lcmap-merlin Documentation

Release 2.2.0

USGS EROS LCMAP

Sep 25, 2018

Contents

A Python3 library for turning LCMAP spatial data into timeseries like magic.

CHAPTER 1

Features

- Retrieve chips & chip specs
- Convert chips & chip specs into time series rods
- Many composable functions
- Works with symmetric or assymetric data arrays
- Built with efficiency in mind… leverages Numpy for heavy lifting.
- Tested with cPython 3.5 & 3.6

CHAPTER 2

Example

```
import merlin

timeseries = merlin.create(x=123,
                           y=456,
                           acquired='1980-01-01/2017-01-01',
                           cfg=merlin.cfg.get(profile='chipmunk-ard',
                                              env={'CHIPMUNK_URL': 'http://
↪localhost:5656'}))

print(timeseries)

(((123, 456, 123, 456), {'reds'  : [9, 8, ...],
                         'greens': [99, 88, ...]},
                         'blues' : [12, 22, ...],
                         'dates':  ['2017-01-01', '2016-12-31', ...]}),
 ((123, 456, 124, 456), {'reds'  : [4, 3, ...],
                         'greens': [19, 8, ...]},
                         'blues' : [1, 11, ...],
                         'dates':  ['2017-01-01', '2016-12-31', ...]}),)
```

CHAPTER 3

Documentation

Complete documentation is available at http://lcmap-merlin.readthedocs.io/

CHAPTER 4

Installation

```
pip install lcmap-merlin
```

CHAPTER 5

Versioning

Merlin follows semantic versioning: http://semver.org/

CHAPTER 6

License

This is free and unencumbered software released into the public domain.

Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.

In jurisdictions that recognize copyright laws, the author or authors of this software dedicate any and all copyright interest in the software to the public domain. We make this dedication for the benefit of the public at large and to the detriment of our heirs and successors. We intend this dedication to be an overt act of relinquishment in perpetuity of all present and future rights to this software under copyright law.

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 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.

For more information, please refer to http://unlicense.org.

6.1 Configuration

Environment Variables	
Variable	Default
CHIPMUNK_URL	None
CHIPMUNK_GRID_RESOURCE	/grid
CHIPMUNK_SNAP_RESOURCE	/grid/snap
CHIPMUNK_NEAR_RESOURCE	/grid/near
CHIPMUNK_CHIPS_RESOURCE	/chips
CHIPMUNK_REGISTRY_RESOURCE	/registry

6.2 Cookbook

6.2.1 Configure Merlin For Chipmunk

Merlin is configurable with environment variables and parameters. Parameters override environment variables.

```
import merlin
import os

# export CHIPMUNK_URL=http://localhost:5656/plus/path
os.environ['CHIPMUNK_URL'] = 'http://localhost:5656/plus/path'

merlin.cfg.get(profile='chipmunk-ard')
```

```
import merlin

merlin.cfg.get(profile='chipmunk-ard',
               env={'CHIPMUNK_URL': 'http://localhost:5656/plus/path'})
```

6.2.2 View All Configuration Profiles

Merlin configurations are organized into profiles, which group function implementations by use. All profiles may be viewed by calling `merlin.cfg.profiles` with a `None` parameter.

```
import merlin

merlin.cfg.profiles(None)
```

6.2.3 Create Timeseries

```
import merlin

timeseries = merlin.create(x=123,
                           y=456,
                           acquired='1980-01-01/2017-01-01',
                           cfg=merlin.cfg.get('chipmunk-ard'))

print(timeseries)

(((123, 456, 123, 456), {'reds'  : [9, 8, ...],
                         'greens': [99, 88, ...]},
                         'blues' : [12, 22, ...],
                         'dates':  ['2017-01-01', '2016-12-31', ...]}),
 ((123, 456, 124, 456), {'reds'  : [4, 3, ...],
                         'greens': [19, 8, ...]},
                         'blues' : [1, 11, ...],
                         'dates':  ['2017-01-01', '2016-12-31', ...]}),)
```

6.2.4 Retrieve Chips

```
import merlin

fn = merlin.cfg.get('chipmunk-ard').get('chips_fn')

fn(x=123, y=456, acquired='1980/2017', ubids=['LC08_SRB4', 'LE07_SRB3', ...])
```

6.2.5 Retrieve Specs

```
import merlin

fn = merlin.cfg.get('chipmunk-ard').get('registry_fn')

fn()
```

6.2.6 Retrieve Specs Mapped To UBIDS

```
import merlin

registry = merlin.cfg.get('chipmunk-ard').get('registry_fn')

merlin.specs.mapped(specs=registry(),
                    ubids=merlin.cfg.ubids.get('chipmunk-ard'))
```

6.2.7 Snap A Point To A Grid

```
import merlin

fn = merlin.cfg.get('chipmunk-ard').get('snap_fn')

fn(x=123, y=456)
```

6.3 Develop

6.3.1 Get The Source

```
$ git clone git@github.com:usgs-eros/lcmap-merlin

# Highly recommend working within a virtual environment
$ conda create --name merlin python=3.6
$ source activate merlin
$ cd lcmap-merlin
$ pip install -e .[test, dev, doc]
```

6.3.2 Testing

```
$ pytest
```

Occasionally test data may need to be updated if source data changes. Merlin uses the vcrpy library to save HTTP requests for replay during testing.

