Pysis Documentation

Release 0.6.0

Trevor Olson

May 22, 2016

Contents

CHAPTER 1

Pysis

Toolkit for using USGS Isis in Python.

- Free software: BSD license
- Documentation: http://pysis.readthedocs.org.

1.1 How to install

At the command line:

```
$ easy_install pysis
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv pysis
$ pip install pysis
```

1.1.1 Dependencies

For working with ISIS commands, you must firts have USGS ISIS 3 installed on your machine. See the ISIS 3 installation guide for further instructions. Remember to set your environmental variables (see step 4 of USGS ISIS guide) so Pysis knows where your installation is.

1.2 Quickstart Guide

How to write ISIS 3 code in python using Pysis.

Using ISIS 3 at the command line you might want to run the following basic commands (examples for the MDIS camera on the MESSENGER mission):

```
mdis2isis from=filename.IMG to=filename.cub
spiceinit from=filename.cub
mdiscal from=filename.cub to=filename.cal.cub
```

using Pysis the syntax is:

```
from pysis.isis import mdis2isis, spiceinit, mdiscal
from pysis.util import file_variations

def calibrate_mids(img_name):
    (cub_name, cal_name) = file_variations(img_name, ['.cub', '.cal.cub'])

    mdis2isis(from_=img_name, to=cub_name)
    spiceinit(from_=cub_name)
    mdiscal(from_=cub_name, to=cal_name)
```

You will notice that we use the keyword *from_* when we call a command because *from* is a reserved word in python.

1.2.1 Numerical and String Arguments

Here is an example of the maptemplate and cam2map commands in Pysis:

```
from pysis import isis

isis.maptemplate(map='MDIS_eqr.map', projection='equirectangular',
                 clon=0.0, clat=0.0, resopt='mpp', resolution=1000,
                 rngopt='user', minlat=-10.0, maxlat=10.0, minlon=-10.0,
                 maxlon=10.0)

isis.cam2map(from_=cal_name, to=proj_name, pixres='map',
             map='MDIS_eqr.map',defaultrange='map')
```

1.2.2 Getting values from ISIS commands

Pysis commands will return the command's STDOUT as a byte string. If the command returns a nonzero exit code, a *ProcessError* will be thrown. This example command uses *getkey* to receive values from the label of an ISIS cube:

```
from pysis.isis import getkey

value = getkey(from_='W1467351325_4.map.cal.cub',
               keyword='minimumringradius', grp='mapping')
```

1.2.3 Catching ProcessingErrors

Pysis supports catching *ISIS* processing errors like so:

```
from pysis.exceptions import ProcessError
from pysis.isis import hi2sis

try:
    hi2isis(from_=filein, to=fileout)
except ProcessError as e:
    print("STDOUT:", e.stdout)
    print("STDERR:", e.stderr)
```

1.2.4 Multiprocessing Isis Commands with IsisPool

Pysis has built-in support to make multiprocessing isis commands simple. To run the above MDIS calibration script for multiple images in multiple processes we could rewrite the function as so:

```
from pysis import IsisPool
from pysis.util import ImageName

def calibrate_mdis(images):
    images = [ImageName(filename) for filename in images]

    with IsisPool() as isis_pool:
        for filename in images:
            isis_pool.mdis2isis(from_=filename.IMG, to=filename.cub)

    with IsisPool() as isis_pool:
        for filename in images:
            isis_pool.spiceinit(from_=filename.cub)

    with IsisPool() as isis_pool:
        for filename in images:
            isis_pool.mdiscal(from_=filename.cub, to=filename.cal.cub)
```

When using IsisPool we can't determine which order commands will be executed in so we much run each command for all the files as a group before moving to the next command and creating a new IsisPool.

CHAPTER 2

Contents

2.1 IsisPool

class pysis.**IsisPool**(*strict=False*, **args*, ***kwargs*)

Multiprocessing pool for ISIS commands.

Example for running the following isis script in parallel for a list of images.

