satpy documentation

Release 0.8.1

SMHI

Feb 19, 2018

Contents

1 Overview **3**
- 1.1 Scene . . . 3
- 1.2 Datasets . . . 3
- 1.3 Reading . . . 4
- 1.4 Compositing . . . 4
- 1.5 Resampling . . . 4
- 1.6 Enhancements . . . 4
- 1.7 Writing . . . 4

2 Installation Instructions **7**
- 2.1 Pip-based Installation . . . 7
- 2.2 Conda-based Installation . . . 7
- 2.3 Ubuntu System Python Installation . . . 8

3 Quickstart **9**
- 3.1 Loading data . . . 9
- 3.2 Generating composites . . . 10
- 3.3 Resampling . . . 11
- 3.4 Saving to disk . . . 11
- 3.5 Troubleshooting . . . 12

4 Readers **13**
- 4.1 Filter loaded files . . . 13
- 4.2 Load data . . . 13
- 4.3 Search for local files . . . 14
- 4.4 Adding a Reader to SatPy . . . 14

5 Composites **15**
- 5.1 Modifiers . . . 15
- 5.2 Making custom composites . . . 15

6 Resampling **17**
- 6.1 Resampling algorithms . . . 17
- 6.2 Create custom area definition . . . 17
- 6.3 Create dynamic area definition . . . 17
- 6.4 Store area definitions . . . 18

7 Writers **19**
7.1 Colorizing and Palettizing using user-supplied colormaps . . . 19

8 Developer's Guide **21**
8.1 How to contribute . . . 21
8.2 Migrating to xarray and dask . . . 23
8.3 Coding guidelines . . . 23
8.4 Development installation . . . 23
8.5 Running tests . . . 23
8.6 Documentation . . . 24

9 satpy package **25**
9.1 Subpackages . . . 25
9.2 Submodules . . . 61
9.3 satpy.config module . . . 61
9.4 satpy.dataset module . . . 61
9.5 satpy.multiscene module . . . 63
9.6 satpy.node module . . . 63
9.7 satpy.plugin_base module . . . 65
9.8 satpy.resample module . . . 65
9.9 satpy.scene module . . . 67
9.10 satpy.utils module . . . 69
9.11 satpy.version module . . . 70
9.12 Module contents . . . 70

10 Indices and tables **73**

Python Module Index **75**

SatPy is a python library for reading and manipulating meteorological remote sensing data and writing it to various image and data file formats. SatPy comes with the ability to make various RGB composites directly from satellite instrument channel data or higher level processing output. The pyresample package is used to resample data to different uniform areas or grids.

Get to the project page for source code and downloads.

It is designed to be easily extendable to support any meteorological satellite by the creation of plugins (readers, compositors, writers, etc). The table at the bottom of this page shows the input formats supported by the base SatPy installation.

Note: SatPy's interfaces are not guaranteed stable and may change until version 1.0 when backwards compatibility will be a main focus.

CHAPTER 1

Overview

SatPy is designed to provide easy access to common operations for processing meteorological remote sensing data. Any details needed to perform these operations are configured internally to SatPy meaning users should not have to worry about *how* something is done, only ask for what they want. Most of the features provided by SatPy can be configured by keyword arguments (see the *API Documentation* or other specific section for more details). For more complex customizations or added features SatPy uses a set of configuration files that can be modified by the user. The various components and concepts of SatPy are described below. The *Quickstart* guide also provides simple example code for the available features of SatPy.

1.1 Scene

SatPy provides most of its functionality through the `Scene` class. The acts as a container for the datasets being operated on and provides methods for acting on those datasets. It attempts to reduce the amount of low-level knowledge needed by the user while still providing a pythonic interface to the functionality underneath.

A Scene object represents a single geographic region of data, typically at a single continuous time range. It is possible to combine Scenes to form a Scene with multiple regions or multiple time observations, but it is not guaranteed that all functionality works in these situations.

1.2 Datasets

SatPy's lowest-level container for data is the `Dataset`. `Dataset` is a subclass of NumPy's `MaskedArray` with an additional `.info` dictionary attribute for various metadata. In most use cases these objects can be operated on like normal NumPy arrays with special care taken to make sure the metadata dictionary contains expected values.

Warning: Starting with version 0.9, SatPy will replace `Dataset` with the `xarray.DataArray` object. The main difference will be that the `.info` metadata dictionary will be accessed via `.attrs`.

