Python Parameter Value Language (PVL) Library Ross A. Beyer[1,2]. [1]SETI Institute, (rbeyer@seti.org), [2]NASA Ames Research Center

The Parameter Value Language (PVL) is a markup language [1], similar to XML, JSON, and others. It is commonly employed for data archived under version three of the Planetary Data System (PDS) used by NASA to archive mission data, among other uses. Although this is often referred to as PVL, in truth, the 'PVL-text' archived under PDS3 is a restricted version of the Object Description Language (ODL)[2] which is a subset of PVL.

This Python PVL package (*pvl*) supports both encoding and decoding a variety of PVL 'flavors' including PVL itself, ODL, NASA PDS 3 Labels, and USGS ISIS Cube Labels. It is also a PlanetaryPy [3] affiliated package.

The *pvl* GitHub repository can be found here: `https://github.com/planetarypy/pvl`, and the documentation can be found here: `https://pvl.readthedocs.io/`.

History: The Python *pvl* library was initially written by Trevor Olson and released as verison 0.1.0 in 2015. Trevor and others continued to work on this version of *pvl* for the next two years through version 0.3.0 in 2017. It was a very stable library, and was used by many Python developers in planetary sciences.

Over the winter of 2019-2020, as the PlanetaryPy Project began to revive itself, the codebase was revisited, and engineering was performed to modernize the Python under the hood to make the library more robust. Since we knew there was so much existing code that already used *pvl* we were (hopefully) careful not to break anything that was working, but also implement some new features. In some cases this ended up breaking some downstream code, but that was because that code had to patch pre-1.0 *pvl* to get it to work, and now their additional patching wasn't necessary with the new architecture and features.

All of this work led to the 1.0.0 release in August 2020, and *pvl* is now at verison 1.2.0 as of March 2021.

Basic Usage: The *pvl* library uses a pattern or API familiar to users of the Python standard library `json` module. When PVL-text in the form of a file or a string is "decoded" the *pvl* library returns a dict-like container that preserves ordering as well as allows multiple values for the same key. It provides similar semantics to a list of key/value tuples but with dict-style access.

Decoding is primarily done through `pvl.load()` for file-like objects and `pvl.loads()` for strings, and even `pvl.loadu()` for reading PVL-text from a URL resource.

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

You may also use `pvl.load()` to read PVL-text directly from an ISIS image which begins with PVL text.

Similarly, encoding Python dict-like objects as PVL-text is done through `pvl.dump()` and `pvl.dumps()`.

The intent is for the loaders (`pvl.load()`, `pvl.loads()`, and `pvl.loadu()`) to be permissive, and attempt to parse as wide a variety of PVL-text as possible, including some kinds of 'broken' PVL-text (that doesn't conform to any known standard, but is close enough).

On the flip side, when dumping a Python object to PVL-text (via `pvl.dumps()` and `pvl.dump()`), the library will default to writing PDS3-Standards-compliant PVL-text, which in some ways is the most restrictive, but the most likely version of PVL-text that you need if you're writing it out (this is different from pre-1.0 versions of *pvl*).

You can change this behavior by giving different parameters to the loaders and dumpers that define the grammar of the PVL-text that you're interested in, as well as custom parsers, decoders, and encoders.

Installation: The *pvl* module can be easily installed via `pip install pvl` or `conda install pvl`.

Returned Values: In general, the *pvl* loaders return a dict-like with objects from the Python standard library. The PVL specifications have patterns that allow the loaders to return data as Python strings, sets, lists, ints, and floats. However, there is at least one PVL-text construct that Python doesn't have a standard library object for, and that is quantities. PVL-text can provide a value with an associated unit. By default, the *pvl* loaders return that as a `pvl.collections.Quantity` that is just a Python `namedtuple` with a `value` and a `unit` parameter.

However, there are 3rd party Python packages

like *astropy* and *pint* which have very smart mechanisms for "quantity" objects. The *pvl* package allows you to specify an alternate to the default `pvl.collections.Quantity` object via the `quantity_cls` argument to a decoder constructor (more information can be found in the *pvl* documentation), and when it comes across a PVL "quantity" (number with a unit) it will return them in the dict-like as `astropy.units.Quantity` or `pint.Quantity` objects, as requested.

Similarly, converting all "real" numbers from the PVL-text to Python `float` objects may alter their precision, and so there is also a `real_cls` argument to the decoders that could take an object like Python's `decimal.Decimal` object, such that precision of the value in the PVL-text can be preserved.

Utility Programs: The *pvl* library also provides some command-line utility programs to work with PVL-text.

The `pvl_translate` program will read a file with PVL-text (any of the kinds of files that `pvl.load()` reads, including ISIS files) or STDIN and will convert that PVL-text to a particular PVL dialect of your choosing (or JSON). It is not particularly robust, and if it cannot make simple conversions, it will raise errors.

The `pvl_validate` program will read a file with PVL-text (any of the kinds of files that `pvl.load()` reads) and will report on which of the various PVL dialects were able to load that PVL-text, and then also reports on whether the *pvl* library can encode the Python Objects back out to PVL-text. You can imagine some PVL-text that could be loaded, but is not able to be written out in a particular strict PVL dialect (like PDS3 labels).

Conclusion: The change to PDS4's XML-based label model may reduce the need for PVL-text parsing (which is good, PVL-text isn't a particularly great markup language), but there is still a tremendous amount of data out there in a PVL-text structure, and hopefully this Python *pvl* library will help you work with that data.

References: **[1]** Consultative Committee for Space Data Systems. *Parameter Value Language Specification (CCSD0006 and CCSD0008)*. Blue Book. June 2000. URL: https://public.ccsds.org/Pubs/641x0b2.pdf. **[2]** "Object Description Language (ODL) Specification and Usage". In: *Planetary Data System Standards Reference*. Ed. by Planetary Data System. 3.8. Jet Propulsion Laboratory, California Institute of Technology, 2009. Chap. 12. URL: https://pds.nasa.gov/datastandards/pds3/standards/sr/Chapter12.pdf. **[3]** K. Michael Aye et al. "The PlanetaryPy Project". In: *Planetary Data Workshop*. 2021.