Plio Documentation

Release Please install this project with setup.py

Jay Laura

Mar 10, 2021

CONTENTS

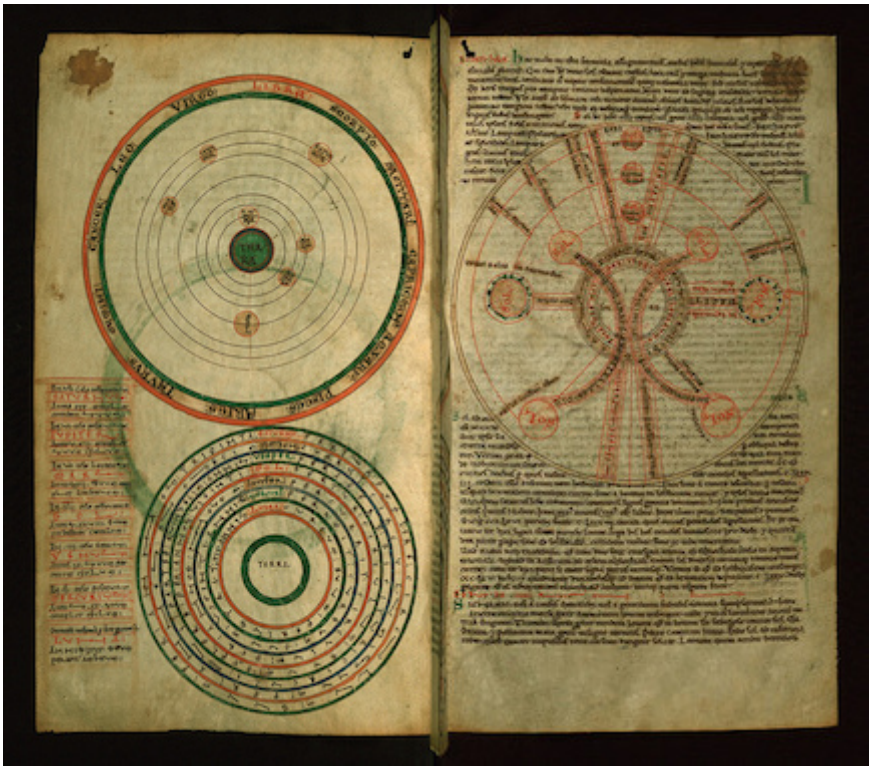


Fig. 1: The Walters Art Museum has scanned and released into the public domain a twelfth century illuminated manuscript created by Christian monks that served as a scientific text. The two reproduced pages illustrate 'the heavens'.

Planetary I/O (Plio) is an open source collection of my commonly used I/O functionality. Plio is licensed in the public domain.

CHAPTER

ONE

ISSUES AND VERSION CONTROL

This project is hosted on GitHub. If you run into a problem, please open an issue on our issue tracker.

CHAPTER

TWO

DOCUMENTATION

2.1 User Guide

2.1.1 Installation

We provide Planetary I/O (plio) as a binary package via conda and for installation via the standard setup.py script.

Via Conda

1. Download and install the Python 3.x Miniconda installer. Respond `Yes` when prompeted to add conda to your BASH profile.
2. Bring up a command line and add the `conda-forge` channel to your channel list: `conda config --add channels conda-forge`. This adds an entry to your `~/.condarc` file.
3. Install plio: `conda install -c jlaura plio`
4. To update plio: `conda update -c jlaura plio`

Via setup.py

This method assumes that you have the necessary dependencies already installed. The installation of dependencies can be non-trivial because of GDAL. We supply an `environment.yml` file that works with Anaconda Python's `conda env` environment management tool.

2.1.2 Getting Started with Plio

Reading a Spatial Data Set using the GDAL Wrapper

[1]:
```
# First, we import the io_gdal module
from plio.io import io_gdal
```

[2]:
```
# We will use an example that ships with plio, so we use the get_path function in the␣
↪examples module
from plio.examples import get_path
```

[3]:
```
# For this example, we will use a lambert projected LOLA hillshade.
path_to_our_example = get_path('Lunar_LRO_LOLA_Shade_MAP2_90.0N20.0_LAMB.tif')
ds = io_gdal.GeoDataset(path_to_our_example)
```

[4]:
```
# Just printing the geodataset object returns the filename
ds
```

[4]:
```
Lunar_LRO_LOLA_Shade_MAP2_90.0N20.0_LAMB.tif
```

[5]:
```
# Here we want to see what attributes and methods are available on the geodataset.
attributes = [i for i in dir(ds) if not i.startswith('__')]
print(attributes)
```

```
['base_name', 'central_meridian', 'coordinate_transformation', 'dataset', 'file_name',
↪ 'footprint', 'geospatial_coordinate_system', 'geotransform', 'inverse_coordinate_
↪transformation', 'latlon_corners', 'latlon_extent', 'latlon_to_pixel', 'metadata',
↪'no_data_value', 'pixel_area', 'pixel_height', 'pixel_to_latlon', 'pixel_width',
↪'proj_corners', 'proj_extent', 'raster_size', 'read_array', 'scale', 'spatial_
↪reference', 'spheroid', 'standard_parallels', 'unit_type', 'x_rotation', 'xy_corners
↪', 'xy_extent', 'y_rotation']
```

The dataset lazy loads, the the `read_array` method has not yet read any data into memory beyond the file handle. In fact, most of the attributes lazy load and will not populate until we access them.

[6]:
```
# Get some PATH information for the data set
print('The filename is {}'.format(ds.base_name))
print('The full PATH is {}'.format(ds.file_name))
```

```
The filename is Lunar_LRO_LOLA_Shade_MAP2_90.0N20.0_LAMB
The full PATH is /Users/jlaura/github/plio/plio/examples/Projections/Lunar_LRO_LOLA_
↪Shade_MAP2_90.0N20.0_LAMB.tif
```

[7]:
```
# How about a spatial reference.
print(ds.spatial_reference)
```

```
PROJCS["LambertConformal_MOON",
    GEOGCS["GCS_MOON",
        DATUM["MOON",
            SPHEROID["MOON",1737400,0]],
        PRIMEM["Reference_Meridian",0],
        UNIT["Degree",0.017453292519943295]],
    PROJECTION["Lambert_Conformal_Conic_2SP"],
    PARAMETER["standard_parallel_1",73],
    PARAMETER["standard_parallel_2",42],
    PARAMETER["latitude_of_origin",90],
    PARAMETER["central_meridian",20],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
```

The above is a human readable WKT string. We can take that to proj4 if we like using GDAL functionality (GDAL is a dependency, so it is installed on your system already!)

