lcmap-merlin Documentation

Release 1.0

USGS EROS LCMAP

Nov 07, 2017

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

queries = {
    'red':   'http://host/v1/landsat/chip-specs?q=tags:red AND sr',
    'green': 'http://host/v1/landsat/chip-specs?q=tags:green AND sr',
    'blue':  'http://host/v1/landsat/chip-specs?q=tags:blue AND sr'}

timeseries = merlin.create(point=(123, 456),
                           acquired='1980-01-01/2017-01-01',
                           queries=queries,
                           chips_url='http://host/v1/landsat/chips')

print(timeseries)

(((123, 456, 123, 456), {'red'  : [9, 8, ...],
                         'green': [99, 88, ...]},
                         'blue' : [12, 22, ...],
                         'dates': ['2017-01-01', '2016-12-31', ...]}),
 ((123, 456, 124, 456), {'red'  : [4, 3, ...],
                         'green': [19, 8, ...]},
                         'blue' : [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 Cookbook

6.1.1 Create Timeseries

```
import merlin

queries = {
    'red':   'http://host/v1/landsat/chip-specs?q=tags:red AND sr',
    'green': 'http://host/v1/landsat/chip-specs?q=tags:green AND sr',
    'blue':  'http://host/v1/landsat/chip-specs?q=tags:blue AND sr'}

timeseries = merlin.create(point=(123, 456),
                           acquired='1980-01-01/2017-01-01',
```

```
                           queries=queries,
                           chips_url='http://host/v1/landsat/chips')

print(timeseries)

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

6.1.2 Create Timeseries From Assymetric Data

```
from functools import partial
from merlin import chips
from merlin import functions
from merlin import timeseries
import merlin

queries = {
    'red':     'http://host/v1/landsat/chip-specs?q=tags:red AND sr',
    'green':   'http://host/v1/landsat/chip-specs?q=tags:green AND sr',
    'blue':    'http://host/v1/landsat/chip-specs?q=tags:blue AND sr',
    'quality': 'http://host/v1/landsat/chip-specs?q=tags:pixelqa'}

data = timeseries.create(
                  point=(123, 456),
                  dates_fn=partial(functions.chexists,
                                   check_fn=timeseries.symmetric_dates,
                                   keys=['quality',]),
                  chips_url='http://localhost',
                  acquired='1980-01-01/2015-12-31',
                  queries=queries)
```

6.1.3 Retrieve Chips & Specs

```
from merlin.chips      import get as chips_fn
from merlin.chip_specs import get as specs_fn
from merlin.composite  import chips_and_specs

queries = {
    'red':   'http://host/v1/landsat/chip-specs?q=tags:red AND sr',
    'green': 'http://host/v1/landsat/chip-specs?q=tags:green AND sr',
    'blue':  'http://host/v1/landsat/chip-specs?q=tags:blue AND sr'}