VCR cassettes are configured in `test/__init__.py`. To support multiple data source versions, add new attributes to the module that correspond to the data source or data set version to be tested against.

```
import vcr as _vcr

# chipmunk parameters for test data
x = -2094585
y = 1952805
x_nodata = -183585.0
y_nodata = 302805.0
acquired = '2010/2013'
profile = 'chipmunk-ard'
env = {'CHIPMUNK_URL': 'http://localhost:5656'}
ard-c1-v1-cassette = 'test/resources/chipmunk-ard-c1-v1-cassette.yaml'
ard-c1-v2-cassette = 'test/resources/chipmunk-ard-c1-v2-cassette.yaml'
aux-c1-v1-cassette = 'test/resources/chipmunk-aux-c1-v1-cassette.yaml'
vcr = _vcr.VCR(record_mode='new_episodes')
```

Apply the proper decorator value to each function for the version under test.

```
import test

@test.vcr.use_cassette(test.ard-c1-v1-cassette)
def test_some_func_c1_v1():
    assert awesomeness

@test.vcr.use_cassette(test.ard-c1-v2-cassette)
def test_some_func_c1_v2():
    assert awesomeness_v2
```

Caution: When expanding the `acquired` date range, keep in mind that PyPi has a limit of 60MB per artifact. Uploads exceeding this limit will result in failure messages while publishing.

6.3.3 Build Sphinx Docs

All Merlin docs are written in reStructuredText. All code comments are written using Google Docstrings.

Installing Sphinx and building the docs are only necessary during development. Release documents are built automatically by readthedocs.io.

First, make sure you've installed Sphinx:

```
$ pip install -e .[doc]
```

Automatically rebuild documentation and refresh the web browser when source files change:

```
$ make autobuild
```

Manually build documentation one time:

```
$ cd docs
$ make html
```

6.4 API Reference

6.4.1 merlin.cfg

merlin.cfg.**get**(*profile='chipmunk-ard'*, *env=None*)
Return a configuration profile.

Parameters

- **profile** (*str*) – Name of profile.
- **env** (*dict*) – Environment variables to override os.environ

Returns A Merlin configuration

Return type dict

merlin.cfg.**profiles**(*env*, *profile=None*)
Retrieve a configuration profile with env applied.

Parameters

- **env** (*dict*) – Environment variables
- **profile** (*str*) – Name of profile to load. If no profile is supplied all profiles are returned.

Returns Profile or profiles with env substitutions.

Return type dict

6.4.2 merlin.chipmunk

Chipmunk.py is the interface module to Chipmunk for Merlin.

Any functions chipmunk exposes are represented here as defined.

New abstractions or higher level abstractions should be created in modules that import chipmunk.py. For maximum flexibility do not import chipmunk.py directly... inject it via a kernel or DI construct.

Multi-version support: To support multiple versions of Chipmunk create new modules that correspond to the appropriate version number.

merlin.chipmunk.**chips**(*x*, *y*, *acquired*, *ubids*, *url*, *resource='/chips'*)
Returns chips from a Chipmunk url given x, y, date range and ubid sequence

Parameters

- **x** (*int*) – projection coordinate x
- **y** (*int*) – projection coordinate y
- **acquired** (*str*) – ISO8601 daterange '2012-01-01/2014-01-03'
- **ubids** (*sequence*) – sequence of ubids
- **url** (*str*) – protocol://host:port/path

- **resource** (*str*) – /chips/resource/path (default: /chips)

Returns chips

Return type tuple

Example

```
>>> chipmunk.chips(url='http://host:port/path',
                   x=123456,
                   y=789456,
                   acquired='2012-01-01/2014-01-03',
                   ubids=['LE07_SRB1', 'LT05_SRB1'])
(LE07_SRB1_DATE1, LT05_SRB1_DATE2, LE07_SRB1_DATE2, ...)
```

merlin.chipmunk.**grid**(*url*, *resource='/grid'*)

Return grid definitions.