`Datasets` are identified throughout SatPy by a `DatasetID`. A `DatasetID` consists of various pieces of available metadata. This usually includes *name* and *wavelength* as identifying metadata, but also includes *resolution*, *calibration*, *polarization*, and additional *modifiers* to further distinguish one dataset from another.

1.3 Reading

One of the biggest advantages of using SatPy is the large number of input file formats that it can read. It encapsulates this functionality in to individual *Readers*. SatPy Readers handle all of the complexity of reading whatever format they represent. Meteorological Satellite file formats can be extremely complex and formats are rarely reused across satellites or instruments. No matter the format, SatPy's Reader interface is meant to provide a consistent data loading interface while still providing flexibility to add new complex file formats.

1.4 Compositing

Many users of satellite imagery combine multiple sensor channels to bring out certain features of the data. This includes using one dataset to enhance another, combining 3 or more datasets in to an RGB image, or any other combination of datasets. SatPy comes with a lot of common composite combinations built-in and allows the user to request them like any other dataset. SatPy also makes it possible to create your own custom composites and have SatPy treat them like any other dataset. See *Composites* for more information.

1.5 Resampling

Satellite imagery data comes in two forms when it comes to geolocation, native satellite swath coordinates and uniform gridded projection coordinates. It is also common to see the channels from a single sensor in multiple resolutions, making it complicated to combine or compare the datasets. Many use cases of satellite data require the data to be in a certain projection other than the native projection or to have output imagery cover a specific area of interest. SatPy makes it easy to resample datasets to allow for users to combine them or grid them to these projections or areas of interest. SatPy uses the PyTroll *pyresample* package to provide nearest neighbor, bilinear, or elliptical weighted averaging resampling methods. See *Resampling* for more information.

1.6 Enhancements

When making images from satellite data the data has to be manipulated to be compatible with the output image format and still look good to the human eye. SatPy calls this functionality "enhancing" the data, also commonly called scaling or stretching the data. This process can become complicated not just because of how subjective the quality of an image can be, but also because of historical expectations of forecasters and other users for how the data should look. SatPy tries to hide the complexity of all the possible enhancement methods from the user and just provide the best looking image by default. SatPy still makes it possible to customize these procedures, but in most cases it shouldn't be necessary. See the documentation on *Writers* for more information on what's possible for output formats and enhancing images.

1.7 Writing

SatPy is designed to make data loading, manipulating, and analysis easy. However, the best way to get satellite imagery data out to as many users as possible is to make it easy to save it in multiple formats. SatPy allows users to save data in image formats like PNG or GeoTIFF as well as data file formats like NetCDF. Each format's complexity is

hidden behind the interface of individual Writer objects and includes keyword arguments for accessing specific format features like compression and output data type. See the *Writers* documentation for the available writers and how to use them.

CHAPTER 2

Installation Instructions

2.1 Pip-based Installation

SatPy is available from the Python Packaging Index (PyPI). A sandbox environment for *satpy* can be created using Virtualenv.

To install the *satpy* package and the minimum amount of python dependencies:

```
$ pip install satpy
```

Additional dependencies can be installed as "extras" and are grouped by reader, writer, or feature added. Extras available can be found in the setup.py file. They can be installed individually:

```
$ pip install satpy[viirs_sdr]
```

Or all at once, although this isn't recommended due to the large number of dependencies:

```
$ pip install satpy[all]
```

2.2 Conda-based Installation

Currently SatPy is not available on any common conda environment. However, it is possible to install SatPy in a conda environment with a combination of the *conda-forge* channel and pip. A typical conda environment for SatPy can be created with the following commands:

```
$ conda config --add channels conda-forge
$ conda env create -n satpy-env python=3.6 xarray dask pyresample netcdf4 h5py gdal
$ source activate satpy-env
$ pip install satpy
```

Using the *pip* commands as described above you should now have a complete conda environment with a majority of SatPy's dependencies installed. Activate the environment with `source activate satpy-env` to use the environment in the future.

