquals**(**args*, ***kwargs*)

array_split.unittest.TestCase.assertNotIn

TestCase.**assertNotIn**(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

array_split.unittest.TestCase.assertNotIsInstance

TestCase.**assertNotIsInstance**(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

array_split.unittest.TestCase.assertNotRegex

TestCase.**assertNotRegex**(*text*, *unexpected_regex*, *msg=None*)
Fail the test if the text matches the regular expression.

array_split.unittest.TestCase.assertRaises

TestCase.**assertRaises**(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

```
        with self.assertRaises(SomeException):
            do_something()
```

An optional keyword argument 'msg' can be provided when assertRaises is used as a context object.

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

```
        with self.assertRaises(SomeException) as cm:
            do_something()
        the_exception = cm.exception
        self.assertEqual(the_exception.error_code, 3)
```

array_split.unittest.TestCase.assertRaisesRegex

TestCase.**assertRaisesRegex**(**args*, ***kwargs*)
See unittest.TestCase.assertRaisesRegex.

array_split.unittest.TestCase.assertRaisesRegexp

TestCase.**assertRaisesRegexp**(**args*, ***kwargs*)
See unittest.TestCase.assertRaisesRegexp.

array_split.unittest.TestCase.assertRegex

TestCase.**assertRegex**(*text*, *expected_regex*, *msg=None*)
Fail the test unless the text matches the regular expression.

array_split.unittest.TestCase.assertRegexpMatches

TestCase.**assertRegexpMatches**(**args*, ***kwargs*)

array_split.unittest.TestCase.assertSequenceEqual

TestCase.**assertSequenceEqual**(**args*, ***kwargs*)
See unittest.TestCase.assertSequenceEqual.

array_split.unittest.TestCase.assertSetEqual

TestCase.**assertSetEqual**(**args*, ***kwargs*)
See unittest.TestCase.assertSetEqual.

array_split.unittest.TestCase.assertTrue

TestCase.**assertTrue**(*expr*, *msg=None*)
Check that the expression is true.

array_split.unittest.TestCase.assertTupleEqual

TestCase.**assertTupleEqual**(**args*, ***kwargs*)
See unittest.TestCase.assertTupleEqual.

array_split.unittest.TestCase.assertWarns

TestCase.**assertWarns**(*expected_warning*, *callable_obj=None*, **args*, ***kwargs*)
Fail unless a warning of class warnClass is triggered by callable_obj when invoked with arguments args and keyword arguments kwargs. If a different type of warning is triggered, it will not be handled: depending on the other warning filtering rules in effect, it might be silenced, printed out, or raised as an exception.

If called with callable_obj omitted or None, will return a context object used like this:

```
with self.assertWarns(SomeWarning):
    do_something()
```

An optional keyword argument 'msg' can be provided when assertWarns is used as a context object.

The context manager keeps a reference to the first matching warning as the 'warning' attribute; similarly, the 'filename' and 'lineno' attributes give you information about the line of Python code from which the warning was triggered. This allows you to inspect the warning after the assertion:

```
with self.assertWarns(SomeWarning) as cm:
    do_something()
the_warning = cm.warning
self.assertEqual(the_warning.some_attribute, 147)
```

array_split.unittest.TestCase.assertWarnsRegex

TestCase.**assertWarnsRegex**(**args*, ***kwargs*)
See unittest.TestCase.assertWarnsRegex.

array_split.unittest.TestCase.assert

TestCase.**assert_**(**args*, ***kwargs*)

array_split.unittest.TestCase.countTestCases

TestCase.**countTestCases**()

array_split.unittest.TestCase.debug

TestCase.**debug**()
Run the test without collecting errors in a TestResult

array_split.unittest.TestCase.defaultTestResult

TestCase.**defaultTestResult**()

array_split.unittest.TestCase.doCleanups

TestCase.**doCleanups**()
Execute all cleanup functions. Normally called for you after tearDown.

array_split.unittest.TestCase.fail

TestCase.**fail**(*msg=None*)
Fail immediately, with the given message.