Parameters

- **url** (*str*) – protocol://host:port/path
- **resource** (*str*) – /the/grid/resource (default: /grid)

Returns dict

Example

```
>>> chipmunk.grid(url='http://host:port/path)
[{"name":"tile",
  "proj":null,
  "rx":1.0,
  "ry":-1.0,
  "sx":150000.0,
  "sy":150000.0,
  "tx":2565585.0,
  "ty":3314805.0},
 {"name":"chip",
  "proj":null,
  "rx":1.0,
  "ry":-1.0,
  "sx":3000.0,
  "sy":3000.0,
  "tx":2565585.0,
  "ty":3314805.0}]
```

merlin.chipmunk.**near**(*x*, *y*, *url*, *resource='/grid/near'*)

Determines chips and tiles that lie a point

Parameters

- **x** (*int*) – projection coordinate x
- **y** (*int*) – projection coordinate y
- **url** (*str*) – protocol://host:port/path
- **resource** (*str*) – /grid/near/resource (default: /grid/near)

Returns dict

Example

```
>>> chipmunk.near(x=0, y=0, url='http://host:port/path')
{'chip': [{'grid-pt': [854.0, 1105.0], 'proj-pt': [-3585.0, -195.0]},
          {'grid-pt': [854.0, 1104.0], 'proj-pt': [-3585.0, 2805.0]},
          {'grid-pt': [854.0, 1103.0], 'proj-pt': [-3585.0, 5805.0]},
          {'grid-pt': [855.0, 1105.0], 'proj-pt': [-585.0, -195.0]},
          {'grid-pt': [855.0, 1104.0], 'proj-pt': [-585.0, 2805.0]},
          {'grid-pt': [855.0, 1103.0], 'proj-pt': [-585.0, 5805.0]},
          {'grid-pt': [856.0, 1105.0], 'proj-pt': [2415.0, -195.0]},
          {'grid-pt': [856.0, 1104.0], 'proj-pt': [2415.0, 2805.0]},
          {'grid-pt': [856.0, 1103.0], 'proj-pt': [2415.0, 5805.0]}],
 'tile': [{'grid-pt': [16.0, 23.0], 'proj-pt': [-165585.0, -135195.0]},
          {'grid-pt': [16.0, 22.0], 'proj-pt': [-165585.0, 14805.0]},
          {'grid-pt': [16.0, 21.0], 'proj-pt': [-165585.0, 164805.0]},
          {'grid-pt': [17.0, 23.0], 'proj-pt': [-15585.0, -135195.0]},
          {'grid-pt': [17.0, 22.0], 'proj-pt': [-15585.0, 14805.0]},
          {'grid-pt': [17.0, 21.0], 'proj-pt': [-15585.0, 164805.0]},
          {'grid-pt': [18.0, 23.0], 'proj-pt': [134415.0, -135195.0]},
          {'grid-pt': [18.0, 22.0], 'proj-pt': [134415.0, 14805.0]},
          {'grid-pt': [18.0, 21.0], 'proj-pt': [134415.0, 164805.0]}]}
```

merlin.chipmunk.**registry**(*url*, *resource='/registry'*)

Retrieve the chip spec registry

Parameters

- **url** (*str*) – protocol://host:port/path
- **resource** (*str*) – /registry/resource/path (default: /registry)

Returns list

Example

```
>>> chipmunk.registry(url='http://host:port/path')
 [{'data_fill': '-9999',
   'data_mask': {},
   'data_range': [],
   'data_scale': None,
   'data_shape': [100, 100],
   'data_type': 'INT16',
   'data_units': None,
   'info': 'band 5 top-of-atmosphere reflectance',
   'tags': ['swir1', 'b5', 'tab5', 'lt05', 'lt05_tab5', 'ta'],
   'ubid': 'LT05_TAB5'},
  {'data_fill': '-9999',
   'data_mask': {},
   'data_range': [],
   'data_scale': None,
   'data_shape': [100, 100],
   'data_type': 'INT16',
   'data_units': None,
   'info': 'band 7 top-of-atmosphere reflectance',
```

(continues on next page)

(continued from previous page)

```
   'tags': ['lt05_tab7', 'b7', 'lt05', 'swir2', 'tab7', 'ta'],
   'ubid': 'LT05_TAB7'}, ...]
```

merlin.chipmunk.**snap**(*x*, *y*, *url*, *resource='/grid/snap'*)

Determine the chip and tile coordinates for a point.