[8]:
```
# Have a look at what exactly the string we see above it.
srs = ds.spatial_reference
type(srs)
```

[8]:
```
osgeo.osr.SpatialReference
```

[9]:
```
# And convert to a proj4 string
srs.ExportToProj4()
```

[9]:
```
'+proj=lcc +lat_1=73 +lat_2=42 +lat_0=90 +lon_0=20 +x_0=0 +y_0=0 +a=1737400␣
↪+b=1737400 +units=m +no_defs '
```

Frequently, in development we need to know a bit about an image. Things like the resolution of the pixels or the spatial extent. These attributes are also available.

[10]:
```
# The extent in geographic coordinates.
print('The data set extent (in GCS) is: {}'.format(ds.latlon_extent))

# or the resolution.
print('The data set resolution is {},{}'.format(ds.pixel_width, ds.pixel_height))
# The pixel width and height are as we expect them to be.  The y-axis is negative and␣
↪the pixels are square.
```

```
The data set extent (in GCS) is: [(69.90349154912009, -29.72166902463681), (37.
↪86992376608661, 39.11610200134293)]
The data set resolution is 3870.0,-3870.0
```

Sometimes we also want to convert from a lat/lon coordinate to a pixel. This requires use of the spatial reference and the geotransformation parameters. This is not an exceptionally robust transformation using a camera model, but a simple affine transformation.

[11]:
```
# Convert from pixel to lat/lon
ds.pixel_to_latlon(100,100)
```

[11]:
```
(62.79324246013935, 14.209561437813814)
```

[12]:
```
# And vice-versa
ds.latlon_to_pixel(62.79324246013935, 14.209561437813814)
# Here we illustrate the imperfection of the transformation...
# In practice pixels are measures in whole numbers, so we round.
```

[12]:
```
(100.00000000000001, 99.99999999999997)
```

[13]:
```
# Finally, we can access the image data itself as a NumPy array
arr = ds.read_array()
type(arr)
```

