pyaavso Documentation

Release 0.1.5

Author

Jun 01, 2018

Contents

pyaavso is a Python library for working with AAVSO (American Association of Variable Star Observers) data. The library is compatible with both Python 2.7 and 3.3+.

CHAPTER 1

Features

- reading and writing variable star observations in AAVSO's Visual File Format
- downloading all observation data for a given observer

CHAPTER 2

Installation

Use `pip` to install latest release available at PyPI:

```
pip install pyaavso
```

CHAPTER 3

Usage

The following code uses `VisualFormatWriter` to report a single observation of **SS Cyg** between the outbursts.

```
>>> from pyaavso.formats import VisualFormatWriter
>>> observer_code = 'XYZ'
>>> with open('data.txt', 'wb') as fp:
...     writer = VisualFormatWriter(fp, observer_code)
...     writer.writerow({
...         'name': 'SS CYG',
...         'date': '2450702.1234',
...         'magnitude': '<11.0',
...         'comp1': '110',
...         'chart': '070613',
...     })
```

The `data.txt` file can be now submitted to AAVSO.

CHAPTER 4

Resources

- Documentation
- Issue tracker
- CI server

CHAPTER 5

Author

- Zbigniew Siciarz (zbigniew at siciarz dot net)

CHAPTER 6

License

pyaavso is free software, licensed under the MIT/X11 License. A copy of the license is provided with the source code in the LICENSE file.

CHAPTER 7

Gittip

Like this project? You can support it via Gittip!

CHAPTER 8

Documentation

8.1 Usage

8.1.1 Basic usage

The following example shows how to use **pyaavso** to download all observations by a given observer.

```
from __future__ import unicode_literals, print_function

import sys
import logging

from pyaavso.formats import VisualFormatWriter
from pyaavso.utils import download_observations


if __name__ == '__main__':
    # configure logging so we can see some informational output
    logger = logging.getLogger('pyaavso.utils')
    logger.setLevel(logging.DEBUG)
    logger.addHandler(logging.StreamHandler())
    try:
        observer_code = sys.argv[1]
    except IndexError:
        print('Usage: python download_observations.py <OBSERVER_CODE>')
    else:
        observations = download_observations(observer_code)
        print('All done.\nDownloaded %d observations.' % len(observations))
        filename = '%s.txt' % observer_code
        with open(filename, 'wb') as fp:
            writer = VisualFormatWriter(fp, observer_code)
            for observation in observations:
                writer.writerow(observation)
        print('Observations written to file %s.' % filename)
```

8.2 API reference

8.2.1 pyaavso Package

pyaavso.__init__.**get_version**()

8.2.2 Subpackages

formats Package

visual Module

exception pyaavso.formats.visual.**FormatException**
Raised when the data does not conform to AAVSO format specification.

class pyaavso.formats.visual.**VisualFormatReader**(*fp*)
A class to read observations from file in AAVSO Visual File Format.

The reader API is also based on `csv` Python module. You create a reader instance by passing a file-like object in the constructor. This will read all the data and validate required headers. Then the reader object can be used to iterate over observation data.

A short example:

```
>>> with open('data.txt', 'rb') as fp:
...     reader = VisualFormatReader(fp)
...     for observation in reader:
...         print '%(name)s %(magnitude)s' % observation
SS Cyg 10.0
RZ Cas 6.4
```

Creates the reader instance and reads file headers.

Raises `FormatException` when any of the required headers could not be found in input. The following header parameters are required:

- *TYPE* - always 'Visual', yet must be specified in file
- *OBSCODE* - official AAVSO-assigned observer code
- *DATE* - date format, must be one of 'JD' or 'Excel'

Other headers described in AAVSO specification have reasonable default values, eg. the default delimiter is a comma, when not specified in headers. Without the *OBSTYPE* header, observations are assumed to be visual.

Parameters **fp** – a file-like object from which data will be read

classmethod **row_to_dict**(*row*)
Converts a raw input record to a dictionary of observation data.