Parameters

- **x** (*int*) – projection coordinate x
- **y** (*int*) – projection coordinate y
- **url** (*str*) – protocol://host:port/path
- **resource** (*str*) – /grid/snap/resource (default: /grid/snap)

Returns dict

Example

```
>>> chipmunk.snap(x=0, y=0, url='http://host:port/path')
 {'chip': {'grid-pt': [855.0, 1104.0], 'proj-pt': [-585.0, 2805.0]},
  'tile': {'grid-pt': [17.0, 22.0], 'proj-pt': [-15585.0, 14805.0]}}
```

6.4.3 merlin.chips

merlin.chips.**chip_to_numpy**(*chip*, *chip_spec*)

Removes base64 encoding of chip data and converts it to a numpy array

Parameters

- **chip** – A chip
- **chip_spec** – Corresponding chip_spec

Returns a decoded chip with data as a shaped numpy array

merlin.chips.**dates**(*chips*)

Dates for a sequence of chips

Parameters **chips** – sequence of chips

Returns datestrings

Return type tuple

merlin.chips.**deduplicate**(*chips*)

Accepts a sequence of chips and returns a sequence of chips minus any duplicates. A chip is considered a duplicate if it shares an x, y, UBID and acquired date with another chip.

Parameters **chips** (*sequence*) – Sequence of chips

Returns A nonduplicated tuple of chips

Return type tuple

merlin.chips.**identity**(*chip*)

Determine the identity of a chip.

Parameters **chip** (*dict*) – A chip

Returns Tuple of the chip identity field

Return type tuple

merlin.chips.**locations**(*x*, *y*, *cw*, *ch*, *rx*, *ry*, *sx*, *sy*)

Computes locations for array elements that fall within the shape specified by chip_spec['data_shape'] using the x & y as the origin. locations() does not snap() x & y... this should be done prior to calling locations() if needed.

Parameters

- **x** – x coordinate
- **y** – y coordinate
- **cw** – chip width in pixels (e.g. 100 pixels)
- **ch** – chip height in pixels (e.g. 100 pixels)
- **rx** – x reflection (e.g. 1)
- **ry** – y reflection (e.g. -1)
- **sx** – x scale (e.g. 3000 meters)
- **sy** – y scale (e.g. 3000 meters)

Returns a two (three) dimensional numpy array of [x, y] coordinates

merlin.chips.**mapped**(*x*, *y*, *acquired*, *specmap*, *chips_fn*)

Maps chips_fn results to keys from specmap.

Parameters

- **x** (*int*) – x coordinate
- **y** (*int*) – y coordinate
- **acquired** (*str*) – iso8601 date range
- **specmap** (*dict*) – map of specs
- **chips_fn** (*fn*) – function to return chips

Returns {k: chips_fn()}

Return type dict

merlin.chips.**rsort**(*chips*, *key=<function <lambda>>*)

Reverse sorts a sequence of chips by date.

Parameters **chips** – sequence of chips

Returns sorted sequence of chips

merlin.chips.**to_numpy**(*chips*, *spec_index*)

Converts the data for a sequence of chips to numpy arrays

Parameters

- **chips** (*sequence*) – a sequence of chips
- **spec_index** (*dict*) – chip_specs keyed by ubid

Returns chips with data as numpy arrays

Return type sequence

merlin.chips.**trim**(*chips*, *dates*)

Eliminates chips that are not from the specified dates

Parameters

- **chips** – Sequence of chips
- **dates** – Sequence of dates that should be included in result

Returns filtered chips

Return type tuple

6.4.4 merlin.dates

merlin.dates.**enddate**(*acquired*)
Returns the enddate from an acquired date string

Parameters **acquired** (*str*) – / separated date range in iso8601 format

Returns End date

Return type str

merlin.dates.**is_acquired**(*acquired*)
Is the date string a / separated date range in iso8601 format?

Parameters **acquired** – A date string

Returns True or False

Return type bool

merlin.dates.**mapped**(*chipmap*)
Transform a dict of chips into a dict of datestrings

Parameters **chipmap** (*dict*) – {k: [chips]}

Returns {k: [datestring2, datestring1, datestring3]}

Return type dict

merlin.dates.**rsort**(*dateseq*)
Reverse sorts a sequence of dates.

Parameters **dateseq** – sequence of dates

Returns reverse sorted sequence of dates

Return type sequence

merlin.dates.**startdate**(*acquired*)
Returns the startdate from an acquired date string

Parameters **acquired** (*str*) – / separated date range in iso8601 format

Returns Start date

Return type str

merlin.dates.**symmetric**(*datemap*)
Returns a sequence of dates that are common to all map values if all datemap values are represented, else Exception.

Parameters **datemap** – {key: [datestrings,]}

Returns Sequence of date strings or Exception

Example

```
>>> common({"reds":  [ds3, ds1, ds2],
            "blues": [ds2, ds3, ds1]})
[2, 3, 1]
>>>
>>> common({"reds":  [ds3, ds1],
            "blues": [ds2, ds3, ds1]})
Exception: reds:[3, 1] does not match blues:[2, 3, 1]
```

merlin.dates.**to_ordinal**(*datestring*)
Extract an ordinal date from a date string

Parameters **datestring** (*str*) – date value

Returns ordinal date

Return type int

6.4.5 merlin.files

Functions for working with files in python

merlin.files.**append**(*path*, *data*)
Append data to a text file.

