pvl Documentation

Release 0.2.0

William Trevor Olson

May 29, 2017

Contents

CHAPTER 1

pvl

Python implementation of PVL (Parameter Value Language)

- Free software: BSD license
- Documentation: http://pvl.readthedocs.org.
- Support for Python 2, 3 and pypi.
- Proudly part of the PlanetaryPy Toolkit

PVL is a markup language, similar to xml, commonly employed for entries in the Planetary Database System used by NASA to store mission data, among other uses. This package supports both encoding a decoding a superset of PVL, including the USGS Isis Cube Label and NASA PDS 3 Label dialects.

Installation

Can either install with pip or with conda.

To install with pip, at the command line:

```
$ pip install pvl
```

Directions for installing with conda-forge:

Installing pvl from the conda-forge channel can be achieved by adding conda-forge to your channels with:

```
conda config --add channels conda-forge
```

Once the conda-forge channel has been enabled, pvl can be installed with:

```
conda install pvl
```

It is possible to list all of the versions of pvl available on your platform with:

```
conda search pvl --channel conda-forge
```

Basic Usage

pvl exposes an API familiar to users of the standard library json module.

Decoding is primarily done through `pvl.load` for file like objects and `pvl.loads` for strings:

```
>>> import pvl
>>> module = pvl.loads("""
...     foo = bar
...     items = (1, 2, 3)
...     END
... """)
>>> print module
PVLModule([
  (u'foo', u'bar')
  (u'items', [1, 2, 3])
])
>>> print module['foo']
bar
```

You may also use `pvl.load` to read a label directly from an image:

```
>>> import pvl
>>> label = pvl.load('pattern.cub')
>>> print label
PVLModule([
  (u'IsisCube',
   PVLObject([
    (u'Core',
     PVLObject([
      (u'StartByte', 65537)
      (u'Format', u'Tile')
# output truncated...
>>> print label['IsisCube']['Core']['StartByte']
65537
```

Similarly, encoding pvl modules is done through `pvl.dump` and `pvl.dumps`:

```
>>> import pvl
>>> print pvl.dumps({
...     'foo': 'bar',
...     'items': [1, 2, 3]
... })
items = (1, 2, 3)
foo = bar
END
```

`PVLModule` objects may also be pragmatically built up to control the order of parameters as well as duplicate keys:

```
>>> import pvl
>>> module = pvl.PVLModule({'foo': 'bar'})
>>> module.append('items', [1, 2, 3])
>>> print pvl.dumps(module)
foo = bar
```

```
items = (1, 2, 3)
END
```

A `PVLModule` is a `dict` like container that preserves ordering as well as allows multiple values for the same key. It provides a similar similar semantics to a `list` of key/value `tuples` but with `dict` style access:

```
>>> import pvl
>>> module = pvl.PVLModule([
...     ('foo', 'bar'),
...     ('items', [1, 2, 3]),
...     ('foo', 'remember me?'),
... ])
>>> print module['foo']
bar
>>> print module.getlist('foo')
['bar', 'remember me?']
>>> print module.items()
[('foo', 'bar'), ('items', [1, 2, 3]), ('foo', u'remember me?')]
>>> print pvl.dumps(module)
foo = bar
items = (1, 2, 3)
foo = "remember me?"
END
```

For more information on custom serilization and deseralization see the full documentation.

Contributing

Feedback, issues, and contributions are always gratefully welcomed. See the contributing guide for details on how to help and setup a development environment.

CHAPTER 2

Contents

Parsing Label

Table of Contents

pvl.load

This module parses a PVL compliant label from a stream and returns a dictionary containing information from the label. This documentation will explain how to use the module as well as some sample code to use the module efficiently.

Simple Use

How to use Module:

```
>>> import pvl
>>> img = 'path\to\img_file.ext'
>>> pvl.load(img)
```

```
Image Label Dictionary
>>> pvl.load(img)['key']
Value

>>> import pvl
>>> img = 'path\to\img_file.ext'
>>> with open(img, 'r+') as r:
        print pvl.load(r)['key']
Value
```

Parameters

Must be a string containing the absolute or relative path to an image file. The keyword argument `strict=False` can be used to parse image files with labels that are not PVL compliant.

Detailed Use

To view the image label as a dictionary:

```
>>> import pvl
>>> img = '1p205337908eff73u6p2438r2m1.img'
>>> pvl.load(img)
Label([
 (u'PDS_VERSION_ID', u'PDS3')
 (u'RECORD_TYPE', u'FIXED_LENGTH')
 (u'RECORD_BYTES', 2048)
 (u'FILE_RECORDS', 1043)
 (u'LABEL_RECORDS', 12)
 (u'^IMAGE_HEADER', 13)
 (u'^IMAGE', 20)
```

