pysat Documentation

Release 0.2.0

Russell Stoneback

April 27, 2015

CONTENTS

CHAPTER ONE

INTRODUCTION

CHAPTER

TWO

API

2.1 Instrument

class `pysat.Instrument`(*platform=None*, *name=None*, *tag=None*, *clean_level='clean'*, *update_files=False*, *pad=None*, *orbit_info=None*, *inst_module=None*, **arg*, ***kwargs*)

Download, load, manage, modify and analyze science data.

Parameters

- **platform** (*string*) – name of platform/satellite.
- **name** (*string*) – name of instrument.
- **tag** (*string, optional*) – identifies particular subset of instrument data.
- **inst_module** (*module, optional*) – Provide instrument module directly (takes precedence over platform/name)
- **clean_level** (*{'clean','dusty','dirty','none'}, optional*) – level of data quality
- **pad** (*pandas.DateOffset, or dictionary, optional*) – length of time to pad the begining and end of loaded data for time-series processing. Extra data is removed after applying all custom functions. Dictionary, if supplied, is simply passed to pandas DateOffset.
- **orbit_info** (*dict*) – Orbit information, {'index':index, 'kind':kind, 'period':period}. See pysat.Orbits for more information.
- **update_files** (*boolean, optional*) – if True, query filesystem for instrument files and store. files.get_new() will return no files after this call until additional files are added.

data

pandas.DataFrame

loaded science data

date

pandas.datetime

date for loaded data

yr

int

year for loaded data

bounds

(datetime/filename/None, datetime/filename/None)

bounds for loading data, supply array_like for a season with gaps

doy
int

day of year for loaded data

files
pysat.Files

interface to instrument files

meta
pysat.Meta

interface to instrument metadata, similar to netCDF 1.6

orbits
pysat.Orbits

interface to extracting data orbit-by-orbit

custom
pysat.Custom

interface to instrument nano-kernel

kwargs
dictionary

keyword arguments passed to instrument loading routine, platform_name.load

Notes

pysat attempts to load the module platform_name.py located in the pysat/instruments directory. This module provides the underlying functionality to download, load, and clean instrument data. Alternatively, the module may be supplied directly using keyword inst_module.

Examples

```
# 1-second mag field data
vefi = pysat.Instrument(platform='cnofs', name='vefi', tag='dc_b',
                        clean_level='clean')
start = pysat.datetime(2009,1,1)
stop = pysat.datetime(2009,1,2)
vefi.download(start, stop)
vefi.load(date=start)
print vefi['dB_mer']
print vefi.meta['db_mer']

# 1-second thermal plasma parameters
ivm = pysat.Instrument(platform='cnofs', name='ivm', tag='', clean_level='clean')
ivm.download(start,stop)
ivm.load(2009,1)
print ivm['ionVelmeridional']

# Ionosphere profiles from GPS occultation
cosmic = pysat.Instrument('cosmic2013', 'gps', 'ionprf', altitude_bin=3)
# bins profile using 3 km step
cosmic.download(start, stop, user=user, password=password)
cosmic.load(date=start)
```

Attributes

bounds	Boundaries for iterating over instrument object by date or file.

Methods

__getitem__(*key*)

Convenience notation for accessing data; obtain inst.data.name using inst['name'].

Examples

By position : inst[row index, 'name']

Slicing by row : inst[row1:row2, 'name']

By Date : inst[datetime, 'name']

Slicing by date [(inclusive)] inst[datetime1:datetime2, 'name']

Slicing by name and row/date : inst[datetime1:datetime1, 'name1':'name2']

__iter__()

Iterates the instrument object by loading subsequent days or files as appropriate.

Limits of iteration, and iteration type (date/file) set by *bounds* attribute.

__setitem__(*key*, *new*)

Convenience method for adding data to instrument.

Examples

Simple Assignment : inst['name'] = newData

Assignment with Metadata : inst['name'] = {'data':new_data, 'long_name':long_name, 'units':units}

Note: If no metadata provided and if metadata for 'name' not already stored then default meta information is also added, long_name = 'name', and units = ''.

bounds

Boundaries for iterating over instrument object by date or file.