[13]:
```
numpy.ndarray
```

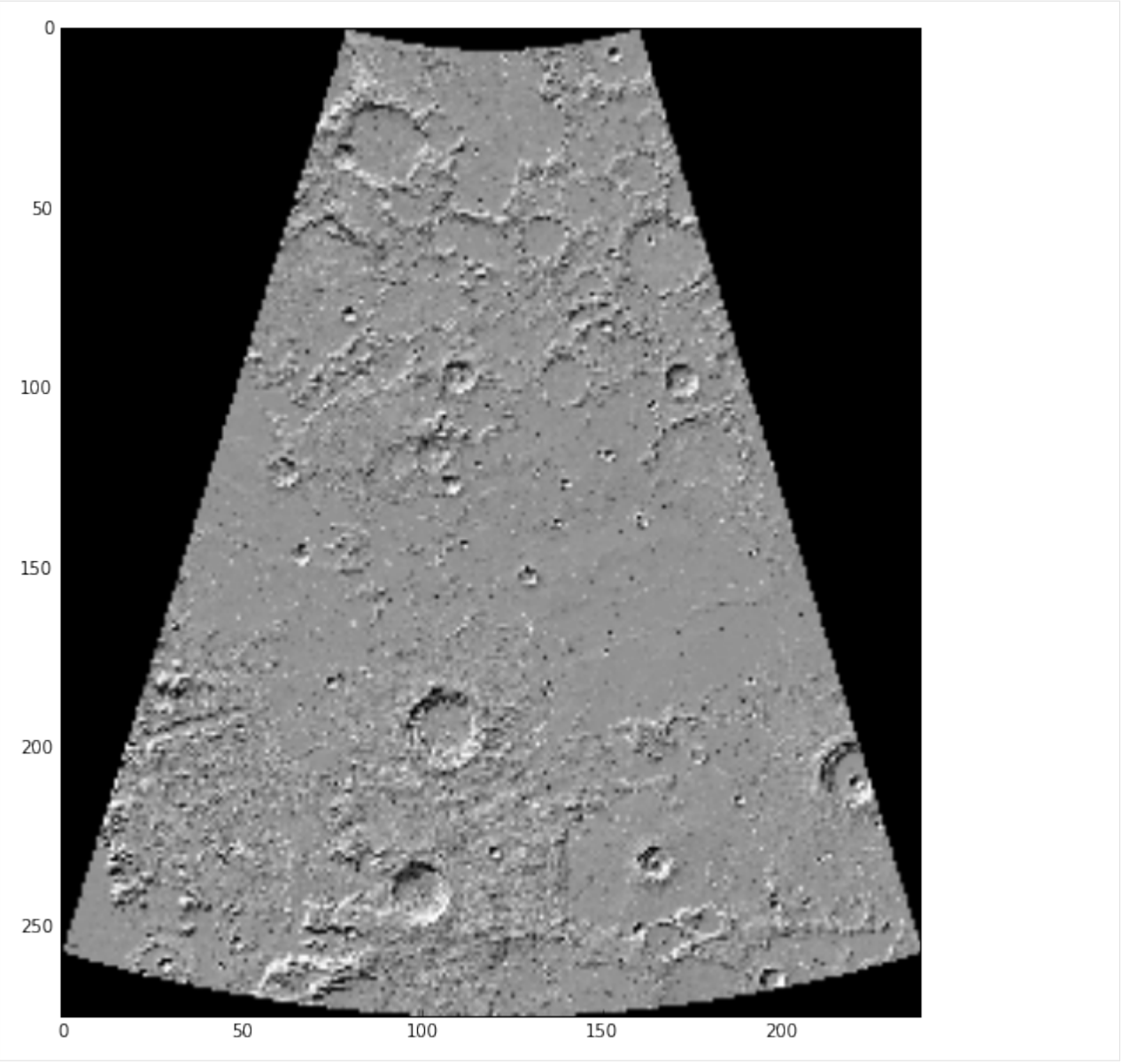
[14]:
```
arr.shape
```

[14]:
```
(275, 239)
```

[15]:
```
# Just an iPython magic to get the figure as an embedded png
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

[16]:
```
# Finally, a plot (this image has been downsamples a ton to make it easier to ship␣
↪via github - just an example!)
figsize(10,10)
imshow(arr, cmap='gray')
show()
```



[3]:
```
import plio
from plio.io.io_tes import Tes
from plio.examples import get_path
```

[9]:
```
pos = Tes(get_path('pos10001.tab'))
```

[10]:
```
pos.data
```

[10]:
```
       sclk_time            et  \
0      604702680 -2.649248e+07
1      604702684 -2.649247e+07
2      604702686 -2.649247e+07
3      604702688 -2.649247e+07
4      604702690 -2.649247e+07
5      604702692 -2.649247e+07
```

(continues on next page)

(continued from previous page)

```
6       604702694 -2.649246e+07
7       604702696 -2.649246e+07
8       604702698 -2.649246e+07
9       604702700 -2.649246e+07
10      604702872 -2.649229e+07
11      604702876 -2.649228e+07
12      604702878 -2.649228e+07
13      604702880 -2.649228e+07
14      604702882 -2.649228e+07
15      604702884 -2.649227e+07
16      604702886 -2.649227e+07
17      604702888 -2.649227e+07
18      604702890 -2.649227e+07
19      604702892 -2.649227e+07
20      604702894 -2.649226e+07
21      604703064 -2.649209e+07
22      604703068 -2.649209e+07
23      604703070 -2.649209e+07
24      604703072 -2.649209e+07
25      604703074 -2.649208e+07
26      604703076 -2.649208e+07
27      604703078 -2.649208e+07
28      604703080 -2.649208e+07
29      604703082 -2.649208e+07
...           ...           ...
19821   605129704 -2.606545e+07
19822   605129706 -2.606545e+07
19823   605129708 -2.606545e+07
19824   605129710 -2.606545e+07
19825   605129712 -2.606545e+07
19826   605129714 -2.606544e+07
19827   605129716 -2.606544e+07
19828   605129718 -2.606544e+07
19829   605129720 -2.606544e+07
19830   605129722 -2.606544e+07
19831   605129724 -2.606543e+07
19832   605129726 -2.606543e+07
19833   605129728 -2.606543e+07
19834   605129730 -2.606543e+07
19835   605129732 -2.606543e+07
19836   605129734 -2.606542e+07
19837   605129736 -2.606542e+07
19838   605129738 -2.606542e+07
19839   605129740 -2.606542e+07
19840   605129742 -2.606542e+07
19841   605129744 -2.606541e+07
19842   605129746 -2.606541e+07
19843   605129748 -2.606541e+07
19844   605129750 -2.606541e+07
19845   605129752 -2.606541e+07
19846   605129754 -2.606540e+07
19847   605129756 -2.606540e+07
19848   605129758 -2.606540e+07
19849   605129760 -2.606540e+07
19850   605129762 -2.606540e+07

                                                      pos  \
```

(continues on next page)

(continued from previous page)

```
0      (1321.625, 3328.09814453125, -1171.3719482421875)
1      (1331.2796630859375, 3327.698974609375, -1161...
2      (1336.1007080078125, 3327.483154296875, -1157...
3      (1340.9173583984375, 3327.2568359375, -1152.47...
4      (1345.729736328125, 3327.02001953125, -1147.73...
5      (1350.5379638671875, 3326.772216796875, -1143...
6      (1355.3416748046875, 3326.513916015625, -1138...
7      (1360.1412353515625, 3326.2451171875, -1133.51...
8      (1364.936279296875, 3325.96533203125, -1128.76...
9      (1369.7271728515625, 3325.675048828125, -1124...
10     (1763.795654296875, 3261.053955078125, -703.44...
11     (1772.500244140625, 3258.625732421875, -693.42...
12     (1776.843994140625, 3257.39599609375, -688.412...
13     (1781.18212890625, 3256.15576171875, -683.3973...
14     (1785.5146484375, 3254.9052734375, -678.380310...
15     (1789.8414306640625, 3253.644287109375, -673.3...
16     (1794.1624755859375, 3252.373046875, -668.3395...
17     (1798.477783203125, 3251.09130859375, -663.316...
18     (1802.787353515625, 3249.79931640625, -658.290...
19     (1807.09130859375, 3248.496826171875, -653.262...
20     (1811.389404296875, 3247.183837890625, -648.23...
21     (2154.297119140625, 3098.24658203125, -214.761...
22     (2161.79931640625, 3093.86865234375, -204.4632...
23     (2165.5400390625, 3091.6650390625, -199.312927...
24     (2169.274169921875, 3089.451416015625, -194.16...
25     (2173.001220703125, 3087.22802734375, -189.010...
26     (2176.721435546875, 3084.994873046875, -183.85...
27     (2180.4345703125, 3082.751953125, -178.7055206...
28     (2184.140869140625, 3080.499267578125, -173.55...
29     (2187.84033203125, 3078.236572265625, -168.398...
...                                                  ...
19821  (-848.8919067382812, -3402.706787109375, 1454...
19822  (-853.8307495117188, -3403.334228515625, 1449...
19823  (-858.7669067382812, -3403.950927734375, 1445...
19824  (-863.7003784179688, -3404.556884765625, 1440...
19825  (-868.6311645507812, -3405.15234375, 1436.3658...
19826  (-873.5592651367188, -3405.737060546875, 1431...
19827  (-878.484619140625, -3406.31103515625, 1427.35...
19828  (-883.4072265625, -3406.874267578125, 1422.838...
19829  (-888.3271484375, -3407.427001953125, 1418.319...
19830  (-893.2442626953125, -3407.968994140625, 1413...
19831  (-898.1585693359375, -3408.50048828125, 1409.2...
19832  (-903.0701293945312, -3409.021240234375, 1404...
19833  (-907.9788818359375, -3409.53125, 1400.2022705...
19834  (-912.884765625, -3410.030517578125, 1395.6618...
19835  (-917.787841796875, -3410.51904296875, 1391.11...
19836  (-922.68798828125, -3410.9970703125, 1386.5678...
19837  (-927.5853271484375, -3411.46435546875, 1382.0...
19838  (-932.4797973632812, -3411.921142578125, 1377...
19839  (-937.3712768554688, -3412.366943359375, 1372...
19840  (-942.2598876953125, -3412.80224609375, 1368.3...
19841  (-947.1455688476562, -3413.226806640625, 1363...
19842  (-952.0282592773438, -3413.640625, 1359.181640...
19843  (-956.9080200195312, -3414.043701171875, 1354...
19844  (-961.7847900390625, -3414.436279296875, 1350...
19845  (-966.6585693359375, -3414.81787109375, 1345.4...
19846  (-971.529296875, -3415.18896484375, 1340.83837...
```

(continues on next page)

(continued from previous page)