chips, specs = chips_and_specs(point=(123, 456),
                               acquired='1980-01-01/2017-08-22',
                               queries=queries,
                               chips_fn=chips_fn,
                               specs_fn=specs_fn,
                               chips_url='http://host/v1/landsat/chips')
```

6.2 Design

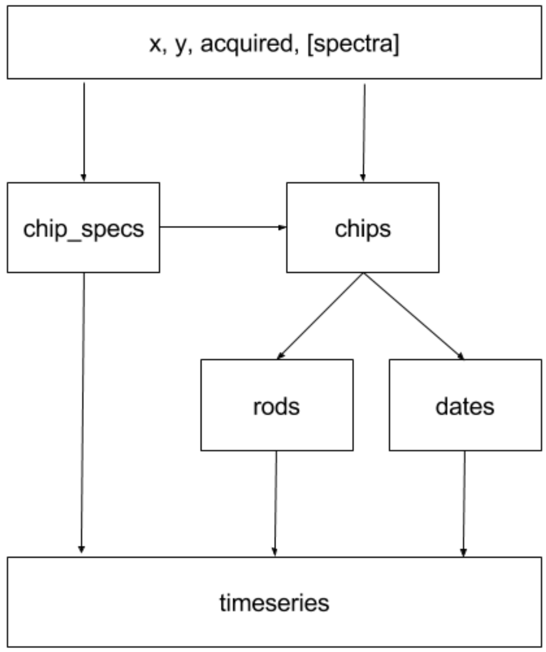

Fig. 6.1: Data Transformations From Source To Timeseries.

Timeseries creation in Merlin is a series of data transformations, beginning with the values for x, y, acquired (range), and a dictionary of chip spec queries. x and y are coordinates in a spatial projection, acquired is an ISO8601 date range string, and queries are ElasticSearch URL queries.

These values are used to retrieve first a set of chip specs, and then the stacks of chips. A chip spec is a dictionary containing shape, data type, and size information for a chip as well as other metadata. A chip is a single two dimensional array of values representing an observation by a particular satellite, from a particular mission, on a known date and in a known spectrum.

A configurable function is applied against the full results of the chips and chip specs queries before any further operations take place. The purpose of this function is to serve as a filter for the data received from the source. This is a logical point for verifying that all data was received correctly and also for determining which dates should and should not be included in the final time series stack.

The filtered stack of chips is combined with chip spec information to split and reorganize the observations into rods, or single pixel height & width observations stacked together in time.

Locations are calculated for each single pixel observation and then merged into a data structure under the appropriate key value for each x & y pair.

The originating chip's upper left x & y are merged with the individual observation's x & y to enable consistent partitioning from source through downstream functions.

Finally, a properly sorted dates array is associated into the top level data structure beside the spectral stacks. The order of the dates array matches the order of observations.

The final data structure appears as...

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

... assuming that the original chip specs query had separate keys for reds, greens, blues, etc.

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 chip and chip spec test data may need to be updated if the source specifications change.

Execute `data.update_specs()` and `data.update_chips()` from a repl. The date range and spatial location of the data may be altered in `merlin/support/__init__.py`. When expanding the data query date range, please note that PyPi has a limit of 60MB per artifact. Uploads exceeding this limit will result in failure messages while publishing.

```
specs_url = 'http://localhost:5678/v1/landsat/chip-specs'
chips_url = 'http://localhost:5678/v1/landsat/chips'

from merlin.support import data
data.update_specs(specs_url=specs_url)
data.update_chips(chips_url=chips_url, specs_url=specs_url)
```

6.3.3 Build Sphinx Docs

This is only necessary during development. Release documents are built automatically by readthedocs.io.

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

6.4 API Reference

6.4.1 merlin.chip_specs

`merlin.chip_specs.`**`byubid`**(*chip_specs*)

Organizes chip_specs by ubid

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

Returns chip_specs keyed by ubid

Return type dict

`merlin.chip_specs.`**`get`**(*query*)

Queries aardvark and returns chip_specs

Parameters **`query`** (*`str`*) – full url query for aardvark

Returns sequence of chip specs

Return type tuple

Example

```
>>> chip_specs('http://host/v1/landsat/chip-specs?q=red AND sr')
('chip_spec_1', 'chip_spec_2', ...)
```

`merlin.chip_specs.`**`ubids`**(*chip_specs*)

Extract ubids from a sequence of chip_specs

Parameters **`chip_specs`** (*`sequence`*) – a sequence of chip_spec dicts

Returns a sequence of ubids

Return type tuple

6.4.2 merlin.chips

`merlin.chips.`**`bounds_to_coordinates`**(*bounds*, *spec*)

Returns chip coordinates from a sequence of bounds. Performs minbox operation on bounds, thus irregular geometries may be supplied.

Parameters

- **`bounds`** – a sequence of bounds.
- **`spec`** – a chip spec representing chip geometry

Returns chip coordinates

Return type tuple

Example