array_split.unittest.TestCase.failIf

TestCase.**failIf**(**args*, ***kwargs*)

array_split.unittest.TestCase.failIfAlmostEqual

TestCase.**failIfAlmostEqual**(**args*, ***kwargs*)

array_split.unittest.TestCase.failIfEqual

TestCase.**failIfEqual**(**args*, ***kwargs*)

array_split.unittest.TestCase.failUnless

TestCase.**failUnless**(**args*, ***kwargs*)

array_split.unittest.TestCase.failUnlessAlmostEqual

TestCase.**failUnlessAlmostEqual**(**args*, ***kwargs*)

array_split.unittest.TestCase.failUnlessEqual

TestCase.**failUnlessEqual**(**args*, ***kwargs*)

array_split.unittest.TestCase.failUnlessRaises

TestCase.**failUnlessRaises**(**args*, ***kwargs*)

array_split.unittest.TestCase.id

TestCase.**id**()

array_split.unittest.TestCase.run

TestCase.**run**(*result=None*)

array_split.unittest.TestCase.setUp

`TestCase.`**`setUp`**`()`
Hook method for setting up the test fixture before exercising it.

array_split.unittest.TestCase.setUpClass

`TestCase.`**`setUpClass`**`()`
Hook method for setting up class fixture before running tests in the class.

array_split.unittest.TestCase.shortDescription

`TestCase.`**`shortDescription`**`()`
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

array_split.unittest.TestCase.skipTest

`TestCase.`**`skipTest`**`(`*reason*`)`
Skip this test.

array_split.unittest.TestCase.subTest

`TestCase.`**`subTest`**`(`*msg=None*, ***params*`)`
Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters. A failure in the subtest marks the test case as failed but resumes execution at the end of the enclosed block, allowing further test code to be executed.

array_split.unittest.TestCase.tearDown

`TestCase.`**`tearDown`**`()`
Hook method for deconstructing the test fixture after testing it.

array_split.unittest.TestCase.tearDownClass

`TestCase.`**`tearDownClass`**`()`
Hook method for deconstructing the class fixture after running all tests in the class.

Attributes

longMessage
maxDiff

array_split.unittest.TestCase.longMessage

TestCase.**longMessage** = True

array_split.unittest.TestCase.maxDiff

TestCase.**maxDiff** = 640

3.7 The array_split.license Module

License and copyright info.

3.7.1 License

Copyright (C) 2016 The Australian National University.

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.

3.7.2 Copyright

Copyright (C) 2016 The Australian National University.

3.7.3 Functions

license()	Returns the *array_split* license string.
copyright()	Returns the *array_split* copyright string.

array_split.license.license

array_split.license.**license**()

Returns the *array_split* license string.

Return type str

Returns License string.

array_split.license.copyright

array_split.license.**copyright**()
Returns the *array_split* copyright string.

Return type str

Returns Copyright string.

a

array_split, 17
array_split.license, 69
array_split.logging, 53
array_split.split, 27
array_split.split_test, 39
array_split.tests, 53
array_split.unittest, 57

Symbols

__init__() (array_split.ShapeSplitter method), 20
__init__() (array_split.logging.SplitStreamHandler method), 54
__init__() (array_split.split.ShapeSplitter method), 31
__init__() (array_split.split_test.SplitTest method), 42
__init__() (array_split.unittest.TestCase method), 60