On the command line:

```
mdis2isis from=filename.IMG to=filename.cub
spiceinit from=filename.cub
mdiscal from=filename.cub to=filename.cal.cub
```

With pysis:

```
from pysis import IsisPool
from pysis.util import ImageName

def calibrate_mdis(images):
    images = [ImageName(filename) for filename in images]

    with IsisPool() as isis_pool:
        for filename in images:
            isis_pool.mdis2isis(from_=filename.IMG, to=filename.cub)

    with IsisPool() as isis_pool:
        for filename in images:
            isis_pool.spiceinit(from_=filename.cub)

    with IsisPool() as isis_pool:
        for filename in images:
            isis_pool.mdiscal(from_=filename.cub, to=filename.cal.cub)
```

Parameters

- **strict** – when in strict mode, the isis pool will initialize its attributes with commands from the isis environment. Otherwise attributes are dynamically added as use
- ****kwargs** – additional parameters used to initialize the multiprocessing pool

close_and_wait()

Close the pool and wait for all commands to complete.

This will be automatically called if used as a context manager.

2.2 CubeFile

class pysis.**CubeFile**(*stream_or_fname*, *filename=None*)

A Isis Cube file reader.

apply_numpy_specials(*copy=True*)

Convert isis special pixel values to numpy special pixel values.

Isis	Numpy
Null	nan
Lrs	-inf
Lis	-inf
His	inf
Hrs	inf

Parameters **copy** – whether to apply the new special values to a copy of the pixel data and leave the orginial unaffected

Returns a numpy array with special values converted to numpy's nan, inf and -inf

apply_scaling(*copy=True*)

Scale pixel values to there true DN.

Parameters **copy** – whether to apply the scalling to a copy of the pixel data and leave the orginial unaffected

Returns a scalled version of the pixel data

bands

Number of image bands.

base

An additive factor by which to offset pixel DN.

data = None

A numpy array representing the image data.

dtype

Pixel data type.

filename = None

The filename if given, otherwise none.

get_image_array()

Create an array for use in making an image.

Creates a linear stretch of the image and scales it to between *0* and *255*. *Null*, *Lis* and *Lrs* pixels are set to *0*. *His* and *Hrs* pixels are set to *255*.

Usage:

```
from pysis import CubeFile
from PIL import Image

# Read in the image and create the image data
image = CubeFile.open('test.cub')
data = image.get_image_array()
```

```
    # Save the first band to a new file
    Image.fromarray(data[0]).save('test.png')
```

Returns A uint8 array of pixel values.

label = None
The parsed label header in dictionary form.

lines
Number of lines per band.

multiplier
A multiplicative factor by which to scale pixel DN.

classmethod open (*filename*)
Read an Isis Cube file from disk.

Parameters **filename** – name of file to read as an isis file

samples
Number of samples per line.

shape
Tuple of images bands, lines and samples.

size
Total number of pixels.

specials_mask()
Create a pixel map for special pixels.

Returns an array where the value is *False* if the pixel is special and *True* otherwise

start_byte
Index of the start of the image data (zero indexed).

tile_lines
Number of lines per tile.

tile_samples
Number of samples per tile.

2.3 Cube Label Parsing

2.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/wtolson/pysis/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Pysis could always use more documentation, whether as part of the official Pysis docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/wtolson/pysis/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.4.2 Get Started!

Ready to contribute? Here's how to set up *pysis* for local development.

1. Fork the *pysis* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pysis.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pysis
$ cd pysis/
$ pip install -r requirements.txt
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make lint
$ make test
$ make test-all
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

2.4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, 3.4, and for PyPy. Check https://travis-ci.org/wtolson/pysis/pull_requests and make sure that the tests pass for all supported Python versions.

2.4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_bining
```

2.5 Credits

2.5.1 Development Lead

- Trevor Olson <trevor@heytrevor.com>

2.5.2 Contributors

- Sarah Braden <braden.sarah@gmail.com>
- Michael Aye <kmichael.aye@gmail.com>

2.6 History

2.6.1 0.6.0 (2016-05-22)

- Support filenames in image constructor. (thanks @michaelaye)

- Use pvl for label decoding.

2.6.2 0.5.2 (2015-05-30)

- Relicense as BSD.

2.6.3 0.5.1 (2015-05-18)

- Add support for line comments.
- Fix packages in setup.py. (thanks @michaelaye)

2.6.4 0.5.0 (2015-04-18)

- Add support for python 2.6/3.3/3.4
- Simplified command api.
- Labels package now shares the json module api.
- Label parser now fully conforms to the PVL spec.
- Add label encoder.

2.6.5 0.4.0 (2015-03-21)

- First release on PyPI.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search