Parameters

- **start** (*datetime object, filename, or None (default)*) – start of iteration, if None uses first data date. list-like collection also accepted
- **end** (*datetime object, filename, or None (default)*) – end of iteration, inclusive. If None uses last data date. list-like collection also accepted

Note: both start and stop must be the same type (date, or filename) or None

copy()

Deep copy of the entire Instrument object.

download(*start*, *stop*, *user=None*, *password=None*)

Download data for given Instrument object.

Parameters

- **start** (*pandas.datetime*) – start date to download data
- **stop** (*pandas.datetime*) – stop date to download data
- **user** (*string*) – username, if required by instrument data archive
- **password** (*string*) – password, if required by instrument data archive

load(*yr=None*, *doy=None*, *date=None*, *fname=None*, *fid=None*, *verifyPad=False*)

Loads data for a chosen instrument into .data. Any functions chosen by the user and added to the custom processing queue (.custom.add) are automatically applied to the data before it is available to user in .data.

Keyword Arguments

- **yr** (*integer*) – year for desired data
- **doy** (*integer*) – day of year
- **date** (*datetime object*) – date to load
- **fname** (*'string'*) – filename to be loaded
- **verifyPad** (*boolean*) – if True, padding data not removed (debug purposes)

next()

Manually iterate through the data loaded in satellite object.

Bounds of iteration and iteration type (day/file) are set by *bounds* attribute

Note: If there were no previous calls to load then the first day (default)/file will be loaded.

prev()

Manually iterate backwards through the data loaded in satellite object.

Bounds of iteration and iteration type (day/file) are set by *bounds* attribute

Note: If there were no previous calls to load then the first day (default)/file will be loaded.

to_netcdf3(*fname=None*)

Stores loaded data into a netCDF3 64-bit file.

Stores 1-D data along dimension 'time' - the date time index. Stores object data (dataframes within dataframe) separately:

The name of the object data is used to prepend extra variable dimensions within netCDF, key_2, key_3, first dimension time

The index organizing the data stored as key_sample_index from_netcdf3 uses this naming scheme to reconstruct data structure

The datetime index is stored as 'UNIX time'. netCDF-3 doesn't support 64-bit integers so it is stored as a 64-bit float. This results in a loss of datetime precision when converted back to datetime index up to hundreds of nanoseconds. Use netCDF4 if this is a problem.

All attributes attached to instrument meta are written to netCDF attrs.

2.2 Custom

class pysat.**Custom**

Applies a queue of functions when instrument.load called.

Nano-kernel functionality enables instrument objects that are 'fire and forget'. The functions are always run whenever the instrument load routine is called so instrument objects may be passed safely to other routines and the data will always be processed appropriately.

Examples

```
def custom_func(inst, opt_param1=False, opt_param2=False):
    return None
instrument.custom.add(custom_func, 'modify', opt_param1=True)

def custom_func2(inst, opt_param1=False, opt_param2=False):
    return data_to_be_added
instrument.custom.add(custom_func2, 'add', opt_param2=True)
instrument.load(date=date)
print instrument['data_to_be_added']
```

See also:

Custom.add

Notes

User should interact with Custom through pysat.Instrument instance's attribute, instrument.custom

Methods

---

add (*function*, *kind='add'*, *at_pos='end'*, **args*, ***kwargs*)

Add a function to custom processing queue.

Custom functions are applied automatically to associated pysat instrument whenever instrument.load command called.

Parameters

- **function** (*string or function object*) – name of function or function object to be added to queue
- **kind** (*{'add', 'modify', 'pass}*) –

  add : Adds data returned from function to instrument object. A copy of pysat instrument object supplied to routine.

  modify : pysat instrument object supplied to routine. Any and all changes to object are retained.

  pass : A copy of pysat object is passed to function. No data is accepted from return.

- **at_pos** (*string or int*) – insert at position. (default, insert at end).