```
19847  (-976.3970336914062, -3415.54931640625, 1336.2...
19848  (-981.26171875, -3415.89892578125, 1331.641357...
19849  (-986.1233520507812, -3416.238037109375, 1327...
19850  (-990.9818725585938, -3416.566162109375, 1322...

                                          sun  \
0       (242380016.0, 35959824.0, 9939954.0)
1       (242379984.0, 35959904.0, 9939991.0)
2       (242379984.0, 35959940.0, 9940010.0)
3       (242379968.0, 35959980.0, 9940028.0)
4       (242379968.0, 35960020.0, 9940047.0)
5       (242379952.0, 35960060.0, 9940065.0)
6       (242379952.0, 35960100.0, 9940083.0)
7       (242379936.0, 35960140.0, 9940102.0)
8       (242379936.0, 35960180.0, 9940120.0)
9       (242379920.0, 35960220.0, 9940139.0)
10      (242379136.0, 35963636.0, 9941726.0)
11      (242379120.0, 35963712.0, 9941763.0)
12      (242379104.0, 35963752.0, 9941782.0)
13      (242379104.0, 35963792.0, 9941800.0)
14      (242379088.0, 35963832.0, 9941819.0)
15      (242379088.0, 35963872.0, 9941837.0)
16      (242379072.0, 35963912.0, 9941856.0)
17      (242379056.0, 35963952.0, 9941874.0)
18      (242379056.0, 35963992.0, 9941892.0)
19      (242379040.0, 35964032.0, 9941911.0)
20      (242379040.0, 35964072.0, 9941929.0)
21      (242378256.0, 35967448.0, 9943498.0)
22      (242378240.0, 35967524.0, 9943535.0)
23      (242378240.0, 35967564.0, 9943554.0)
24      (242378224.0, 35967604.0, 9943572.0)
25      (242378208.0, 35967644.0, 9943591.0)
26      (242378208.0, 35967684.0, 9943609.0)
27      (242378192.0, 35967724.0, 9943628.0)
28      (242378192.0, 35967764.0, 9943646.0)
29      (242378176.0, 35967804.0, 9943664.0)
...                                        ...
19821  (240235392.0, 44406032.0, 13871953.0)
19822  (240235392.0, 44406072.0, 13871971.0)
19823  (240235376.0, 44406112.0, 13871989.0)
19824  (240235360.0, 44406152.0, 13872008.0)
19825  (240235360.0, 44406192.0, 13872026.0)
19826  (240235344.0, 44406228.0, 13872044.0)
19827  (240235328.0, 44406268.0, 13872063.0)
19828  (240235328.0, 44406308.0, 13872081.0)
19829  (240235312.0, 44406348.0, 13872099.0)
19830  (240235296.0, 44406388.0, 13872118.0)
19831  (240235296.0, 44406428.0, 13872136.0)
19832  (240235280.0, 44406468.0, 13872155.0)
19833  (240235264.0, 44406504.0, 13872173.0)
19834  (240235264.0, 44406544.0, 13872191.0)
19835  (240235248.0, 44406584.0, 13872210.0)
19836  (240235232.0, 44406624.0, 13872228.0)
19837  (240235216.0, 44406664.0, 13872246.0)
19838  (240235216.0, 44406704.0, 13872265.0)
19839  (240235200.0, 44406744.0, 13872283.0)
19840  (240235184.0, 44406780.0, 13872302.0)
```

(continues on next page)

(continued from previous page)

```
19841  (240235184.0, 44406820.0, 13872320.0)
19842  (240235168.0, 44406860.0, 13872338.0)
19843  (240235152.0, 44406900.0, 13872357.0)
19844  (240235152.0, 44406940.0, 13872375.0)
19845  (240235136.0, 44406980.0, 13872393.0)
19846  (240235120.0, 44407016.0, 13872412.0)
19847  (240235120.0, 44407056.0, 13872430.0)
19848  (240235104.0, 44407096.0, 13872448.0)
19849  (240235088.0, 44407136.0, 13872467.0)
19850  (240235088.0, 44407176.0, 13872485.0)