A

acquire() (array_split.logging.SplitStreamHandler method), 55
addCleanup() (array_split.split_test.SplitTest method), 43
addCleanup() (array_split.unittest.TestCase method), 60
addFilter() (array_split.logging.SplitStreamHandler method), 55
addTypeEqualityFunc() (array_split.split_test.SplitTest method), 43
addTypeEqualityFunc() (array_split.unittest.TestCase method), 60
ARRAY_BOUNDS (in module array_split), 27
ARRAY_BOUNDS (in module array_split.split), 39
array_itemsize (array_split.ShapeSplitter attribute), 25
array_itemsize (array_split.split.ShapeSplitter attribute), 35
array_shape (array_split.ShapeSplitter attribute), 25
array_shape (array_split.split.ShapeSplitter attribute), 35
array_split (module), 17
array_split() (in module array_split), 19
array_split() (in module array_split.split), 38
array_split.license (module), 69
array_split.logging (module), 53
array_split.split (module), 27
array_split.split_test (module), 39
array_split.tests (module), 53
array_split.unittest (module), 57
array_start (array_split.ShapeSplitter attribute), 25
array_start (array_split.split.ShapeSplitter attribute), 35
assert_() (array_split.split_test.SplitTest method), 49
assert_() (array_split.unittest.TestCase method), 66
assertAlmostEqual() (array_split.split_test.SplitTest method), 43
assertAlmostEqual() (array_split.unittest.TestCase method), 60
assertAlmostEquals() (array_split.split_test.SplitTest method), 43
assertAlmostEquals() (array_split.unittest.TestCase method), 61
assertArraySplitEqual() (array_split.split_test.SplitTest method), 43
assertArraySplitEqual() (array_split.unittest.TestCase method), 61
assertCountEqual() (array_split.split_test.SplitTest method), 44
assertCountEqual() (array_split.unittest.TestCase method), 61
assertDictContainsSubset() (array_split.split_test.SplitTest method), 44
assertDictContainsSubset() (array_split.unittest.TestCase method), 61
assertDictEqual() (array_split.split_test.SplitTest method), 44
assertDictEqual() (array_split.unittest.TestCase method), 61
assertEqual() (array_split.split_test.SplitTest method), 44
assertEqual() (array_split.unittest.TestCase method), 61
assertEquals() (array_split.split_test.SplitTest method), 44
assertEquals() (array_split.unittest.TestCase method), 62
assertFalse() (array_split.split_test.SplitTest method), 44
assertFalse() (array_split.unittest.TestCase method), 62
assertGreater() (array_split.split_test.SplitTest method), 44
assertGreater() (array_split.unittest.TestCase method), 62
assertGreaterEqual() (array_split.split_test.SplitTest method), 44
assertGreaterEqual() (array_split.unittest.TestCase method), 62
assertIn() (array_split.split_test.SplitTest method), 45
assertIn() (array_split.unittest.TestCase method), 62
assertIs() (array_split.split_test.SplitTest method), 45

assertIs() (array_split.unittest.TestCase method), 62
assertIsInstance() (array_split.split_test.SplitTest method), 45
assertIsInstance() (array_split.unittest.TestCase method), 62
assertIsNone() (array_split.split_test.SplitTest method), 45
assertIsNone() (array_split.unittest.TestCase method), 62
assertIsNot() (array_split.split_test.SplitTest method), 45
assertIsNot() (array_split.unittest.TestCase method), 62
assertIsNotNone() (array_split.split_test.SplitTest method), 45
assertIsNotNone() (array_split.unittest.TestCase method), 63
assertItemsEqual() (array_split.split_test.SplitTest method), 45
assertItemsEqual() (array_split.unittest.TestCase method), 63
assertLess() (array_split.split_test.SplitTest method), 45
assertLess() (array_split.unittest.TestCase method), 63
assertLessEqual() (array_split.split_test.SplitTest method), 45
assertLessEqual() (array_split.unittest.TestCase method), 63
assertListEqual() (array_split.split_test.SplitTest method), 46
assertListEqual() (array_split.unittest.TestCase method), 63
assertLogs() (array_split.split_test.SplitTest method), 46
assertLogs() (array_split.unittest.TestCase method), 63
assertMultiLineEqual() (array_split.split_test.SplitTest method), 46
assertMultiLineEqual() (array_split.unittest.TestCase method), 63
assertNotAlmostEqual() (array_split.split_test.SplitTest method), 46
assertNotAlmostEqual() (array_split.unittest.TestCase method), 64
assertNotAlmostEquals() (array_split.split_test.SplitTest method), 46
assertNotAlmostEquals() (array_split.unittest.TestCase method), 64
assertNotEqual() (array_split.split_test.SplitTest method), 46
assertNotEqual() (array_split.unittest.TestCase method), 64
assertNotEquals() (array_split.split_test.SplitTest method), 47
assertNotEquals() (array_split.unittest.TestCase method), 64
assertNotIn() (array_split.split_test.SplitTest method), 47
assertNotIn() (array_split.unittest.TestCase method), 64
assertNotIsInstance() (array_split.split_test.SplitTest method), 47
assertNotIsInstance() (array_split.unittest.TestCase method), 64
assertNotRegex() (array_split.split_test.SplitTest method), 47
assertNotRegex() (array_split.unittest.TestCase method), 64
assertRaises() (array_split.split_test.SplitTest method), 47
assertRaises() (array_split.unittest.TestCase method), 64
assertRaisesRegex() (array_split.split_test.SplitTest method), 47
assertRaisesRegex() (array_split.unittest.TestCase method), 65
assertRaisesRegexp() (array_split.split_test.SplitTest method), 47
assertRaisesRegexp() (array_split.unittest.TestCase method), 65
assertRegex() (array_split.split_test.SplitTest method), 48
assertRegex() (array_split.unittest.TestCase method), 65
assertRegexpMatches() (array_split.split_test.SplitTest method), 48
assertRegexpMatches() (array_split.unittest.TestCase method), 65
assertSequenceEqual() (array_split.split_test.SplitTest method), 48
assertSequenceEqual() (array_split.unittest.TestCase method), 65
assertSetEqual() (array_split.split_test.SplitTest method), 48
assertSetEqual() (array_split.unittest.TestCase method), 65
assertTrue() (array_split.split_test.SplitTest method), 48
assertTrue() (array_split.unittest.TestCase method), 65
assertTupleEqual() (array_split.split_test.SplitTest method), 48
assertTupleEqual() (array_split.unittest.TestCase method), 66
assertWarns() (array_split.split_test.SplitTest method), 48
assertWarns() (array_split.unittest.TestCase method), 66
assertWarnsRegex() (array_split.split_test.SplitTest method), 49
assertWarnsRegex() (array_split.unittest.TestCase method), 66