Notes

Allowed *add* function returns :

- {'data' : pandas Series/DataFrame/array_like, 'units' : string/array_like of strings, 'long_name' : string/array_like of strings, 'name' : string/array_like of strings (iff data array_like)}
- pandas DataFrame, names of columns are used
- pandas Series, .name required
- (string/list of strings, numpy array/list of arrays)

clear()

Clear custom function list.

2.3 Files

class pysat.**Files**(*sat*)

Maintains collection of files for instrument object.

Uses the list_files functions for each specific instrument to create an ordered collection of files in time. Used by instrument object to load the correct files. Files also contains helper methods for determining the presence of new files and creating an ordered list of files.

base_path

string

path to .pysat directory in user home

start_date

datetime

date of first file, used as default start bound for instrument object

stop_date

datetime

date of last file, used as default stop bound for instrument object

data_path

string

path to the directory containing instrument files, top_dir/platform/name/tag/

Notes

User should generally use the interface provided by a pysat.Instrument instance. Exceptions are the classmethod from_os, provided to assist in generating the appropriate output for an instrument routine.

Examples

```
# convenient file access
inst = pysat.Instrument(platform=platform, name=name, tag=tag)
# first file
inst.files[0]

# files from start up to stop (exclusive on stop)
start = pysat.datetime(2009,1,1)
stop = pysat.datetime(2009,1,3)
print vefi.files[start:stop]

# files for date
print vefi.files[start]

# files by slicing
print vefi.files[0:4]

# get a list of new files
# new files are those that weren't present the last time
# a given instrument's file list was stored
new_files = vefi.files.get_new()

# search pysat appropriate directory for instrument files and
# update Files instance, knowledge not written to disk.
vefi.files.refresh()

# search pysat appropriate directory for files and store new list
vefi.files.refresh(store=True)
# running get_new will now return an empty list until
# additional files are introduced
```

Methods

classmethod **`from_os`**(*data_path=None*, *format_str=None*, *two_digit_year_break=None*)

Produces a list of files and and formats it for Files class.

Parameters

- **`data_path`** (*string*) – Top level directory to search files for. This directory is provided by pysat to the instrument_module.list_files functions as data_path.
- **`format_string`** (*string with python format codes*) – Provides the naming pattern of the instrument files and the locations of date information so an ordered list may be produced.
- **`two_digit_year_break`** (*int*) – If filenames only store two digits for the year, then '1900' will be added for years >= two_digit_year_break, and '2000' will be added for years < two_digit_year_break.

Notes

Does not produce a Files instance, but the proper output from instrument_module.list_files method.

`get_file_array`(*start*, *end*)

Return a list of filenames between and including start and end.

Parameters

- **`start`** (*array_like or single string*) – filenames for start of returned filelist
- **`stop`** (*array_like or single string*) – filenames inclusive end of list

Returns

- *list of filenames between and including start and end over all*
- *intervals.*

`get_index`(*fname*)

Return index for a given filename.

Parameters **`fname`** (*string*) – filename

Notes

If fname not found in the file information already attached to the instrument.files instance, then a files.refresh() call is made.

`get_new`()

List all new files since last time list was stored.

pysat stores filenames in the user_home/.pysat directory. Returns a list of all new fileanmes since the last store. Filenames are stored if update_files is True at instrument object level and if files.refresh(store=True) is called.

Returns

- *pandas Series of filenames*
- *False if no filenames*

`refresh`(*store=False*)

Refresh loaded instrument filelist by searching filesystem.

Searches pysat provided path, pysat_data_dir/platform/name/tag/, where pysat_data_dir is set by pysat.utils.set_data_dir(path=path).

Parameters **`store`** (*boolean*) – set True to store loaded file names into .pysat directory

2.4 Meta

class `pysat.`**`Meta`**(*metadata=None*)

Stores metadata for Instrument instance, similar to CF-1.6 netCDFdata standard.

Parameters **`metadata`** (*pandas.DataFrame*) – DataFrame should be indexed by variable name that contains at minimum the standard_name (name), units, and long_name for the data stored in the associated pysat Instrument object.