                                                   quat             id
0      (0.1824042946100235, -0.3314073383808136, -0.4...  (b'c', b'c')
1      (0.18153679370880127, -0.33282148838043213, -0...  (b'c', b'c')
2      (0.18109986186027527, -0.33354103565216064, -0...  (b'c', b'c')
3      (0.1806604117155075, -0.33425629138946533, -0...  (b'c', b'c')
4      (0.1802179366350174, -0.33496642112731934, -0...  (b'c', b'c')
5      (0.1797812581062317, -0.33567821979522705, -0...  (b'c', b'c')
6      (0.17935168743133545, -0.3363921642303467, -0...  (b'c', b'c')
7      (0.17892010509967804, -0.33710038661956787, -0...  (b'c', b'c')
8      (0.17848612368106842, -0.3378016948699951, -0...  (b'c', b'c')
9      (0.17805038392543793, -0.3385055959224701, -0...  (b'c', b'c')
10     (0.14008298516273499, -0.39840176701545715, -0...  (b'c', b'c')
11     (0.13917456567287445, -0.3997543156147003, -0...  (b'c', b'c')
12     (0.13872121274471283, -0.40044155716896057, -0...  (b'c', b'c')
13     (0.13827399909496307, -0.40112653374671936, -0...  (b'c', b'c')
14     (0.1378343552350998, -0.40180885791778564, -0...  (b'c', b'c')
15     (0.13739295303821564, -0.4024978578090668, -0...  (b'c', b'c')
16     (0.13694943487644196, -0.40319517254829407, -0...  (b'c', b'c')
17     (0.13650058209896088, -0.40388038754463196, -0...  (b'c', b'c')
18     (0.13604523241519928, -0.4045509099960327, -0...  (b'c', b'c')
19     (0.13559292256832123, -0.40522345900535583, -0...  (b'c', b'c')
20     (0.13514433801174164, -0.4058985114097595, -0...  (b'c', b'c')
21     (0.09671157598495483, -0.4623425304889679, -0...  (b'c', b'c')
22     (0.09579837322235107, -0.46363431215286255, -0...  (b'c', b'c')
23     (0.09533984214067459, -0.46428006887435913, -0...  (b'c', b'c')
24     (0.09488427639007568, -0.46492496132850647, -0...  (b'c', b'c')
25     (0.09443235397338867, -0.46556881070137024, -0...  (b'c', b'c')
26     (0.093976229429245, -0.4662158489227295, -0.50...  (b'c', b'c')
27     (0.09351497888565063, -0.4668668508529663, -0...  (b'c', b'c')
28     (0.09305869042873383, -0.46751669049263, -0.50...  (b'c', b'c')
29     (0.0926084965467453, -0.46816515922546387, -0...  (b'c', b'c')
...                                                  ...           ...
19821                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19822                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19823                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19824                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19825                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19826                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19827                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19828                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19829                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19830                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19831                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19832                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19833                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
19834                            (0.0, -0.0, -0.0, -0.0)  (b'c', b'c')
```

(continues on next page)

(continued from previous page)

```
19835                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19836                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19837                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19838                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19839                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19840                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19841                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19842                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19843                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19844                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19845                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19846                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19847                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19848                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19849                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')
19850                              (0.0, -0.0, -0.0, -0.0)   (b'c', b'c')

[19851 rows x 6 columns]
```

2.2 Plio Reference

Release Please install this project with setup.py

Date Mar 10, 2021

2.2.1 Date/Time Utilities

In this section Input/Output docstring are presented, grouped by the type of file they are designed to work on. The examples assume that Plio is imported with:

```
>>> import plio
```

Date/Time Conversion

`date.julian2ls` — Conversion from Julian to LsubS

The `date.julian2ls` module provides functionality for the conversion between Julian date and LsubS

New in version 0.1.0.

`plio.date.julian2ls.`**`julian2ls`**(*date*, *marsyear=None*, *reverse=False*)

Original IDL from Hugh Keiffer

Parameters

- **`date`** (*`numeric`*) – Scalar or NumPy ndarray or dates
- **`marsyear`** (*`float`*) – Mars year for use in reverse
- **`reverse`** (*`bool`*) – Reverse conversion from L_{s} to julian

Returns

- **out** (*numeric*) – float or array of float LsubS or Julian dates

- **myn** (*float*) – If LsubS to Mars year, return the Mars year

References

[1] M. Allison and M. McEwen.' A post-Pathfinder evaluation of areocentric solar coordinates with improved timing recipes for Mars seasonal/diurnal climate studies'. Plan. Space Sci., 2000, v=48, pages = {215-235},

[2] http://www.giss.nasa.gov/tools/mars24/help/algorithm.html

`plio.date.julian2ls.`**`zero360`**(*angles*, *rad=False*)

Convert angle to base 0-360

Parameters

- **`angles`** (*`float`*) – a scalar angle to convert
- **`rad`** (*`boolean`*) – flag whether angles are in radians

Returns **bb** – converted angle

Return type float

`date.julian2season` — Conversion from Julian to Mars Season

The `date.julian2season` module provides functionality for the conversion between Julian date and Martian season

New in version 0.1.0.

`plio.date.julian2season.`**`j2season`**(*_date*, *year=686.9799625*, *marsday=8.5875*, *startdate=144.61074994*)

Ls date to a KRC season to determine which KRC seasonal lookup tables to use

Parameters **`_date`** (*`float`*) – The input date to be converted

year [float] The mars year to search within

marsday [float] The length of a Mars day

startdate [float] The zero, start date

Returns

- **startseason** (*int*) – The integer index to the start season
- **stopseasons** (*int*) – The integer index to the stop season

2.2.2 Input/Output

In this section Input/Output docstring are presented, grouped by the type of file they are designed to work on. The examples assume that Plio is imported with:

```
>>> import plio
```

Planetary Datasets

io.io_spectral_profiler — Kaguya Spectral Profiler to DataFrame

The :mod:`io.io_spectral_profiler`module supports reading Kaguya SP data into a Pandas DataFrame.

New in version 0.1.0.

class plio.io.io_spectral_profiler.**Spectral_Profiler**(*input_data*, *label=None*, *cleaned=True*, *qa_threshold=2000*)

spectra
A pandas panel containing n individual spectra.

Type panel

ancillary_data
A pandas DataFrame of the parsed ancillary data (PVL label)

Type dataframe

label
The raw PVL label object

Type object

offsets
with key as the spectra index and value as the start byte offset

Type dict

open_browse(*extension='.jpg'*)
Attempt to open the browse image corresponding to the spc file

Parameters **extension** (*str*) – The file type extension to be added to the base name of the spc file.

io.io_tes — Reader for MGS TES Data

The io.io_tes module supports reading raw Mars Global Surveyor TES data into a Pandas DataFrame.

New in version 0.1.1.

class plio.io.io_tes.**Tes**(*input_data*, *var_file=None*, *data_set=None*)

spectra
A pandas panel containing n individual spectra.

Type panel

ancillary_data
A pandas DataFrame of the parsed ancillary data (PVL label)

Type dataframe

label
The raw PVL label object

Type object

join()
Given a list of Tes objects, merges them into a single dataframe using SPACECRAFT_CLOCK_START_COUNT (sclk_time) as the index.

Parameters **tes_data** (*iterable*) – A Python iterable of Tes objects

Returns

- *dataframe* – A pandas dataframe containing the merged data
- *outliers* – A list of Tes() objects containing the tables containing no matches

Generic Spatial Data

io.io_gdal — Geospatial Data Abstraction Library

The io.io_gdal and io.extract_metadata modules provide convenience wrappers to GDAL.

New in version 0.1.0.

class plio.io.io_gdal.**GeoDataset**(*file_name*)
Geospatial dataset object that represents.

Parameters **file_name** (*str*) – The name of the input image, including its full path.

base_name
The base name of the input image, extracted from the full path.

Type str

bounding_box
The bounding box of the image in lat/lon space

Type object

geotransform
Geotransform reference OGR object as an array of size 6 containing the affine transformation coefficients for transforming from raw sample/line to projected x/y. xproj = geotransform[0] + sample * geotransform[1] + line * geotransform[2] yproj = geotransform[3] + sample * geotransform[4] + line * geotransform[5]

Type object

geospatial_coordinate_system
Geospatial coordinate system OSR object.

Type object

latlon_extent
of two tuples containing the latitide/longitude boundaries. This list is in the form [(lowerlat, lowerlon), (upperlat, upperlon)].

Type list

pixel_width
The width of the image pixels (i.e. displacement in the x-direction). Note: This is the second value geotransform array.

Type float

pixel_height
The height of the image pixels (i.e. displacement in the y-direction). Note: This is the sixth (last) value geotransform array.

Type float

spatial_reference
Spatial reference system OSR object.

Type object

standard_parallels
of the standard parallels used by the map projection found in the metadata using the spatial reference for this GeoDataset.

Type list

unit_type
Name of the unit used by the raster, e.g. 'm' or 'ft'.

Type str

x_rotation
The geotransform coefficient that represents the rotation about the x-axis. Note: This is the third value geotransform array.

Type float

xy_extent
of two tuples containing the sample/line boundaries. The first value is the upper left corner of the upper left pixel and the second value is the lower right corner of the lower right pixel. This list is in the form [(minx, miny), (maxx, maxy)].

Type list

xy_corners
of tuple corner coordinates in the form: [upper left, lower left, lower right, upper right]

Type list

y_rotation
The geotransform coefficient that represents the rotation about the y-axis. Note: This is the fifth value geotransform array.

Type float

coordinate_transformation
The coordinate transformation from the spatial reference system to the geospatial coordinate system.

Type object

inverse_coordinate_transformation
The coordinate transformation from the geospatial coordinate system to the spatial reference system.

Type object

scale
The name and value of the linear projection units of the spatial reference system. This tuple is of type string/float of the form (unit name, value). To transform a linear distance to meters, multiply by this value. If no units are available ("Meters", 1) will be returned.

Type tuple

spheroid
The spheroid found in the metadata using the spatial reference system. This is of the form (semi-major, semi-minor, inverse flattening).

Type tuple

raster_size
The dimensions of the raster, i.e. (number of samples, number of lines).

Type tuple

central_meridian
The central meridian of the map projection from the metadata.

Type float

no_data_value
Special value used to indicate pixels that are not valid.

Type float

metadata
A dictionary of available image metadata

Type dict

footprint
An OGR footprint object

Type object

property geotransform
Where the array is in the form: [top left x, w-e pixel resolution, x-rotation, top left y, y-rotation, n-s pixel resolution]

latlon_to_pixel(*lat*, *lon*)
Convert from lat/lon space to pixel space (i.e. sample/line).

Parameters

- **lat** (*float*) – Latitude to be transformed.
- **lon** (*float*) – Longitude to be transformed.

Returns **x, y** – (Sample, line) position corresponding to the given (latitude, longitude).

Return type tuple

pixel_to_latlon(*x*, *y*)
Convert from pixel space (i.e. sample/line) to lat/lon space.

Parameters

- **x** (*float*) – x-coordinate to be transformed.
- **y** (*float*) – y-coordinate to be transformed.

Returns **lat, lon** – (Latitude, Longitude) corresponding to the given (x,y).

Return type tuple

read_array(*band=1*, *pixels=None*, *dtype=None*)
Extract the required data as a NumPy array

Parameters

- **band** (*int*) – The image band number to be extracted as a NumPy array. Default band=1.
- **pixels** (*list*) – [xstart, ystart, xcount, ycount]. Default pixels=None.
- **dtype** (*str*) – The NumPy dtype for the output array. Defaults to the band dtype.