Parameters

- **path** (*str*) – Full path to file
- **data** (*str*) – File text

Returns Number of characters appended

Return type int

merlin.files.**appendb**(*path*, *data*)
Write data to a binary file.

Parameters

- **path** (*str*) – Full path to file
- **data** (*str*) – File bytes

Returns Number of bytes appended

Return type int

merlin.files.**delete**(*path*)
Delete a file.

Parameters **path** (*str*) – Full path to file

Returns True if the file was deleted, False if not (with message logged)

Return type bool

merlin.files.**exists**(*path*)
Determine if a file exists.

Parameters **path** (*str*) – Full path to file

Returns True if the file exists, False if not

Return type bool

merlin.files.**mkdirs**(*filename*)
Ensures the set of directories exist for the supplied filename.

Parameters **filename** (*str*) – Full path to where the file should exist

Returns Full file path if the directories were created or None

merlin.files.**read**(*path*)
Read a text file.

Parameters **path** (*str*) – Full path to file

Returns File text

Return type str

merlin.files.**readb**(*path*)
Read a binary file.

Parameters **path** (*str*) – Full path to file

Returns File bytes

merlin.files.**readlines**(*path*)
Read lines from a text file.

Parameters **path** (*str*) – Full path to file

Returns File text

Return type list

merlin.files.**readlinesb**(*path*)
Read lines from a binary file.

Parameters **path** (*str*) – Full path to file

Returns File bytes

Return type list

merlin.files.**write**(*path*, *data*)
Write data to a text file.

Parameters

- **path** (*str*) – Full path to file
- **data** (*str*) – File text

Returns Number of characters written

Return type int

merlin.files.**writeb**(*path*, *data*)
Write data to a binary file.

Parameters

- **path** (*str*) – Full path to file
- **data** (*str*) – File bytes

Returns Number of bytes written

Return type int

6.4.6 merlin.formats

merlin.formats.**pyccd**(*x*, *y*, *locations*, *dates_fn*, *specmap*, *chipmap*)

Builds inputs for the pyccd algorithm.

Parameters

- **x** – x projection coordinate of chip
- **y** – y projection coordinate of chip
- **locations** – chip shaped 2d array of projection coordinates
- **dates_fn** (*fn*) – returns dates that should be included in time series
- **specmap** (*dict*) – mapping of keys to specs
- **chipmap** (*dict*) – mapping of keys to chips

Returns A tuple of tuples.

The pyccd format key is `(chip_x, chip_y, x, y)` with a dictionary of sorted numpy arrays representing each spectra plus an additional sorted dates array.

```
>>> pyccd_format(*args)
    (((chip_x, chip_y, x1, y1), {"dates":  [],  "reds": [],
                                 "greens": [],  "blues": [],
                                 "nirs1":  [],  "swir1s": [],
                                 "swir2s": [],  "thermals": [],
                                 "qas":    []}),
     ((chip_x, chip_y, x1, y2), {"dates":  [],  "reds": [],
                                 "greens": [],  "blues": [],
                                 "nirs1":  [],  "swir1s": [],
                                 "swir2s": [],  "thermals": [],
                                 "qas":    []}))
    ...
```

6.4.7 merlin.functions

functions.py is a module of generalized, reusable functions

merlin.functions.**chexists**(*dictionary*, *keys*, *check_fn*)

applies check_fn against dictionary minus keys then ensures the items returned from check_fn exist in dictionary[keys]

Parameters

- **dictionary** (*dict*) – {key: [v1, v3, v2]}
- **keys** (*sequence*) – A sequence of keys in dictionary
- **check_fn** (*function*) – Function that accepts dict and returns sequence of items or Exception

Returns A sequence of items that are returned from check_fn and exist in dictionary[keys] or Exception

merlin.functions.**cqlstr**(*string*)

Makes a string safe to use in Cassandra CQL commands

Parameters **string** – The string to use in CQL

Returns A safe string replacement

Return type str

merlin.functions.**denumpify**(*arg*)

Converts numpy datatypes to python datatypes

bool_ and bool8 are converted to Python bool float64, float32 and float16's are converted to Python float() intc, intp, int8, int16, int32, int64, uint8, uint16, uint32 and uint64 are converted to Python int() **complex_**, complex64 and complex128 are converted to Python complex() None is returned as None numpy ndarrays are returned as list()

Python lists, maps, sets and tuples are returned with all values converted to Python types (recursively).

If there is no implemented converter, returns arg.

Parameters **arg** – A (possibly numpy) data structure

Returns A Python data structure

merlin.functions.**deserialize**(*string*)

Reconstitues datastructure from a string.

Parameters **string** – A serialized data structure

Returns Data structure represented by string parameter

merlin.functions.**difference**(*a*, *b*)

Subtracts items in b from items in a.

