pysatMadrigal Documentation

Release 0.0.4-alpha

Burrell, Angeline G., Klenzing, Jeff, Stoneback, Russell, Pembroke

Jun 11, 2021

CONTENTS

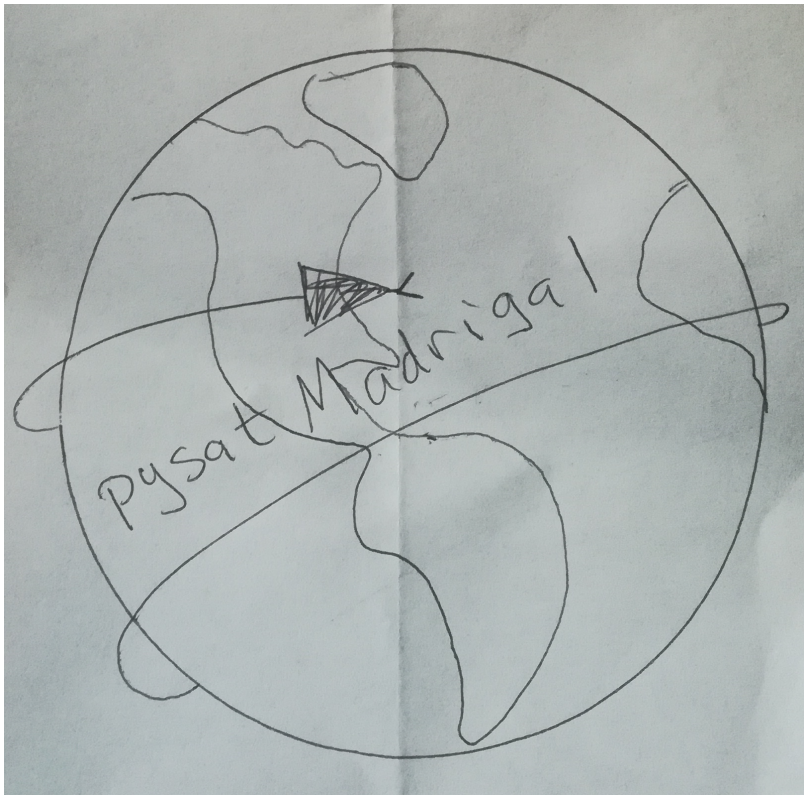
This documentation describes the pysatMadrigal module, which contains routines to download, load, and support analysis for data sets available at the Madrigal data base as pysat.Instrument objects.

CHAPTER

ONE

OVERVIEW

The CEDAR Madrigal database houses several ground- and space-based data sets for instruments that provide upper atmospheric, scientific observations.



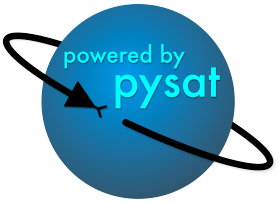
CHAPTER

TWO

INSTALLATION

The following instructions will allow you to install pysatMadrigal.

2.1 Prerequisites



pysatMadrigal uses common Python modules, as well as modules developed by and for the Space Physics community. This module officially supports Python 3.6+.

Common modules	Community modules
h5py	madrigalWeb
numpy	pysat
pandas	
xarray	

2.2 Installation Options

You may either install pysatMadrigal via pip or by cloning the git repository

1. Install from pip

```
pip install pysatMadrigal
```

2. Clone the git repository and use the `setup.py` file to install

```
git clone https://github.com/pysat/pysatMadrigal.git

# Install on the system (root privileges required)
sudo python3 setup.py install

# Install at the user level
python3 setup.py install --user
```

(continues on next page)

(continued from previous page)

```
# Install at the user level with the intent to develop locally
python3 setup.py develop --user
```

CHAPTER

THREE

CITATION GUIDELINES

When publishing work that uses pysatMadrigal, please cite the package and any package it depends on that plays an important role in your analysis. Specifying which version of pysatMadrigal used will also improve the reproducibility of your presented results.

3.1 pysatMadrigal

- Burrell, A. G., et al. (2020). pysat/pysatMadrigal (Version 0.0.3). Zenodo. doi:10.5281/zenodo.3824979.

```
@Misc{pysatMadrigal,
      author = {Burrell, A. G. and Klenzing, J. H. and Stoneback, R.
                and Pembroke, A.},
      title  = {pysat/pysatMadrigal},
      year   = {2020},
      date   = {2020-05-13},
      url    = {https://github.com/pysat/pysatMadrigal},
      doi    = {10.5281/zenodo.3824979},
      publisher = {Zenodo},
      version = {v0.0.3},
      }
```

3.2 MadrigalWeb

pysatMadrigal uses MadrigalWeb to access the Madrigal database. This package is described in the following journal article.