C

calculate_axis_split_extents() (array_split.ShapeSplitter method), 21
calculate_axis_split_extents() (array_split.split.ShapeSplitter method), 32
calculate_num_slices_per_axis() (in module array_split.split), 28
calculate_split() (array_split.ShapeSplitter method), 22
calculate_split() (array_split.split.ShapeSplitter method), 32

calculate_split_by_indices_per_axis() (array_split.ShapeSplitter method), 22
calculate_split_by_indices_per_axis() (array_split.split.ShapeSplitter method), 32
calculate_split_by_split_size() (array_split.ShapeSplitter method), 22
calculate_split_by_split_size() (array_split.split.ShapeSplitter method), 32
calculate_split_by_tile_max_bytes() (array_split.ShapeSplitter method), 22
calculate_split_by_tile_max_bytes() (array_split.split.ShapeSplitter method), 32
calculate_split_by_tile_shape() (array_split.ShapeSplitter method), 22
calculate_split_by_tile_shape() (array_split.split.ShapeSplitter method), 33
calculate_split_from_extents() (array_split.ShapeSplitter method), 23
calculate_split_from_extents() (array_split.split.ShapeSplitter method), 33
calculate_tile_shape_for_max_bytes() (in module array_split.split), 29
check_consistent_parameter_dimensions() (array_split.ShapeSplitter method), 23
check_consistent_parameter_dimensions() (array_split.split.ShapeSplitter method), 33
check_consistent_parameter_grouping() (array_split.ShapeSplitter method), 23
check_consistent_parameter_grouping() (array_split.split.ShapeSplitter method), 33
check_halo() (array_split.ShapeSplitter method), 23
check_halo() (array_split.split.ShapeSplitter method), 33
check_split_parameters() (array_split.ShapeSplitter method), 23
check_split_parameters() (array_split.split.ShapeSplitter method), 33
check_tile_bounds_policy() (array_split.ShapeSplitter method), 23
check_tile_bounds_policy() (array_split.split.ShapeSplitter method), 34
close() (array_split.logging.SplitStreamHandler method), 55
copyright() (in module array_split.license), 70
countTestCases() (array_split.split_test.SplitTest method), 49
countTestCases() (array_split.unittest.TestCase method), 66
createLock() (array_split.logging.SplitStreamHandler method), 55

D

debug() (array_split.split_test.SplitTest method), 49
debug() (array_split.unittest.TestCase method), 66
defaultTestResult() (array_split.split_test.SplitTest method), 49
defaultTestResult() (array_split.unittest.TestCase method), 66
doCleanups() (array_split.split_test.SplitTest method), 49
doCleanups() (array_split.unittest.TestCase method), 67