Parameters

- **cls** – current class
- **row** – a single observation as a list or tuple

class pyaavso.formats.visual.**VisualFormatWriter**(*fp*, *observer_code*, *delimiter=','*, *date_format=u'JD'*, *obstype=u'Visual'*)
A class responsible for writing observation data in AAVSO Visual File Format.

The API here mimics the `csv` module in Python standard library.

To write your observations into the data file, you first need to create the writer, passing to it the destination file and your observer code. Then call *`writerow()`* for every single observation, for example:

```
>>> with open('data.txt', 'wb') as fp:
...     writer = VisualFormatWriter(fp, 'XYZ')
...     writer.writerow({
...         'name': 'SS CYG',
...         'date': '2450702.1234',
...         'magnitude': '<11.1',
...         'comment_code': '',
...         'comp1': '110',
...         'comp2': '113',
...         'chart': '070613',
...         'notes': 'This is a test',
...     })
```

Creates the writer which will write observations into the file-like object given in first parameter. The only other required parameter is the official AAVSO-assigned observer code.

Parameters

- **fp** – file-like object to write observations into
- **observer_code** – AAVSO observer code
- **delimiter** – field delimiter (set as DELIM header)
- **date_format** – observation date format (one of *JD* or *Excel*)
- **obstype** – observation type (*Visual* or *PTG*)

classmethod dict_to_row(*observation_data*)

Takes a dictionary of observation data and converts it to a list of fields according to AAVSO visual format specification.

Parameters

- **cls** – current class
- **observation_data** – a single observation as a dictionary

writerow(*observation_data*)

Writes a single observation to the output file.

If the `observation_data` parameter is a dictionary, it is converted to a list to keep a consisted field order (as described in format specification). Otherwise it is assumed that the data is a raw record ready to be written to file.

Parameters **observation_data** – a single observation as a dictionary or list

parsers Package

webobs Module

class pyaavso.parsers.webobs.**WebObsResultsParser**(*html_source*)

Parser for WebObs search results page.

The parser reads an HTML page with search results (presented as a table) and parses the table into a list of observations.

Creates the parser and feeds it source code of the page.

get_observations()
Parses the HTML table into a list of dictionaries, each of which represents a single observation.

utils Module

pyaavso.utils.**download_observations**(*observer_code*)
Downloads all variable star observations by a given observer.

Performs a series of HTTP requests to AAVSO's WebObs search and downloads the results page by page. Each page is then passed to *WebObsResultsParser* and parse results are added to the final observation list.

8.3 Development

8.3.1 Feature roadmap

- implement the Extended File Format
- add VSX search
- add programmatic access to lightcurves generated by LCG
- create an API client for Variable Star Plotter

8.3.2 Contributing

Looking to improve pyaavso? Here's how you can help.

Report issues

If you think you found a **bug** in pyaavso or have a **feature request**, feel free to file an issue. We rely on GitHub for issue tracking. Please, search through existing issues before you report a new one; perhaps your problem was already discussed or fixed.

When submitting an issue, please include the following:

- problem description
- steps to reproduce (a smallest possible code example that reproduces the issue would be most welcome!)
- expected outcome
- actual outcome
- platform information: your operating system, Python version, etc.
- any other relevant information

Contribute code

Contributions to pyaavso source code are accepted as **pull requests** on GitHub. Fork the project, work on it in your repository and when you think your patch is ready, send us a pull request.

License

By contributing your code, you agree to license your contribution under the terms of MIT license (see `LICENSE` file for details).

8.4 Changelog

8.4.1 0.1.5

- fixed VisualFormatReader bug on Python 3 when input is bytes, not string

8.4.2 0.1.4

- minor packaging and documentation fixes

8.4.3 0.1.3

- Python 3.4 compatibility
- more specific Python version classifiers in setup.py

8.4.4 0.1.2

- added wheel distribution

8.4.5 0.1.1

- less memory-hungry VisualFormatReader

8.4.6 0.1.0

- initial release

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index