`data`

pandas.DataFrame

index is variable standard name, ‘units’ and ‘long_name’ are also stored along with additional user provided labels.

Methods

__getitem__(*key*)

Convenience method for obtaining metadata.

Maps to pandas DataFrame.ix method.

Examples

```
print meta['name']
```

__setitem__(*name*, *value*)

Convenience method for adding metadata.

Examples

```
meta = pysat.Meta()
meta['name'] = {'long_name':string, 'units':string}
# update 'units' to new value
meta['name'] = {'units':string}
# update 'long_name' to new value
meta['name'] = {'long_name':string}
# attach new info with partial information, 'long_name' set to 'name2'
meta['name2'] = {'units':string}
# units are set to '' by default
meta['name3'] = {'long_name':string}
```

classmethod from_csv(*name=None*, *col_names=None*, *sep=None*, ***kwargs*)

Create instrument metadata object from csv.

Parameters

- **name** (*string*) – absolute filename for csv file or name of file stored in pandas instruments location
- **col_names** (*list-like collection of strings*) – column names in csv and resultant meta object
- **sep** (*string*) – column seperator for supplied csv filename

Note: column names must include at least ['name', 'long_name', 'units'], assumed if col_names is None.

classmethod from_dict()

not implemented yet, load metadata from dict of items/list types

classmethod from_nc()

not implemented yet, load metadata from netCDF

replace(*metadata=None*)

Replace stored metadata with input data.

Parameters metadata (*pandas.DataFrame*) – DataFrame should be indexed by variable name that contains at minimum the standard_name (name), units, and long_name for the data stored in the associated pysat Instrument object.

2.5 Orbits

class pysat.**Orbits**(*sat=None*, *index=None*, *kind=None*, *period=None*)

Determines orbits on the fly and provides orbital data in .data.

Determines the locations of orbit breaks in the loaded data in inst.data and provides iteration tools and convenient orbit selection via inst.orbit[orbit num].

Parameters

- **sat** (*pysat.Instrument instance*) – instrument object to determine orbits for
- **index** (*string*) – name of the data series to use for determing orbit breaks
- **kind** (*{'local time', 'longitude', 'polar'}*) – kind of orbit, determines how orbital breaks are determined
  - local time: negative gradients in lt or breaks in inst.data.index
  - longitude: negative gradients or breaks in inst.data.index
  - polar: zero crossings in latitude or breaks in inst.data.index
- **period** (*np.timedelta64*) – length of time for orbital period, used to gauge when a break in the datetime index (inst.data.index) is large enough to consider it a new orbit

Notes

class should not be called directly by the user, use the interface provided by inst.orbits where inst = pysat.Instrument()

Examples

```
info = {'index':'longitude', 'kind':'longitude'}
vefi = pysat.Instrument(platform='cnofs', name='vefi', tag='dc_b',
                        clean_level=None, orbit_info=info)
start = pysat.datetime(2009,1,1)
stop = pysat.datetime(2009,1,10)
vefi.load(date=start)
vefi.bounds(start, stop)

# iterate over orbits
for vefi in vefi.orbits:
    print 'Next available orbit ', vefi['dB_mer']

# load fifth orbit of first day
vefi.load(date=start)
vefi.orbits[5]

# less convenient load
vefi.orbits.load(5)

# manually iterate orbit
vefi.orbits.next()
# backwards
vefi.orbits.prev()
```

Methods

`__getitem__`(*key*)

Enable convenience notation for loading orbit into parent object.

Example

```
inst.load(date=date)
inst.orbits[4]
print 'Orbit data ', inst.data
```

Note: A day of data must already be loaded.

`__iter__`()

Support iteration by orbit.

For each iteration the next available orbit is loaded into inst.data.

Example

```
for inst in inst.orbits:
    print 'next available orbit ', inst.data
```

`load`(*orbit=None*)

Load a particular orbit into .data for loaded day.

Parameters = orbit number, 1 indexed (*orbit*) –

Note: A day of data must be loaded before this routine functions properly. If the last orbit of the day is requested, it will automatically be padded with data from the next day. The orbit counter will be reset to 1.