2.3 Ubuntu System Python Installation

To install SatPy on an Ubuntu system we recommend using virtual environments to separate SatPy and its dependencies from the rest of the system. Note that these instructions require using "sudo" privileges which may not be available to all users and can be very dangerous. The following instructions attempt to install some SatPy dependencies using the Ubuntu *apt* package manager to ease installation. Replace */path/to/pytroll-env* with the environment to be created.

```
$ sudo apt-get install python-pip python-gdal
$ sudo pip install virtualenv
$ virtualenv /path/to/pytroll-env
$ source /path/to/pytroll-env/bin/activate
$ pip install satpy
```

CHAPTER 3

Quickstart

3.1 Loading data

To work with weather satellite data you must create a *Scene* object. In order for SatPy to get access to the data it must be told what files to read and what *SatPy Reader* should read them:

```
>>> from satpy import Scene
>>> from glob import glob
>>> filenames = glob("/home/a001673/data/satellite/Meteosat-10/seviri/lvl1.5/2015/04/
↪20/HRIT/*201504201000*")
>>> global_scene = Scene(reader="hrit_msg", filenames=filenames)
```

To load data from the files use the *Scene.load* method:

```
>>> global_scene.load([0.6, 0.8, 10.8])
>>> print(global_scene)

seviri/IR_108:
        area: On-the-fly area
        start_time: 2015-04-20 10:00:00
        units: K
        wavelength_range: (9.8, 10.8, 11.8) μm
        shape: (3712, 3712)
seviri/VIS006:
        area: On-the-fly area
        start_time: 2015-04-20 10:00:00
        units: %
        wavelength_range: (0.56, 0.635, 0.71) μm
        shape: (3712, 3712)
seviri/VIS008:
        area: On-the-fly area
        start_time: 2015-04-20 10:00:00
        units: %
        wavelength_range: (0.74, 0.81, 0.88) μm
        shape: (3712, 3712)
```

SatPy allows loading file data by wavelengths in micrometers (shown above) or by channel name:

```
>>> global_scene.load(["VIS006", "VIS008", "IR_108"])
```

To have a look at the available channels for loading from your *Scene* object use the *available_datasets* method:

```
>>> global_scene.available_dataset_names()

['HRV',
 'IR_108',
 'IR_120',
 'VIS006',
 'WV_062',
 'IR_039',
 'IR_134',
 'IR_097',
 'IR_087',
 'VIS008',
 'IR_016',
 'WV_073']
```

To access the loaded data use the wavelength or name:

```
>>> print(global_scene[0.6])
```

To visualize loaded data in a pop-up window:

```
>>> global_scene.show(0.6)
```

To make combine datasets and make a new dataset:

```
>>> global_scene["ndvi"] = (global_scene[0.8] - global_scene[0.6]) / (global_scene[0.
↪8] + global_scene[0.6])
>>> global_scene.show("ndvi")
```

For more information on loading datasets by resolution, calibration, or other advanced loading methods see the *Readers* documentation.

3.2 Generating composites

SatPy comes with many composite recipes built-in and makes them loadable like any other dataset:

```
>>> global_scene.load(['overview'])
```

To get a list of all available composites for the current scene:

```
>>> global_scene.available_composite_names()

['overview_sun',
 'airmass',
 'natural',
 'night_fog',
 'overview',
```

```
 'green_snow',
 'dust',
 'fog',
 'natural_sun',
 'cloudtop',
 'convection',
 'ash']
```

Loading composites will load all necessary dependencies to make that composite and unload them after the composite has been generated.

Note: Some composite require datasets to be at the same resolution or shape. When this is the case the Scene object must be resampled before the composite can be generated (see below).

3.3 Resampling

In certain cases it may be necessary to resample datasets whether they come from a file or are generated composites. Resampling is useful for mapping data to a uniform grid, limiting input data to an area of interest, changing from one projection to another, or for preparing datasets to be combined in a composite (see above). For more details on resampling, different resampling algorithms, and creating your own area of interest see the *Resampling* documentation. To resample a SatPy Scene:

```
>>> local_scene = global_scene.resample("eurol")
```

This creates a copy of the original `global_scene` with all loaded datasets resampled to the built-in "eurol" area. Any composites that were requested, but could not be generated are automatically generated after resampling. The new `local_scene` can now be used like the original `global_scene` for working with datasets, saving them to disk or showing them on screen:

```
>>> local_scene.show('overview')
>>> local_scene.save_dataset('overview', './local_overview.tif')
```

3.4 Saving to disk

To save all loaded datasets to disk as geotiff images:

```
>>> global_scene.save_datasets()
```

To save all loaded datasets to disk as PNG images:

```
>>> global_scene.save_datasets(writer='simple_image')
```

Or to save an individual dataset:

```
>>> global_scene.save_dataset('VIS006', 'my_nice_image.png')
```

Datasets are automatically scaled or "enhanced" to be compatible with the output format and to provide the best looking image. For more information on saving datasets and customizing enhancements see the documentation on *Writers*.