Parameters

- **a** – sequence a
- **b** – sequence b

Returns items that exist in a but not b

Return type set

merlin.functions.**extract**(*sequence*, *elements*)

Given a sequence (possibly with nested sequences), extract the element identifed by the elements sequence.

Parameters

- **sequence** – A sequence of elements which may be other sequences
- **elements** – Sequence of nested element indicies (in sequence parameter) to extract

Returns The target element

Example

```
>>> inputs = [1, (2, 3, (4, 5)), 6]
>>> extract(inputs, [0])
>>> 1
>>> extract(inputs, [1])
>>> (2, 3, (4, 5))
>>> extract(inputs, [1, 0])
>>> 2
>>> extract(inputs, [1, 1])
>>> 3
>>> extract(inputs, [1, 2])
>>> (4, 5)
```

(continues on next page)

(continued from previous page)

```
>>> extract(inputs, [1, 2, 0])
>>> 4
```

...

merlin.functions.**flatten**(*iterable*)

Reduce dimensionality of iterable containing iterables

Parameters **iterable** – A multi-dimensional iterable

Returns A one dimensional iterable

merlin.functions.**flip_keys**(*dods*)

Accepts a dictionary of dictionaries and flips the outer and inner keys. All inner dictionaries must have a consistent set of keys or key Exception is raised.

Parameters **dods** – dict of dicts

Returns dict of dicts with inner and outer keys flipped

Example

```
>>> dods = {"reds":   {(0, 0): [110, 110, 234, 664],
                       (0, 1): [23, 887, 110, 111]},
            "greens": {(0, 0): [120, 112, 224, 624],
                       (0, 1): [33, 387, 310, 511]},
            "blues":  {(0, 0): [128, 412, 244, 654],
                       (0, 1): [73, 987, 119, 191]},
>>> flip_keys(dods)
{(0, 0): {"reds":   [110, 110, 234, 664],
          "greens": [120, 112, 224, 624],
          "blues":  [128, 412, 244, 654], ... },
 (0, 1), {"reds":   [23, 887, 110, 111],
          "greens": [33, 387, 310, 511],
          "blues":  [73, 987, 119, 191], ... }}
```

merlin.functions.**insert_into_every**(*dods*, *key*, *value*)

Insert key:values into every subdictionary of dods.

Parameters

- **dods** – dictionary of dictionaries
- **key** – key to hold values in subdictionaires
- **value** – value to associate with key

Returns dictionary of dictionaries with key:values inserted into each

Return type dict

merlin.functions.**intersection**(*items*)

Returns the intersecting set contained in items

Parameters **items** – Two dimensional sequence of items

Returns Intersecting set of items

Example

```
>>> items = [[1, 2, 3], [2, 3, 4], [3, 4, 5]]
>>> intersection(items)
{3}
```

merlin.functions.**isnumeric**(*value*)
Does a string value represent a number (positive or negative?)

Parameters **value** (*str*) – A string

Returns True or False

Return type bool

merlin.functions.**issubset**(*a*, *b*)
Determines if a exists in b

Parameters

- **a** – sequence a
- **b** – sequence b

Returns True or False

Return type bool

merlin.functions.**md5**(*string*)
Computes and returns an md5 digest of the supplied string

Parameters **string** – string to digest

Returns digest value

merlin.functions.**represent**(*item*)
Represents callables and values consistently

Parameters **item** – The item to represent

Returns Item representation

merlin.functions.**rsort**(*iterable*, *key=None*)
Reverse sorts an iterable

merlin.functions.**seqeq**(*a*, *b*)
Determine if two unordered sequences are equal.

Parameters

- **a** – sequence a
- **b** – sequence b

Returns True or False

Return type bool

merlin.functions.**serialize**(*arg*)
Converts datastructure to json, computes digest

Parameters **dictionary** – A python dict

Returns (digest,json)

Return type tuple

merlin.functions.**sha256**(*string*)
Computes and returns a sha256 digest of the supplied string

Parameters **string** (*str*) – string to digest

Returns digest value

merlin.functions.**sort**(*iterable*, *key=None*)
Sorts an iterable

merlin.functions.**timed**(*f*)
Timing wrapper for functions. Prints start and stop time to INFO along with function name, arguments and keyword arguments.

Parameters **f** (*function*) – Function to be timed

Returns Wrapped function

Return type function

6.4.8 merlin.geometry

merlin.geometry.**coordinates**(*points*, *grid*, *snap_fn*)
Returns grid coordinates contained within points.

Points may be specified as dicts, tuples, lists, or sets.

- dict with keys ulx, uly, lrx, lry
- sequence of sequences: ((0,0), (100, 167), (-212, 6621))

Irregular perimeters may be specified in sequences as points will be minboxed.