Returns **array** – The dataset for the specified band.

Return type ndarray

plio.io.io_gdal.**array_to_raster**(*array*, *file_name*, *projection=None*, *geotransform=None*, *outformat='GTiff'*, *ndv=None*, *bittype='GDT_Float64'*)

Converts the given NumPy array to a raster format using the GeoDataset class.

Parameters

- **array** (*ndarray*) – The data to be written via GDAL
- **file_name** (*str*) – The output file PATH (relative or absolute)
- **projection** (*object*) – A GDAL readable projection object, WKT string, PROJ4 string, etc. Default: None
- **geotransform** (*object*) – A six parameter geotransformation Default:None.
- **outformat** (*str*) – A GDAL supported output format Default: 'GTiff'.
- **ndv** (*float*) – The no data value for the given band. Default: None.
- **bittype** (*str*) – A GDAL supported bittype, e.g. GDT_Int32 Default: GDT_Float64

Planetary Tool Formats

AutoCNET graph object (io.io_autocnetgraph)

Reading/Writing the Graph Object

New in version 0.1.1.

class plio.io.io_autocnetgraph.**NumpyEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

default(*obj*)

If input object is an ndarray it will be converted into a dict holding dtype, shape and the data, base64 encoded.

plio.io.io_autocnetgraph.**load**(*projectname*)

Read an AutoCNET project into memory.

Parameters **projectname** (*str*) – PATH to the project

Returns **cg** – AutoCNET CandidateGraph object

Return type object

plio.io.io_autocnetgraph.**save**(*network*, *projectname*)

Save an AutoCNet candiate graph to disk in a compressed file. The graph adjacency structure is stored as human readable JSON and all potentially large numpy arrays are stored as compressed binary. The project archive is a standard .zip file that can have any ending, e.g., <projectname>.project, <projectname>.zip, <projectname>.myname.

TODO: This func. writes a intermediary .npz to disk when saving. Can we write the .npz to memory?

Parameters

- **network** (*object*) – The AutoCNet Candidate Graph object
- **projectname** (*str*) – The PATH to the output file.

io.io_controlnetwork — ISIS Compatible Control Network Creation

The io.io_controlnetwork module supports the creation of control networks in ISIS format.

New in version 0.1.0.

class plio.io.io_controlnetwork.**IsisControlNetwork**(*data=None*, *index: Optional[Collection] = None*, *columns: Optional[Collection] = None*, *dtype: Optional[Union[ExtensionDtype, str, numpy.dtype, Type[Union[str, float, int, complex, bool, object]]]] = None*, *copy: bool = False*)

class plio.io.io_controlnetwork.**IsisStore**(*path*, *mode=None*, ***kwargs*)

Class to manage IO of an ISIS control network (version 2). .. attribute:: pointid

The current index to be assigned to newly added points

type int

create_buffer_header(*networkid*, *targetname*, *description*, *username*, *point_sizes*, *creation_date*, *modified_date*)

Create the Google Protocol Buffer header using the protobuf spec. :param networkid: The user defined identifier of this control network :type networkid: str :param targetname: The name of the target, e.g. Moon :type targetname: str :param description: A description for the network. :type description: str :param username: The name of the user / application that created the control network :type username: str :param point_sizes: of the point sizes for each point message :type point_sizes: list

Returns

- **header_message** (*str*) – The serialized message to write
- **header_message_size** (*int*) – The size of the serialized header, in bytes

create_points(*df*, *pointid_prefix*, *pointid_suffix*)

Step through a control network (C) and return protocol buffer point objects :param df: with the appropriate attributes: point_id, point_type, serial,

measure_type, x, y required. The entries in the list must support grouping by the point_id attribute.

Returns

- **point_messages** (*list*) – of serialized points buffers
- **point_sizes** (*list*) – of integer point sizes

create_pvl_header(*version*, *headerstartbyte*, *networkid*, *targetname*, *description*, *username*, *buffer_header_size*, *points_bytes*, *creation_date*, *modified_date*)

Create the PVL header object :param version: The current ISIS version to write, defaults to 2 :type version: int :param headerstartbyte: The seek offset that the protocol buffer header starts at :type headerstartbyte: int :param networkid: The name of the network :type networkid: str :param targetname: The name of the target, e.g. Moon :type targetname: str :param description: A description for the network. :type description: str :param username: The name of the user / application that created the control network :type username: str :param buffer_header_size: Total size of the header in bytes :type buffer_header_size: int :param points_bytes: The total number of bytes all points require :type points_bytes: int

Returns An ISIS compliant PVL header object

Return type object

read()
Given an ISIS store, read the underlying ISIS3 compatible control network and return an IsisControlNetwork dataframe.

write(*data*, *offset=0*)

Parameters

- **data** (*bytes*) – Encoded header to be written to the file
- **offset** (*int*) – The byte offset into the output binary

class plio.io.io_controlnetwork.**MeasureMessageType**(*value*)
An enum to mirror the ISIS3 MeasureLogData enum.

plio.io.io_controlnetwork.**write_filelist**(*lst*, *path='fromlist.lis'*)
Writes a filelist to a file so it can be used in ISIS3. :param lst: A list containing full paths to the images used, as strings. :type lst: list :param path: The name of the file to write out. Default: fromlist.lis :type path: str

io.isis_serial_number — Generate ISIS3 Compliant Serial Numbers

The io.isis_serial_number, using translation tables loaded using the io.io_db module supports generation of ISIS3 compliant serial numbers. These are necessary when creating ISIS3 compliant control networks.

New in version 0.1.0.

class plio.io.isis_serial_number.**SerialNumberDecoder**(*grammar=None*, *quantity_cls=None*)
A PVL Decoder class to handle cube label parsing for the purpose of creating a valid ISIS serial number. Inherits from the PVLDecoder in planetarypy's pvl module.

decode_simple_value(*value: str*)
Returns a Python object based on *value*, assuming that *value* can be decoded as a PVL Simple Value:

```
<Simple-Value> ::= (<Numeric> | <String>)

Modified from https://pvl.readthedocs.io/en/stable/_modules/pvl/decoder.html
↪#PVLDecoder.decode_simple_value
Modification entails stripping datetime from list of functions.
```

decode_unquoted_string(*value: str*) → str
Returns a Python str if *value* can be decoded as an unquoted string, based on this decoder's grammar. Raises a ValueError otherwise.

Modified from: https://pvl.readthedocs.io/en/stable/_modules/pvl/decoder.html#PVLDecoder.decode_unquoted_string Modification entails removal of decode_datetime call

plio.io.isis_serial_number.**generate_serial_number**(*label*)
Generate an ISIS compatible serial number using the ISIS team's translation files

Parameters **label** (*dict or str*) – A PVLModule object (dict) or string PATH to a file containing a PVL header

Returns The ISIS compatible serial number

Return type str

plio.io.isis_serial_number.**get_isis_translation**(*label*)
Compute the ISIS serial number for a given image using the input cube or the label extracted from the cube.

Parameters **label** (*dict or str*) – A PVL dict object or file name to extract the PVL object from

Returns **translation** – A PVLModule object containing the extracted translation file

Return type dict

class plio.io.io_db.**StringToMission**(*key*, *value*)
Table mapping for the ISIS mission name cleaner table

class plio.io.io_db.**Translations**(*mission*, *instrument*, *translation*)
Table mapping for the ISIS Translation file table

plio.io.io_db.**setup_db_session**(*db*)
Add a database session object to the root namespace

Parameters **db** (*str*) – Database name

Returns A SQLAlchemy session object

Return type object

io.io_krc — KRC Thermal Model Reader

The io.io_krc`module supports reading KRC thermal model Bin52 and `.tds files for both the floating point and double versions.

New in version 0.1.1.

class plio.io.io_krc.**ReadBin52**(*filename*, *headerlen=512*)
Class to read a bin 52 and organize the output

bin52data
The raw, n-dimensional KRC data

Type ndarray

casesize
The number of bytes in each KRC case

Type int

date
The date the read file was created

Type bytes

ddd
Raw temperature data

Type ndarray

ddd_pd
A pandas dataframe of surface temperature data

Type dataframe

filename
The name of the KRC file that was input

Type str

header
The KRC header unpack to a list of ints

Type list

headerlength
The length, in bytes, of the KRC header

Type int

ncases
The number of different cases the model was run for

Type int

ndim
TBD

Type int

ndx
The number of indices for a single KRC run

Type int

nhours
The number of hour bins per 24 Mars hours

Type int

nlats
The number of valid, non-zero latitude bins

Type int

nseasons
The number of seasons the model was run for

Type int

nvariables
The number of variables contained within the KRC lookup table

Type int

structdtype
Describing endianess and data type

Type str

temperature_data
A multi-indexed dataframe of surface temperatures

Type dataframe

transferedlayers
The number of KRC transfered layers

Type int

version
The krc verison used to create the file in the form [major, minor, release]

Type list

words_per_krc
The number of bytes per krc entry

Type int

definekrc(*what='KRC'*, *endianness='<'*)
Defines a custom binary data structure for the KRC files.

readbin5(*headerlen*)
Reads the type 52 file containing KRC output.

Tested with KRC version 2.2.2. Note that the output format can change

Parameters **(str) Full PATH to the file**(*filename*) –

readcase(*case=0*)
Read a single dimension (case) from a bin52 file.

Parameters **case**(*int*) – The case to be extracted

General IO

io.io_json — JSON Wrapper

The io.io_json module provides a convenience wrapper to access JSON .. versionadded:: 0.1.0

class plio.io.io_json.**NumpyEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

default(*obj*)
Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

plio.io.io_json.**read_json**(*inputfile*)
Read the input json file into a python dictionary.

Parameters **inputfile**(*str*) – PATH to the file on disk

Returns **jobs** – returns a dictionary

Return type dict

plio.io.io_json.**write_json**(*outdata*, *outputfile*)
Write a Python dictionary as a plain-text JSON file

Parameters

- **outdata**(*dict*) – The data structure to be serialized
- **outputfile**(*str*) – The file to write the data to.

io.io_yaml — Super Simple YAML Reader

The io.io_yaml module wraps the yaml library and provides a super simple reader.

New in version 0.1.0.

plio.io.io_yaml.**read_yaml**(*inputfile*)
Read the input yaml file into a python dictionary

Parameters **inputfile** (*str*) – PATH to the file on disk

Returns **ydict** – YAML file parsed to a Python dict

Return type dict

2.2.3 Generic Utilities

Generic utilities for supporting IO. The examples assume that Plio is imported with:

```
>>> import plio
```

File System and Data Structure Traversal

utils.utils — Utilities

The utils.utils module provides generic I/O utilities .. versionadded:: 0.1.0

plio.utils.utils.**convert_string_to_float**(*s*)
Attempt to convert a string to a float.

Parameters **s** (*str*) – The string to convert

Returns If successful, the converted value, else the argument is passed back out.

Return type float / str

plio.utils.utils.**create_dir**(*basedir=''*)
Create a unique, temporary directory in /tmp where processing will occur

Parameters **basedir** (*str*) – The PATH to create the temporary directory in.

plio.utils.utils.**delete_dir**(*dir*)
Delete a directory

Parameters **dir** (*str*) – Remove a directory

plio.utils.utils.**file_search**(*searchdir*, *searchstring*)
Recursively search for files in the specified directory

Parameters

- **searchdir** (*str*) – The directory to be searched
- **searchstring** (*str*) – The string to be searched for

Returns **filelist** – of files

Return type list

plio.utils.utils.**find_in_dict**(*obj*, *key*)
Recursively find an entry in a dictionary

Parameters

- **obj** (*dict*) – The dictionary to search
- **key** (*str*) – The key to find in the dictionary

Returns **item** – The value from the dictionary

Return type obj

plio.utils.utils.**find_nested_in_dict**(*data*, *key_list*)
Traverse a list of keys into a dict.

Parameters

- **data** (*dict*) – The dictionary to be traversed
- **key_list** (*list*) – The list of keys to be travered. Keys are traversed in the order they are entered in the list

Returns **value** – The value in the dict

Return type object

plio.utils.utils.**is_number**(*s*)
Check if an argument is convertable to a number

Parameters **s** (*object*) – The argument to check for conversion

Returns True if conversion is possible, otherwise False.

Return type bool

plio.utils.utils.**split_all_ext**(*path*)
General function for removeing all potential extensions from a file.

Parameters **path** (*str*) – Path or file name with potential extension

Returns **base** – Path or file name with all potnetial extensions removed

Return type str

plio.utils.utils.**xstr**(*s*)
Return an empty string if the input is a NoneType. Otherwise cast to string and return

Parameters **s** (*obj*) – An input object castable to a string

Returns The input object cast to a string

Return type str

Logging

utils.log — Wrapper for setting up a user defined logger

The utils.log module provides logging setup assistance .. versionadded:: 0.1.0

plio.utils.log.**setup_logging**(*path='/home/docs/checkouts/readthedocs.org/user_builds/plio/checkouts/latest/plio/example*
level='INFO', *env_key='LOG_CFG'*)
Read a log configuration file, written in JSON

Parameters

- **path** (*string*) – The path to the logging configuration file
- **level** (*object*) – The logger level at which to report

- **env_key** (*str*) – A potential environment variable where the user defaults logs

2.3 Developer Guide

2.4 Unlicense

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/>

PYTHON MODULE INDEX

p

INDEX

T

U

V

W

X

Y

Z