- Burrell A. G., et al. (2018). Snakes on a Spaceship - An Overview of Python in Heliophysics. Journal of Geophysical Research: Space Physics, 123, doi:10.1029/2018ja025877.

CHAPTER

FOUR

SUPPORTED INSTRUMENTS

4.1 DMSP_IVM

Supports the Defense Meteorological Satelitte Program (DMSP) Ion Velocity Meter (IVM) Madrigal data.

4.2 GNSS_TEC

The Global Navigation Satellite System (GNSS) Total Electron Content (TEC) provides a measure of column plasma density over the globle. The Madrigal TEC is provided by MIT Haystack.

4.3 JRO_ISR

The incoherent scatter radar (ISR) at the Jicamarca Radio Observatory regularly measures the velocity, density, and other ionospheric characteristics near the magnetic equator over Peru.

CHAPTER

FIVE

METHODS

Several methods exist to help combine multiple data sets and convert between equivalent indices.

5.1 DMSP

Supports the Defense Meteorological Satellite Program instruments by providing common custom routines alongside reference and acknowledgement information.

5.2 GNSS

Supports the Global Navigation Satellite System instruments by providing reference and acknowledgement information.

5.3 JRO

Supports the Jicamarca Radio Observatory instrumnets by providing common custom routines alongside reference and acknowledgement information.

5.4 General

Supports the Madrigal data access.

CHAPTER

SIX

EXAMPLES

Here are some examples that demonstrate how to use various pysatMadrigal tools

6.1 Loading DMSP IVM

pysatMadrigal uses pysat to download, load, and provide an analysis framework for data sets archived at the Madrigal database. As specified in the pysat tutorial, data may be loaded using the following commands. Defense Meteorological Satellite Program (DMSP) Ion Velocity Meter (IVM) data is used as an example.

```
import datetime as dt
import pysat
import pysatMadrigal as py_mad

stime = dt.datetime(2012, 5, 14)
ivm = pysat.Instrument(inst_module=py_mad.instruments.dmsp_ivm,
                       tag='utd', inst_id='f15', update_files=True)
ivm.download(start=stime, user="Name+Surname", password="email@org.inst")
ivm.load(date=stime)
print(ivm)
```

The output includes a day of data with UTDallas quality flags from the F15 spacecraft (as implied by the *tag* and *inst_id*), for the specified date. At the time of publication this produces the output shown below.

```
pysat Instrument object
-----------------------
Platform: 'dmsp'
Name: 'ivm'
Tag: 'utd'
Instrument id: 'f15'

Data Processing
---------------
Cleaning Level: 'clean'
Data Padding: None
Keyword Arguments Passed to load: {'xarray_coords': [], 'file_type': 'hdf5'}
Keyword Arguments Passed to list_remote_files: {'user': None, 'password': None, 'url':
↪'http://cedar.openmadrigal.org', 'two_digit_year_break': None}
Custom Functions: 0 applied

Local File Statistics
```

(continues on next page)

(continued from previous page)

```
---------------------
Number of files: 1
Date Range: 31 December 2014 --- 1 January 2015

Loaded Data Statistics
----------------------
Date: 31 December 2014
DOY: 365
Time range: 31 December 2014 00:00:04 --- 31 December 2014 23:18:20
Number of Times: 4811
Number of variables: 30

Variable Names:
year     month    day
            ...
rms_x    sigma_vy sigma_vz

pysat Meta object
-----------------
Tracking 7 metadata values
Metadata for 30 standard variables
Metadata for 0 ND variables
```

CHAPTER

SEVEN

GUIDE FOR DEVELOPERS

7.1 Contributor Covenant Code of Conduct

7.1.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

7.1.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

7.1.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

7.1.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

7.1.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at pysat.developers@gmail.com. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

7.1.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://contributor-covenant.org/version/1/4

7.2 Contributing

Bug reports, feature suggestions and other contributions are greatly appreciated! pysat and pysatMadrigal are community-driven projects that welcome both feedback and contributions.

7.2.1 Short version

- Submit bug reports, feature requests, and questions at GitHub Issues
- Make pull requests to the `develop` branch

7.2.2 More about Issues

Bug reports, questions, and feature requests should all be made as GitHub Issues. Templates are provided for each type of issue, to help you include all the necessary information.

Questions

Not sure how something works? Ask away! The more information you provide, the easier the question will be to answer. You can also interact with the pysat developers on our slack channel.