E

emit() (array_split.logging.SplitStreamHandler method), 55

F

fail() (array_split.split_test.SplitTest method), 49
fail() (array_split.unittest.TestCase method), 67
failIf() (array_split.split_test.SplitTest method), 49
failIf() (array_split.unittest.TestCase method), 67
failIfAlmostEqual() (array_split.split_test.SplitTest method), 49
failIfAlmostEqual() (array_split.unittest.TestCase method), 67
failIfEqual() (array_split.split_test.SplitTest method), 50
failIfEqual() (array_split.unittest.TestCase method), 67
failUnless() (array_split.split_test.SplitTest method), 50
failUnless() (array_split.unittest.TestCase method), 67
failUnlessAlmostEqual() (array_split.split_test.SplitTest method), 50
failUnlessAlmostEqual() (array_split.unittest.TestCase method), 67
failUnlessEqual() (array_split.split_test.SplitTest method), 50
failUnlessEqual() (array_split.unittest.TestCase method), 67
failUnlessRaises() (array_split.split_test.SplitTest method), 50
failUnlessRaises() (array_split.unittest.TestCase method), 67
filter() (array_split.logging.SplitStreamHandler method), 55
flush() (array_split.logging.SplitStreamHandler method), 55
format() (array_split.logging.SplitStreamHandler method), 56

G

get_formatter() (in module array_split.logging), 57
get_name() (array_split.logging.SplitStreamHandler method), 56

H

halo (array_split.ShapeSplitter attribute), 25
halo (array_split.split.ShapeSplitter attribute), 35
handle() (array_split.logging.SplitStreamHandler method), 56

handleError() (array_split.logging.SplitStreamHandler method), 56

I

id() (array_split.split_test.SplitTest method), 50
id() (array_split.unittest.TestCase method), 67
indices_per_axis (array_split.ShapeSplitter attribute), 25
indices_per_axis (array_split.split.ShapeSplitter attribute), 35
initialise_loggers() (in module array_split.logging), 57

L

license() (in module array_split.license), 69
logger (array_split.ShapeSplitter attribute), 25
logger (array_split.split.ShapeSplitter attribute), 36
logger (array_split.split_test.SplitTest attribute), 53
longMessage (array_split.split_test.SplitTest attribute), 53
longMessage (array_split.unittest.TestCase attribute), 69

M

main() (in module array_split.unittest), 58
max_tile_bytes (array_split.ShapeSplitter attribute), 25
max_tile_bytes (array_split.split.ShapeSplitter attribute), 36
max_tile_shape (array_split.ShapeSplitter attribute), 26
max_tile_shape (array_split.split.ShapeSplitter attribute), 36
maxDiff (array_split.split_test.SplitTest attribute), 53
maxDiff (array_split.unittest.TestCase attribute), 69

N

name (array_split.logging.SplitStreamHandler attribute), 57
NO_BOUNDS (in module array_split), 27
NO_BOUNDS (in module array_split.split), 39

R

release() (array_split.logging.SplitStreamHandler method), 56
removeFilter() (array_split.logging.SplitStreamHandler method), 56
run() (array_split.split_test.SplitTest method), 50
run() (array_split.unittest.TestCase method), 67