```
>>> xys = bounds_to_coordinates(
                                bounds=((112, 443), (112, 500), (100, 443)),
                                spec=chip_spec)
>>> ((100, 500),)
```

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.**coordinates**(*ulx*, *uly*, *lrx*, *lry*, *chip_spec*)

Returns all the chip coordinates that are needed to cover a supplied bounding box.

Parameters

- **ulx** – upper left x
- **uly** – upper left y
- **lrx** – lower right x
- **lry** – lower right y
- **chip_spec** – dict containing chip_x, chip_y, shift_x, shift_y

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

Return type tuple

This example assumes chip sizes of 500 pixels.

Example

```
>>> chip_coordinates = coordinates(1000, -1000, -500, 500, chip_spec)
((-1000, 500), (-500, 500), (-1000, -500), (-500, -500))
```

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.`**`difference`**(*point*, *interval*)

Calculate difference between a point and 'prior' point on an interval.

The value of this function can be used to answer the question, what do I subtract from a point to find the point of the nearest chip that contains it?

Geospatial raster data geometry can be somewhat counter-inuitive because coordinates and pixel geometry are expressed with both positive and negative values.

Along the x-axis, pixel size (and thus the interval) is a positive number (e.g. 30 * 100). Along the y-axis though, the pixel size and interval is a _negative_ value. Even though this may seem peculiar, using a negative value helps us avoid special cases for finding the nearest tile-point.

Parameters

- **`point`** – a scalar value on the real number line
- **`interval`** – a scalar value describing regularly spaced points on the real number line

Returns difference between a point and prior point on an interval.

`merlin.chips.`**`get`**(*url*, *x*, *y*, *acquired*, *ubids*)

Returns aardvark chips for given x, y, date range and ubid sequence

Parameters

- **`url`** – full url to aardvark endpoint
- **`x`** – longitude
- **`y`** – latitude
- **`acquired`** – ISO8601 daterange '2012-01-01/2014-01-03'
- **`ubids`** – sequence of ubid strings
- **`url`** – string
- **`x`** – number
- **`y`** – number

Returns chips

Return type tuple

Example