Points as dicts are an implicit minbox.

Parameters

- **points** (*collection*) – Points outlining an area
- **grid** (*dict*) – The target grid: {'name': 'chip', 'sx': 3000, 'sy': 3000, 'rx': 1, 'ry': -1}
- **snap_fn** (*func*) – A function that accepts x, y and returns a snapped x, y

Returns tuple of tuples of grid coordinates ((x1,y1), (x2,y2) . . .)

Return type tuple

Example

```
>>> grid = {'name': 'chip', 'sx': 500, 'sy': 500, 'rx': 1, 'ry': 1}
>>> sfn = partial(chipmunk.snap, url='http://localhost:5656')
>>> coordinates({'ulx': -1001, 'uly': 1000, 'lrx': -500, 'lry': 500},
                grid=grid,
                snap_fn=sfn)
((-3585.0, 2805.0),
 (-3085.0, 2805.0),
 (-2585.0, 2805.0),
 (-2085.0, 2805.0),
 (-1585.0, 2805.0),
 (-1085.0, 2805.0),
 (-585.0, 2805.0))
```

```
>>> grid = {'name': 'chip', 'sx': 3000, 'sy': 3000, 'rx': 1, 'ry': -1}
>>> coordinates(((112, 443), (112, 500), (100, 443)),
                grid=grid,
                snap_fn=sfn})
((-585.0, 2805.0),)
```

merlin.geometry.**extents**(*ulx*, *uly*, *grid*)

Given an ulx, uly and grid, returns the grid extents.

ulx and uly are not translated during this calculation.

Parameters

- **ulx** (*float*) – 0
- **uly** (*float*) – 0
- **grid** (*dict*) – {'rx', 'ry', 'sx', 'sy'}

Returns {'ulx', 'uly', 'lrx', 'lry'}

Return type dict

Example

```
>>> extents(ulx=0,
            uly=0,
            grid={'rx': 1, 'ry': -1, 'sx': 3000, 'sy': 3000}
{'ulx': 0, 'uly': 0, 'lrx': 2999, 'lry': -2999}
```

merlin.geometry.**minbox**(*points*)

Returns the minimal bounding box necessary to contain points

Parameters **points** (*tuple, list, set*) – ((0,0), (40, 55), (66, 22))

Returns {ulx, uly, lrx, lry}

Return type dict

Example

```
>>> minbox((0, 0), (40, 55), (66,22))
{'ulx': 0, 'uly': 55, 'lrx': 66, 'lry': 0}
```

6.4.9 merlin

merlin.**create**(*x*, *y*, *acquired*, *cfg*)

Create a timeseries.

Parameters

- **x** (*int*) – x coordinate
- **y** (*int*) – y coordinate
- **acquired** (*string*) – iso8601 date range
- **cfg** (*dict*) – A Merlin configuration

Returns tuple - Results of format_fn applied to results of chips_fn

6.4.10 merlin.rods

merlin.rods.**create**(*x*, *y*, *chipseq*, *dateseq*, *locations*, *spec_index*)

Transforms a sequence of chips into a sequence of rods filtered by date, deduplicated, sorted, located and identified.

Parameters

- **x** (*int*) – x projection coordinate of chip
- **y** (*int*) – y projection coordinate of chip
- **chipseq** (*seq*) – sequence of chips
- **dates** (*seq*) – sequence of dates that should be included in the rods
- **locations** (*numpy.Array*) – 2d numpy array of pixel coordinates
- **spec_index** (*dict*) – specs indexed by ubid

Returns {(chip_x, chip_y, x, y): {'k1': [], 'k2': [], 'k3': [], ...}}

Return type dict

merlin.rods.**from_chips**(*chips*)

Accepts sequences of chips and returns time series pixel rods organized by x, y, t for all chips. Chips should be sorted as desired before calling rods() as outputs preserve input order.

Parameters **chips** – sequence of chips with data as numpy arrays

Returns 3d numpy array organized by x, y, and t. Output shape matches input chip shape with the chip value replaced by another numpy array of chip time series values

1. For each chip add data to master numpy array.
2. Transpose the master array
3. Horizontally stack the transposed master array elements
4. Reshape the master array to match incoming chip dimensions
5. Pixel rods are now organized for timeseries access by x, y, t

```
>>> chip_one   = np.int_([[11, 12, 13],
                          [14, 15, 16],
                          [17, 18, 19]])
>>> chip_two   = np.int_([[21, 22, 23],
                          [24, 25, 26],
                          [27, 28, 29]])
>>> chip_three  = np.int_([[31, 32, 33],
                           [34, 35, 36],
                           [37, 38, 39]])
>>> master = np.conj([chip_one, chip_two, chip_three])
>>> np.hstack(master.T).reshape(3, 3, -1)
array([[[ 11, 21, 31], [ 12, 22, 32], [ 13, 23, 33]],
       [[ 14, 24, 34], [ 15, 25, 35], [ 16, 26, 36]],
       [[ 17, 27, 37], [ 18, 28, 38], [ 19, 29, 39]]])
```

merlin.rods.**identify**(*rod*, *x*, *y*)