Not all image labels are formatted the same so different labels will have different information that you can obtain. To view what information you can extract use the `.keys()` function:

```
>>> import pvl
>>> img = '1p205337908eff73u6p2438r2m1.img'
>>> lbl = pvl.load(img)
>>> lbl.keys()
[u'INSTRUMENT_ID',
u'SUBFRAME_REQUEST_PARMS',
u'SOLAR_LONGITUDE',
u'PRODUCER_INSTITUTION_NAME',
u'PRODUCT_ID',
u'PLANET_DAY_NUMBER',
u'PROCESSING_HISTORY_TEXT',]
```

Now you can just copy and paste from this list:

```
>>> lbl['INSTRUMENT_ID']
u'PANCAM_RIGHT'
```

The list `.keys()` returns is out of order, to see the keys in the order of the dictionary use `.items()` function:

```
>>> import pvl
>>> img = '1p205337908eff73u6p2438r2m1.img'
```

```
>>> for item in pvl.load(img).items():
        print item[0]
PDS_VERSION_ID
RECORD_TYPE
RECORD_BYTES
FILE_RECORDS
LABEL_RECORDS
^IMAGE_HEADER
^IMAGE
DATA_SET_ID
```

We can take advantage of the fact `.items()` returns a list in order and use the index number of the key instead of copying and pasting. This will make extracting more than one piece of information at time more convenient. For example, if you want to print out the first 5 pieces of information:

```
>>> import pvl
>>> img = '1p205337908eff73u6p2438r2m1.img'
>>> keys = pvl.load(img).items()
>>> for n in range(0,5):
       print keys[n][0],keys[n][1]
0PDS_VERSION_ID PDS3
RECORD_TYPE FIXED_LENGTH
RECORD_BYTES 2048
FILE_RECORDS 1043
LABEL_RECORDS 12
```

Some values have sub dictionaries. You can access those by:

```
>>> print pvl.load(img)[keys[1]].keys()
[u'LINE_SAMPLES', u'FIRST_LINE_SAMPLE', u'LINES', u'GROUP_APPLICABILITY_FLAG', u
↪'SUBFRAME_TYPE', u'SOURCE_ID', u'FIRST_LINE']
>>> print pvl.load(img)[keys[1]]['SOURCE_ID']
GROUND COMMANDED
```

`pvl.load` also works for Isis Cube files:

```
>>> import pvl
>>> img = 'pattern.cub'
>>> keys = pvl.load(img).keys()
>>> for n, item in enumerate(keys):
       print n, item
0 Label
1 IsisCube
>>> print pvl.load(img)[keys[0]]
LabelObject([
 (u'Bytes', 65536)
])
>>> print pvl.load(img)[keys[0]]['Bytes']
65536
```

Another way of using `pvl.load` is to use python's `with open()` command. Otherwise the using this method is very similar to using the methods described above:

```
>>> import pvl
>>> with open('pattern.cub','r') as r:
       print pvl.load(r)['Label']['Bytes']
65536
```

pvl.loads

This module parses a PVL compliant label from a string and returns a dictionary containing information from the label. This documentation will explain how to use the module as well as some sample code to use the module efficiently.

Simple Use

How to use Module:

```
>>> import pvl
>>> img = """String
containing the label

of the image"""
>>> pvl.loads(img).keys()
>>> pvl.loads(img)['key']
value
```

Parameters

Must be a string containing the PVL label. The keyword argument `strict=False` can be used to parse labels that are not PVL compliant.

Detailed Use

To view the image label dictionary:

```
>>> import pvl
>>> string = """Object = IsisCube
  Object = Core
    StartByte   = 65537
    Format      = Tile
    TileSamples = 128
    TileLines   = 128

  End_Object
End_Object

Object = Label
  Bytes = 65536
End_Object
End"""
>>> print pvl.loads(string)
Label([
 (u'IsisCube',
  LabelObject([
   (u'Core',
    LabelObject([
     (u'StartByte', 65537)
     (u'Format', u'Tile')
     (u'TileSamples', 128)
     (u'TileLines', 128)
   ]))
 ]))
```

```
 (u'Label', LabelObject([
   (u'Bytes', 65536)
 ]))
])
```

To view the keys available:

```
>>> print pvl.loads(string).keys()
[u'Label', u'IsisCube']
```

And to see the information contained in the keys:

```
>>> print pvl.loads(string)['Label']
LabelObject([
 (u'Bytes', 65536)
])
```

And what is in the subdirectory:

```
>>> print pvl.loads(string)['Label']['Bytes']
65536
```

Load a non-pvl compliant label using the `strict=False` keyword argument. The only type of of non-pvl compliant labels are those where there is a parameter without a corresponding value. Other types of broken labels are not supported:

```
>>> string = """
Object = Label
  A =
End_Object
End"""
```

```
>>> print pvl.loads(string, strict=False)
PVLModule([
 (u'Label',
  PVLObject([
   (u'A', EmptyValueAtLine(3 does not have a value. Treat as an empty␣
↪string))
 ]))
```

])

Encoding Label

This documentation explains how you can use `pvl.dump` and `pvl.dumps` so you can change, add, and/or write out the label to another file. This documentation assumes that you know what pvl.load and pvl.loads are and how to use them. Read the documentation on pvl.load and pvl.loads if you do not. The examples use an IsisCube image label format, however this module can write/alter any PVL compliant label.