```
>>> chips(url='http://host:port/landsat/chips',
          x=123456,
          y=789456,
          acquired='2012-01-01/2014-01-03',
          ubids=['LANDSAT_7/ETM/sr_band1', 'LANDSAT_5/TM/sr_band1'])
```

`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`**(*startx*, *starty*, *chip_spec*)

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

Parameters

- **`startx`** – x coordinate (longitude) of upper left pixel of chip
- **`starty`** – y coordinate (latitude) of upper left pixel of chip

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

`merlin.chips.`**`near`**(*point*, *interval*, *offset*)

Find nearest point given an interval and offset.

The nearest point will be lesser than the point for a positive interval, and greater than the point for a negative interval as is required when finding an 'upper-left' point on a cartesian plane.

This function is used to calculate the nearest points along the x- and y- axis.

Parameters

- **`point`** – a scalar value on the real number line
- **`interval`** – a scalar value describing regularly spaced points on the real number line
- **`offset`** – a scalar value used to shift point before and after finding the 'preceding' interval.

Returns a number representing a point.

`merlin.chips.`**`point_to_chip`**(*x*, *y*, *x_interval*, *y_interval*, *x_offset*, *y_offset*)

Find the nearest containing chip's point.

The resulting *x* value will be less than or equal to the input while the resulting *y* value will be greater than or equal.

For this function to work properly, intervals and offsets must be expressed in terms of projection coordinate system 'easting' and 'northing' units.

Along the x-axis, this works as expected. The interval is a multiple of pixel size and tile size (e.g. 30 * 100 = 3000). Along the y-axis the interval is negative because the pixel size is negative, as you move from the origin of a raster file, the y-axis value _decreases_.

The offset value is used for grids that are not aligned with the origin of the projection coordinate system.

Parameters

- **`x`** – longitudinal value
- **`y`** – latitude value
- **`x_interval`** –
- **`y_interval`** –
- **`x_offset`** –
- **`y_offset`** –

Returns x,y where x and y are the identifying coordinates of a chip.

Return type tuple

`merlin.chips.`**`snap`**(*x*, *y*, *chip_spec*)

Transform an arbitrary projection system coordinate (x,y) into the coordinate of the chip that contains it.

This function only works when working with points on a cartesian plane, it cannot be used with other coordinate systems.

Parameters

- **x** – x coordinate
- **y** – y coordinate
- **chip_spec** – parameters for a chip's grid system

Returns chip x,y

Return type tuple

`merlin.chips.to_numpy(chips, chip_specs_byubid)`

Converts the data for a sequence of chips to numpy arrays

Parameters

- **chips** (*sequence*) – a sequence of chips
- **chip_specs_byubid** (*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

6.4.3 merlin.composite

`merlin.composite.chips_and_specs(point, specs_fn, chips_url, chips_fn, acquired, query)`

Returns chips and specs for a given chip spec query

Parameters

- **point** (*tuple*) – (x, y) which is within the extents of a chip
- **specs_fn** (*function*) – Function that accepts a url query and returns chip specs
- **chips_url** (*str*) – URL to the chips host:port/context
- **chips_fn** (*function*) – Function that accepts x, y, acquired, url, ubids and returns chips.
- **acquired** (*str*) – ISO8601 date range
- **query** (*str*) – URL query to retrieve chip specs

Returns [chips], [specs]

Return type tuple

`merlin.composite.locate(point, spec)`

Returns chip_x, chip_y and all chip locations given a point and spec

Parameters

- **point** (*sequence*) – sequence of x,y
- **spec** (*dict*) – chip spec

Returns (chip_x, chip_y, chip_locations), where chip_locations is a two dimensional chip-shaped numpy array of (x,y)

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.**from_cas**(*cas*)
Transform a dict of chips and specs into a dict of datestrings

Parameters **cas** – chips and specs {k: [chips],[specs]}

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

Return type dict

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

Parameters **acquired** – A date string

Returns; bool: True or False

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.**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.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.`**`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; set: items that exist in a but not b

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

`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.`**`minbox`**

Returns the minimal bounding box necessary to contain points

Parameters **points** (*tuple*) – (x,y) points: ((0,0), (40, 55), (66, 22))

Returns ulx, uly, lrx, lry

Return type dict

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.**simplify_objects**(*obj*)

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

6.4.7 merlin.rods

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

Description:

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

Combines location information with pixel rods.

Parameters

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

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

Return type dict

Description: 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]
    }
```

6.4.8 merlin.timeseries

merlin.timeseries.**add_dates**(*dates*, *dods*, *key='dates'*)

Inserts dates into each subdictionary of the parent dictionary.

Parameters

- **dod** – A dictionary of dictionaries
- **dates** – A sequence of dates
- **key** – Subdict key where dates values is inserted

Returns An updated dictionary of dictionaries with

merlin.timeseries.**create**(*point*, *chips_url*, *acquired*, *queries*, *chips_fn=<function get>*, *dates_fn=<function symmetric_dates>*, *format_fn=<function pyccd_format>*, *specs_fn=<function get>*)

Queries data, performs date filtering/checking and formats the results.

Parameters

- **point** – Tuple of (x, y) which is within the extents of a chip
- **chips_url** – URL to the chips host:port/context
- **acquired** – Date range string as start/end, ISO 8601 date format
- **queries** – dict of URL queries to retrieve chip specs keyed by spectra
- **chips_fn** – Function that accepts x, y, acquired, url, ubids and returns chips.
- **dates_fn** – Function that accepts dict of {spectra: [specs],[chips]} and returns a sequence of dates that should be included in the time series. May raise an Exception to halt time series construction.
- **format_fn** – Function that accepts chip_x, chip_y, chip_locations, chips_and_specs, dates and returns it's representation of a time series.
- **specs_fn** – Function that accepts a url query and returns chip specs

Returns Return value from format_fn

merlin.timeseries.**errorhandler**(*msg=''*, *raises=False*)
Constructs, logs and raises error messages

Parameters

- **msg** – Custom message string
- **raises** – Whether to raise an exception or not

Returns exception handler function

merlin.timeseries.**identify**(*chip_x*, *chip_y*, *rod*)
Adds chip ids (chip_x, chip_y) to the key for each dict entry

Parameters

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

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

Return type dict

merlin.timeseries.**pyccd_format**(*chip_x*, *chip_y*, *chip_locations*, *chips_and_specs*, *dates*)
Builds inputs for the pyccd algorithm.