Bug reports

When 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

Feature requests

If you are proposing a new feature or a change in something that already exists:

- 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 code contributions are welcome :)

7.2.3 More about Development

To set up `pysatMadrigal` for local development:

1. Fork pysatMadrigal on GitHub.
2. Clone your fork locally:

   git clone git@github.com:your_name_here/pysatMadrigal.git

3. Create a branch for local development:

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

4. Make your changes locally. Tests for new instruments are performed automatically. Tests for custom functions should be added to the appropriately named file in `pysatMadrigal/tests`. For example, Jicamarca methods containined in `pysatMadrigal/instruments/methods/jro.py` should be named `pysatMadrigal/tests/test_methods_jro.py`. If no test file exists, then you should create one. This testing uses pytest, which will run tests on any python file in the test directory that starts with `test`. Test classes must begin with `Test`, and test methods must also begin with `test`.
5. When you're done making changes, run all the checks to ensure that nothing is broken on your local system:

   pytest -vs pysatMadrigal

6. Update/add documentation (in `docs`). Even if you don't think it's relevant, check to see if any existing examples have changed.
7. Add your name to the .zenodo.json file as an author

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

   git add . git commit -m "Brief description of your changes" git push origin name-of-your-bugfix-or-feature

9. Submit a pull request through the GitHub website. Pull requests should be made to the `develop` branch.

Pull Request Guidelines

If you need some code review or feedback while you're developing the code, just make a pull request. Pull requests should be made to the `develop` branch.

For merging, you should:

1. Include an example for use
2. Add a note to `CHANGELOG.md` about the changes
3. Ensure that all checks passed (current checks include Travis-CI and Coveralls)[1]

Project Style Guidelines

In general, pysat follows PEP8 and numpydoc guidelines. Pytest runs the unit and integration tests, flake8 checks for style, and sphinx-build performs documentation tests. However, there are certain additional style elements that have been settled on to ensure the project maintains a consistent coding style. These include:

- Line breaks should occur before a binary operator (ignoring flake8 W503)
- Combine long strings using `join`
- Preferably break long lines on open parentheses rather than using \
- Use no more than 80 characters per line
- Avoid using Instrument class key attribute names as unrelated variable names: `platform`, `name`, `tag`, and `inst_id`
- The pysat logger is imported into each sub-module and provides status updates at the info and warning levels (as appropriate)
- Several dependent packages have common nicknames, including:
  - `import datetime as dt`
  - `import numpy as np`
  - `import pandas as pds`
  - `import xarray as xr`
- All classes should have `__repr__` and `__str__` functions
- Docstrings use `Note` instead of `Notes`
- Try to avoid creating a try/except statement where except passes
- Use setup and teardown in test classes
- Use pytest parametrize in test classes when appropriate
- Provide testing class methods with informative failure statements and descriptive, one-line docstrings

[1] If you don't have all the necessary Python versions available locally or have trouble building all the testing environments, you can rely on Travis to run the tests for each change you add in the pull request. Because testing here will delay tests by other developers, please ensure that the code passes all tests on your local system first.

- Block and inline comments should use proper English grammar and punctuation with the exception of single sentences in a block, which may then omit the final period
- **When casting is necessary, use `np.int64` and `np.float64` to ensure operating** system agnosticism

CHAPTER

EIGHT

CHANGE LOG

All notable changes to this project will be documented in this file. This project adheres to Semantic Versioning.

8.1 [0.0.4] - 2021-06-11

- Made changes to structure to comply with updates in pysat 3.0.0
- Deprecations
  - Restructed Instrument methods, moving `madrigal` to `general` and extracting local methods from the instrument modules to platform-specific method files
  - Cycled testing support to cover Python 3.7-3.9
- Enhancements
  - Added coords from pysat.utils
  - Added Vertical TEC Instrument
  - Added documentation
  - Added load routine for simple formatted data
  - Expanded feedback during data downloads
  - Updated documentation configuration to improve maintainability
  - Updated documentation style, displaying logo on sidebar in html format
  - Changed zenodo author name format for better BibTeX compliance
  - Updated CONTRIBUTING and README information
- Bug Fix
  - Updated Madrigal methods to simplify compound data types and enable creation of netCDF4 files using `Instrument.to_netcdf4()`
  - Updated load for multiple files in pandas format
  - Fixed remote listing routine to return filenames instead of experiments
  - Fixed bug introduced by change in xarray requiring engine kwarg
  - Fixed bug that would not list multiple types of files

8.2 [0.0.3] - 2020-06-15

- pypi compatibility

8.3 [0.0.2] - 2020-05-13

- zenodo link

8.4 [0.0.1] - 2020-05-13

- Alpha release

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search