Table of Contents

pvl.dump

This module allows you to modify an existing image label and then write the new label to the file or to a new file.

Simple Use

How to use module:

```
>>> import pvl
>>> img = 'path/to/image.ext'
>>> label = pvl.load(img)
# Change information
>>> label['Existing_Key'] = 'Different_Value'
# Add Information
>>> label['New_Key'] = 'New_Value'
# Write out new label to file
>>> with open(img,'w') as stream:
        pvl.dump(label,stream)
```

Parameters

Must include a label that is a dictionary and a file to write the label to.

Changing A Key

In order to change the value assigned to a key:

```
>>> import pvl
>>> img = 'pattern.cub'
>>> label = pvl.load(img)
>>> print label['IsisCube']['Core']['Format']
Tile
# Changing key 'Format' to 'Changed_Value'
>>> label['IsisCube']['Core']['Format'] = 'Changed_Value'
# Writing out file with new value
>>> with open(img,'w') as stream:
        pvl.dump(label,stream)
# Showing the value changed in the file
>>> new_label = pvl.load(img)
```

```
>>> print new_label['IsisCube']['Core']['Format']
Changed_Value
```

Adding A Key

In order to add a new key and value to a label:

```
>>> import pvl
>>> img = 'pattern.cub'
>>> label = pvl.load(img)
# Adding a new key and value
>>> label['New_Key'] = 'New_Value'
# Adding a new key and value to a sub group
>>> label['IsisCube']['Core']['New_SubKey'] = 'New_SubValue'
# Writing new keys and values to file
>>> with open(img,'w') as stream:
        pvl.dump(label,stream)
# Showing the value changed in the file
>>> new_label = pvl.load(img)
>>>print new_label['New_Key']
New_Value
>>> print new_label['IsisCube']['Core']['New_SubKey']
New_SubValue
```

Writing to a Different File

If you do not want to overwrite the existing file and make a detached label:

```
>>> import pvl
>>> img = 'pattern.cub'
>>> label = pvl.load(img)
>>> label['IsisCube']['Core']['Format'] = 'Changed_Value'
# Creating new file with same name but with .lbl extension
>>> new_name = img.replace('.img','.lbl')
>>> print new_name
pattern.lbl
>>> with open(new_name, 'w') as stream:
        pvl.dump(label,new_name)
>>> new_label = pvl.load(new_name)
>>> print new_label['IsisCube']['Core']['Format']
Changed_Value
```

pvl.dumps

This module takes a label dictionary and converts the dictionary to a string.

Simple Use

How to use module:

```
>>> import pvl
>>> img = 'path/to/image.ext'
>>> label = pvl.load(img)
# Change information
>>> label['Existing_Key'] = 'Different_Value'
# Add Information
>>> label['New_Key'] = 'New_Value'
# Convert to a string
>>> label_string = pvl.dumps(label)
>>> print label_String
Existing_Key = Different_Value
New_Key = New_Value
```

Parameters

Must include a label as a dictionary.

Example

```
>>> import pvl
>>> img = 'pattern.cub'
>>> label = pvl.load(img)
>>> label['New_Key'] = 'New_Value'
>>> label_string = pvl.dumps(label)
>>> print label_string
Object = IsisCube
 Object = Core
   StartByte = 65537
   Format = Tile
   TileSamples = 128
   TileLines = 128
   Group = Dimensions
     Samples = 90
     Lines = 90
     Bands = 1
   End_Group
End_Object
New_Key = New_Value
End
```

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:

Types of Contributions

Report Bugs

Report bugs at https://github.com/planetarypy/pvl/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

pvl could always use more documentation, whether as part of the official pvl 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/planetarypy/pvl/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 :)

Get Started!

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

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

```
$ git clone git@github.com:your_name_here/pvl.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 pvl
$ cd pvl/
$ 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.

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/planetarypy/pvl/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests/test_pvl.py
```

Credits

Development Lead

- Trevor Olson <trevor@heytrevor.com>

Contributors

- Sarah Braden <braden.sarah@gmail.com>
- Michael Aye <kmichael.aye@gmail.com>
- Austin Godber <godber@uberhip.com>
- Perry Vargas <perrybvargas@gmail.com>

History

0.3.0 (2017-06-28)

- Create methods to add items to the label
- Give user option to allow the parser to succeed in parsing broken labels

0.2.0 (2015-08-13)

- Drastically increase test coverage.
- Lots of bug fixes.
- Add Cube and PDS encoders.
- Cleanup README.
- Use pvl specification terminology.
- Added element access by index and slice.

0.1.1 (2015-06-01)

- Fixed issue with reading Pancam PDS Products.

0.1.0 (2015-05-30)

- First release on PyPI.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search