`next`(**arg*, ***kwarg*)

Load the next orbit into .data.

Note: Forms complete orbits across day boundaries. If no data loaded then the first orbit from the first date of data is returned.

`prev`(**arg*, ***kwarg*)

Load the next orbit into .data.

Note: Forms complete orbits across day boundaries. If no data loaded then the last orbit of data from the last day is loaded into .data.

2.6 Utilities

`pysat.utils.`**`create_datetime_index`**(*year=None*, *month=None*, *day=None*, *uts=None*)

Create a timeseries index using supplied year, month, day, and ut in seconds.

Parameters

- **year** (*array_like of ints*) –
- **month** (*array_like of ints or None*) –
- **day** (*array_like of ints*) – for day (default) or day of year (use month=None)
- **uts** (*array_like of floats*) –

Returns

Return type Pandas timeseries index.

Note: Leap seconds have no meaning here.

`pysat.utils.`**`getyrdoy`**(*date*)

Return a tuple of year, day of year for a supplied datetime object.

`pysat.utils.`**`load_netcdf3`**(*fnames=None*, *strict_meta=False*, *index_label=None*, *unix_time=False*, ***kwargs*)

Load netCDF-3 file produced by pysat.

Parameters

- **fnames** (*string or array_like of strings*) – filenames to load
- **strict_meta** (*boolean*) – check if metadata across filenames is the same
- **index_label** (*string*) – name of data to be used as DataFrame index
- **unix_time** (*boolean*) – True if index_label refers to UNIX time

`pysat.utils.`**`season_date_range`**(*start*, *stop*, *freq='D'*)

Return array of datetime objects using input frequency from start to stop

Supports single datetime object or list, tuple, ndarray of start and stop dates.

freq codes correspond to pandas date_range codes, D daily, M monthly, S secondly

`pysat.utils.`**`set_data_dir`**(*path=None*)

set the top level directory pysat uses to look for data.

2.7 Supported Instruments

2.7.1 C/NOFS VEFI

Supports the Vector Electric Field Instrument (VEFI) onboard the Communication and Navigation Outage Forecasting System (C/NOFS) satellite. Downloads data from the NASA Coordinated Data Analysis Web (CDAWeb).

param tag

type tag {'dc_b'}

Notes

- tag = 'dc_b': 1 second DC magnetometer data

Warning:

- Currently no cleaning routine.
- Module not written by VEFI team.

2.7.2 C/NOFS IVM

Supports the Ion Velocity Meter (IVM) onboard the Communication and Navigation Outage Forecasting System (C/NOFS) satellite, part of the Coupled Ion Netural Dynamics Investigation (CINDI). Downloads data from the NASA Coordinated Data Analysis Web (CDAWeb) in CDF format.

param tag No tags supported

type tag string

Warning:

- The sampling rate of the instrument changes on July 29th, 2010. The rate is attached to the instrument object as .sample_rate.
- The cleaning parameters for the instrument are still under development.

2.7.3 COSMIC 2013 GPS

Loads data from the COSMIC satellite, 2013 reprocessing.

The Constellation Observing System for Meteorology, Ionosphere, and Climate (COSMIC) are six satellites in LEO with GPS receivers. The occultation of GPS signals by the atmosphere provides a measurement of atmospheric parameters. Data downloaded from the COSMIC Data Analaysis and Archival Center.

Notes

- ‘ionprf: ‘ionPrf’ ionosphere profiles
- ‘sonprf’: ‘sonPrf’ files
- ‘wetprf’: ‘wetPrf’ files
- ‘atmPrf’: ‘atmPrf’ files

Warning:

- Routine was not produced by COSMIC team

2.7.4 COSMIC GPS

Loads data from the COSMIC satellite.

tags: ‘ionprf: ‘ionPrf’ ionosphere profiles ‘sonprf’: ‘sonPrf’ files ‘wetprf’: ‘wetPrf’ files ‘atmPrf’: ‘atmPrf’

PYTHON MODULE INDEX