Adds chip ids (chip_x, chip_y) to the key for a rod

Parameters

- **rod** – dict of (x, y): [values]
- **x** – x coordinate that identifies the source chip
- **y** – y coordinate that identifies the source chip

Returns {(chip_x, chip_y, x, y): [values]}

Return type dict

merlin.rods.**locate**(*rods*, *locations*)

Combines location information with pixel rods.

Parameters

- **rods** – Chip shaped numpy array of rods
- **locations** – Chip shaped numpy array of locations

Returns (location):rod for each location and rod in the arrays.

Return type dict

Incoming locations as 3d array:

```
>>> array([[[0,0], [0,1], [0,2]],
           [[1,0], [1,1], [1,2]],
           [[2,0], [2,1], [2,2]]])
```

Incoming rods also as 3d array:

```
>>> array([[[110,110,234,664], [23,887,110,111], [110,464,223,112]],
           [[111,887,1,110],   [33,111,12,111],  [0,111,66,112]],
           [[12,99,112,110],   [112,87,231,111], [112,45,47,112]]])
```

locrods converts locations to:

```
>>> locations.reshape(locations.shape[0] * locations.shape[1], -1)
array([[0, 0],
       [0, 1],
       [0, 2],
       [1, 0],
       [1, 1],
       [1, 2],
       [2, 0],
       [2, 1],
       [2, 2]])
```

And rods to:

```
>>> rods.reshape(rods.shape[0] * rods.shape[1], -1)
array([[110, 110, 234, 664],
       [23,  887, 110, 111],
       [110, 464, 223, 112],
       [111, 887, 1,   110],
       [33,  111, 12,  111],
       [0,   111, 66,  112],
       [12,  99,  112, 110],
       [112, 87,  231, 111],
       [112, 45,  47,  112]])
```

Then the locations and rods are zipped together via a dictionary comprehension and returned.

```
>>> {
     (0,0): [110, 110, 234, 664],
     (0,1): [23,  887, 110, 111],
     (0,2): [110, 464, 223, 112],
     (1,0): [111, 887, 1,   110],
     (1,1): [33,  111, 12,  111],
     (1,2): [0,   111, 66,  112],
     (2,0): [12,  99,  112, 110],
     (2,1): [112, 87,  231, 111],
     (2,2): [112, 45,  47,  112]
    }
```

6.4.11 merlin.specs

merlin.specs.**exist**(*ubids*, *specs*)
Checks that all ubids are in the specs

Parameters

- **ubids** (*seq*) – [ubid1, ubid2, ubid3, . . .]
- **specs** (*seq*) – [{spec1}, {spec2}, {spec3}]

Returns True or False

Return type bool

merlin.specs.**index**(*specs*)
Organizes specs by ubid

Parameters **specs** (*sequence*) – a sequence of chip specs

Returns specs keyed by ubid

Return type dict

merlin.specs.**mapped**(*ubids*, *specs*)
Organizes specs by key.

Parameters

- **ubids** (*dict*) – {'reds': ['ubid1', 'ubid2'], 'greens': ['ubid3', 'ubid4']}
- **specs** (*seq*) – [{spec1}, {spec2}, {spec3}, {spec4}]

Returns dict

Example

```
>>> {'reds':   [spec1, spec2],
     'greens': [spec3, spec4]}
```

merlin.specs.**only**(*ubids*, *specs*)
Filter specs on ubids.

Parameters

- **ubids** (*seq*) – [ubid1, ubid3]

- **specs** (*seq*) – [{spec1}, {spec2}, {spec3}, ...])

Returns [{spec1}, {spec3}]

Return type tuple

merlin.specs.**refspec**(*specmap*)
Returns the first chip spec from the first key to use as a reference.

Parameters **specmap** – {key: [specs]}

Returns spec

Return type dict

merlin.specs.**ubids**(*specs*)
Extract ubids from a sequence of specs

Parameters **specs** (*sequence*) – a sequence of spec dicts

Returns a sequence of ubids

Return type tuple

6.5 FAQ

I received an empty result from Merlin. No data was received from the source. Adjust x, y, and acquired parameters and verify the Chipmunk instance located at CHIPMUNK_URL has data.

Merlin issued a symmetric data error. This means that there are missing observations from one or more data layers. The data layers should be checked at the source for the given location. This is a built-in QA check on the data which may be replaced by injecting a new `date_fn` into the Merlin profile.

Python Module Index

m

Index

R

S

T

U

W