S

set_name() (array_split.logging.SplitStreamHandler method), 57
set_split_extents() (array_split.ShapeSplitter method), 23
set_split_extents() (array_split.split.ShapeSplitter method), 34
set_split_extents_by_indices_per_axis() (array_split.ShapeSplitter method), 24
set_split_extents_by_indices_per_axis() (array_split.split.ShapeSplitter method), 34
set_split_extents_by_split_size() (array_split.ShapeSplitter method), 24
set_split_extents_by_split_size() (array_split.split.ShapeSplitter method), 34
set_split_extents_by_tile_max_bytes() (array_split.ShapeSplitter method), 24
set_split_extents_by_tile_max_bytes() (array_split.split.ShapeSplitter method), 34
set_split_extents_by_tile_shape() (array_split.ShapeSplitter method), 24
set_split_extents_by_tile_shape() (array_split.split.ShapeSplitter method), 34
setFormatter() (array_split.logging.SplitStreamHandler method), 56
setLevel() (array_split.logging.SplitStreamHandler method), 57
setUp() (array_split.split_test.SplitTest method), 50
setUp() (array_split.unittest.TestCase method), 68
setUpClass() (array_split.split_test.SplitTest method), 50
setUpClass() (array_split.unittest.TestCase method), 68
shape_factors() (in module array_split.split), 28
shape_split() (in module array_split), 18
shape_split() (in module array_split.split), 37
ShapeSplitter (class in array_split), 19
ShapeSplitter (class in array_split.split), 30
shortDescription() (array_split.split_test.SplitTest method), 50
shortDescription() (array_split.unittest.TestCase method), 68
skipTest() (array_split.split_test.SplitTest method), 51
skipTest() (array_split.unittest.TestCase method), 68
split_begs (array_split.ShapeSplitter attribute), 26
split_begs (array_split.split.ShapeSplitter attribute), 36
split_ends (array_split.ShapeSplitter attribute), 26
split_ends (array_split.split.ShapeSplitter attribute), 36
split_num_slices_per_axis (array_split.ShapeSplitter attribute), 26
split_num_slices_per_axis (array_split.split.ShapeSplitter attribute), 36
split_shape (array_split.ShapeSplitter attribute), 26
split_shape (array_split.split.ShapeSplitter attribute), 36
split_size (array_split.ShapeSplitter attribute), 26
split_size (array_split.split.ShapeSplitter attribute), 36
SplitStreamHandler (class in array_split.logging), 54
SplitTest (class in array_split.split_test), 41
sub_tile_shape (array_split.ShapeSplitter attribute), 26
sub_tile_shape (array_split.split.ShapeSplitter attribute), 37
subTest() (array_split.split_test.SplitTest method), 51
subTest() (array_split.unittest.TestCase method), 68

T

tearDown() (array_split.split_test.SplitTest method), 51
tearDown() (array_split.unittest.TestCase method), 68
tearDownClass() (array_split.split_test.SplitTest method), 51
tearDownClass() (array_split.unittest.TestCase method), 68
terminator (array_split.logging.SplitStreamHandler attribute), 57
test_array_split() (array_split.split_test.SplitTest method), 51
test_calculate_num_slices_per_axis() (array_split.split_test.SplitTest method), 51
test_calculate_split_by_tile_max_bytes_1d() (array_split.split_test.SplitTest method), 51
test_calculate_split_by_tile_shape_1d() (array_split.split_test.SplitTest method), 51
test_calculate_split_by_tile_shape_2d() (array_split.split_test.SplitTest method), 51
test_calculate_split_with_array_start_1d() (array_split.split_test.SplitTest method), 52
test_calculate_split_with_array_start_2d() (array_split.split_test.SplitTest method), 52
test_calculate_split_with_halo_1d() (array_split.split_test.SplitTest method), 52
test_calculate_split_with_halo_2d() (array_split.split_test.SplitTest method), 52
test_calculate_tile_shape_for_max_bytes_1d() (array_split.split_test.SplitTest method), 52
test_calculate_tile_shape_for_max_bytes_2d() (array_split.split_test.SplitTest method), 52
test_shape_factors() (array_split.split_test.SplitTest method), 52
test_split_by_num_slices() (array_split.split_test.SplitTest method), 52
test_split_by_per_axis_indices() (array_split.split_test.SplitTest method), 52
TestCase (class in array_split.unittest), 58
tile_beg_min (array_split.ShapeSplitter attribute), 26
tile_beg_min (array_split.split.ShapeSplitter attribute), 37
tile_bounds_policy (array_split.ShapeSplitter attribute), 27
tile_bounds_policy (array_split.split.ShapeSplitter attribute), 37
tile_end_max (array_split.ShapeSplitter attribute), 27
tile_end_max (array_split.split.ShapeSplitter attribute), 37
tile_shape (array_split.ShapeSplitter attribute), 27
tile_shape (array_split.split.ShapeSplitter attribute), 37

U

update_tile_extent_bounds() (array_split.ShapeSplitter method), 24
update_tile_extent_bounds() (array_split.split.ShapeSplitter method), 34

V

valid_tile_bounds_policies (array_split.ShapeSplitter attribute), 27
valid_tile_bounds_policies (array_split.split.ShapeSplitter attribute), 37