Parameters

- **chip_x** – x coordinate for chip identifier
- **chip_y** – y coordinate for chip identifier
- **chip_locations** – chip shaped 2d array of projection coordinates
- **chips_and_specs** – {k: [chips],[specs]}
- **dates** – sequence of chip dates to be included in output

Returns A tuple of tuples.

Description: The pyccd format requires a key of (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": [],
                             "quality": []}),
 ((chip_x, chip_y, x1, y2), {"dates": [],  "reds": [],
                             "greens": [], "blues": [],
                             "nirs1": [],  "swir1s": [],
                             "swir2s": [], "thermals": [],
                             "quality": []}))
...
```

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

Parameters **cas** – chips and specs {key: [chips],[specs]}

Returns chip spec

Return type dict

merlin.timeseries.**sort**(*chips*, *key=<function <lambda>>*)
Sorts all the returned chips by date.

Parameters **chips** – sequence of chips

Returns sorted sequence of chips

merlin.timeseries.**symmetric_dates**(*dates*)
Returns a sequence of dates for chips that should be included in downstream functions. May raise Exception.

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

Returns Sequence of date strings or Exception

Example

```
>>> symmetrical_dates({"red":  [ds3, ds1, ds2],
                       "blue": [ds2, ds3, ds1]})
[2, 3, 1]
>>>
>>> symmetrical_dates({"red":  [ds3, ds1],
                       "blue": [ds2, ds3, ds1]})
Exception: red:[3, 1] does not match blue:[2, 3, 1]
```

6.4.9 merlin.support

merlin.support.**chip_spec_queries**(*url*)
A map of pyccd spectra to chip-spec queries

Parameters **url** (*str*) – full url (http://host:port/resource) for chip-spec endpoint

Returns map of spectra to chip spec queries

Return type dict

Example:

```
>>> chip_spec_queries('http://host/v1/landsat/chip-specs')
{'reds':     'http://host/v1/landsat/chip-specs?q=tags:red AND sr',
 'greens':   'http://host/v1/landsat/chip-specs?q=tags:green AND sr'
 'blues':    'http://host/v1/landsat/chip-specs?q=tags:blue AND sr'
 'nirs':     'http://host/v1/landsat/chip-specs?q=tags:nir AND sr'
 'swir1s':   'http://host/v1/landsat/chip-specs?q=tags:swir1 AND sr'
 'swir2s':   'http://host/v1/landsat/chip-specs?q=tags:swir2 AND sr'
 'thermals': 'http://host/v1/landsat/chip-specs?q=tags:thermal AND ta'
 'quality':  'http://host/v1/landsat/chip-specs?q=tags:pixelqa'}
```

merlin.support.**data_config**()

Default configuration for retrieving and serving test data

Returns x, y, acquired, dataset_name, chips_dir, specs_dir

Return type dict

6.4.10 merlin.support.data

Functions for working with local data. This module allows merlin functions to test using local data rather than requiring external systems such as aardvark to be available.

Mock servers (such as aardvark) live in other modules, not here.

There are functions contained for updating the data that lives under merlin/support/data. The locations of this data is controlled by values in merlin/support/__init__.py

merlin.support.data.**chips**(*spectra, support.data_config()['chips_dir']*)

Return chips for named spectra

Parameters

- **spectra** (*str*) – red, green, blue, nir, swir1, swir2, thermal or cfmask
- **root_dir** (*str*) – directory where chips are located

Returns sequence of chips

merlin.support.data.**chip_specs**(*spectra, root_dir=support.data_config()['specs_dir']*)

Returns chip specs for the named spectra.

Parameters

- **spectra** (*str*) – red, green, blue, nir, swir1, swir2, thermal or cfmask
- **root_dir** (*str*) – directory where chip specs are located

Returns sequence of chip specs

merlin.support.data.**chip_ids**(*root_dir=support.data_config()['chips_dir']*)

Returns chip ids for available chip data in root_dir

Parameters **root_dir** – directory where chips are located

Returns tuple of tuples of chip ids (UL coordinates)

merlin.support.data.**spectra_from_queryid**(*queryid, root_dir=support.data_config()['specs_dir']*)

Returns the spectra name for a chip spec from the supplied queryid

Parameters

- **queryid** (*str*) – query identifier
- **root_dir** (*str*) – directory where chip specs are located

Returns spectra names associated with the query identifier

Return type list

`merlin.support.data.`**`test_specs`**(*root_dir=support.data_config()['specs_dir']*)

Returns a dict of all test chip specs keyed by spectra

Parameters **`root_dir`** (*`str`*) – directory where chip specs are located

Returns spectra: [chip_spec1, chip_spec2, ...]

Return type dict

`merlin.support.data.`**`update_specs`**(*specs_url*, *conf=support.data_config()*)

Updates the spec test data

Parameters

- **`specs_url`** (*`str`*) – chip spec query
- **`conf`** (*`dict`*) – key:values for data_config

Returns True or Exception

Return type bool

`merlin.support.data.`**`update_chips`**(*chips_url*, *specs_url*, *conf=support.data_config()*)

Updates the chip test data

Parameters

- **`chips_url`** (*`str`*) – chips url
- **`specs_url`** (*`str`*) – chip spec query
- **`conf`** (*`dict`*) – key:values for data_config

Returns True or Exception

Return type bool

`merlin.support.data.`**`live_specs`**(*specs_url*)

Returns a dict of all chip specs defined by the driver.chip_spec_urls keyed by spectra

Parameters **`specs_url`** (*`str`*) – chip spec query

Returns spectra: [chip_spec1, chip_spec2, ...]

Return type dict

`merlin.support.data.`**`spec_query_ids`**(*specs_url*)

Returns a dictionary of key: query from a specs_url

Parameters **`specs_url`** (*`str`*) – URL to chip specs

Returns key:query for all the queries associated with specs_url

Return type dict

`merlin.support.data.`**`spec_query_id`**(*url*)

Generates identifier for spec query url based on the querystring

Parameters **`url`** (*`str`*) – url containing a querystring

Returns query identifier

Return type str

merlin.support.data.**spectra_from_specfile**(*filename*)
Returns the spectra the named chip spec file is associated with

Parameters **filename** (*str*) – File to obtain spectra from

Returns spectra associated with file

Return type str

merlin.support.data.**spectra_index**(*specs*)
Returns a dict keyed by ubid that maps to the spectra name

Parameters **specs** (*dict*) – A dict of spectra: chip_specs

Returns spectra

Return type A dict of ubid

6.4.11 merlin.support.aardvark

Filesystem based local aardvark operations.

merlin.support.aardvark.**chip_specs**(*url*)
Return chip specs from local disk

Parameters **url** (*str*) – chip spec query

Returns sequence of chip specs matching query

Return type tuple

merlin.support.aardvark.**chips**(*x*, *y*, *acquired*, *url*, *ubids*)
Return chips from local disk

Parameters

- **x** (*number*) – x coordinate
- **y** (*number*) – y coordinate
- **acquired** (*str*) – ISO8601 date range string
- **url** (*str*) – Not used. Necessary to support chips interface
- **ubids** (*sequence*) – universal band ids

Returns sequence of chips

Return type tuple

6.5 FAQ

I received *ValueError: need at least one array to concatenate*. No data was received from the source. Adjust x, y, acquired or queries for an area, time and set of ubids that have data.

Python Module Index

m

Index

P

R

S

T

U

W