3.5 Troubleshooting

Due to the way SatPy works, producing as many datasets as possible, there are times that behavior can be unexpected but with no exceptions raised. To help troubleshoot these situations log messages can be turned on. To do this run the following code before running any other SatPy code:

```
>>> from satpy.utils import debug_on
>>> debug_on()
```

CHAPTER 4

Readers

SatPy supports reading and loading data from many input file formats and schemes. The *Scene* object provides a simple interface around all the complexity of these various formats through its `load` method. The following sections describe the different way data can be loaded, requested, or added to a Scene object.

4.1 Filter loaded files

Coming soon...

4.2 Load data

Datasets in SatPy are identified by certain pieces of metadata set during data loading. These include *name*, *wavelength*, *calibration*, *resolution*, *polarization*, and *modifiers*. Normally, once a `Scene` is created requesting datasets by *name* or *wavelength* is all that is needed:

```
>>> from satpy import Scene
>>> scn = Scene(reader="hrit_msg", filenames=filenames)
>>> scn.load([0.6, 0.8, 10.8])
>>> scn.load(['IR_120', 'IR_134'])
```

However, in many cases datasets are available in multiple spatial resolutions, multiple calibrations (`brightness_temperature`, `reflectance`, `radiance`, etc), multiple polarizations, or have corrections or other modifiers already applied to them. By default SatPy will provide the version of the dataset with the highest resolution and the highest level of calibration (brightness temperature or reflectance over radiance). It is also possible to request one of these exact versions of a dataset by using the *DatasetID* class:

```
>>> from satpy import DatasetID
>>> my_channel_id = DatasetID(name='IR_016', calibration='radiance')
>>> scn.load([my_channel_id])
>>> print(scn['IR_016'])
```

Or request multiple datasets at a specific calibration, resolution, or polarization:

```
>>> scn.load([0.6, 0.8], resolution=1000)
```

Or multiple calibrations:

```
>>> scn.load([0.6, 10.8], calibrations=['brightness_temperature', 'radiance'])
```

In the above case SatPy will load whatever dataset is available and matches the specified parameters. So the above `load` call would load the `0.6` (a visible/reflectance band) radiance data and `10.8` (an IR band) brightness temperature data.

Note: If a dataset could not be loaded there is no exception raised. You must check the *scn.missing_datasets* property for any `DatasetID` that could not be loaded.

To find out what datasets are available from a reader from the files that were provided to the `Scene` use *available_dataset_ids()*:

```
>>> scn.available_dataset_ids()
```

Or *available_dataset_names()* for just the string names of Datasets:

```
>>> scn.available_dataset_names()
```

4.3 Search for local files

SatPy provides a utility *find_files_and_readers()* for searching for files in a base directory matching various search parameters. This function discovers files based on filename patterns. It returns a dictionary mapping reader name to a list of filenames supported. This dictionary can be passed directly to the *Scene* initialization.

```
>>> from satpy import find_files_and_readers, Scene
>>> from datetime import datetime
>>> my_files = find_files_and_readers(base_dir='/data/viirs_sdrs',
...                                   reader='viirs_sdr',
...                                   start_time=datetime(2017, 5, 1, 18, 1, 0),
...                                   end_time=datetime(2017, 5, 1, 18, 30, 0))
>>> scn = Scene(filenames=my_files)
```

See the *find_files_and_readers()* documentation for more information on the possible parameters.

4.4 Adding a Reader to SatPy

Coming soon...

CHAPTER 5

Composites

Documentation coming soon...

5.1 Modifiers

5.2 Making custom composites

Note: These features will be added to the `Scene` object in the future.

Building custom composites makes use of the `GenericCompositor` class. For example, building an overview composite can be done manually with:

```
>>> from satpy.composites import GenericCompositor
>>> compositor = GenericCompositor("myoverview", "bla", "")
>>> composite = compositor([local_scene[0.6],
...                         local_scene[0.8],
...                         local_scene[10.8]])
>>> from satpy.writers import to_image
>>> img = to_image(composite)
>>> img.invert([False, False, True])
>>> img.stretch("linear")
>>> img.gamma(1.7)
>>> img.show()
```

One important thing to notice is that there is an internal difference between a composite and an image. A composite is defined as a special dataset which may have several bands (like R, G, B bands). However, the data isn't stretched, or clipped or gamma filtered until an image is generated.

To save the custom composite, the following procedure can be used:

1. Create a custom directory for your custom configs.

2. Set it in the environment variable called PPP_CONFIG_DIR.
3. Write config files with your changes only (look at eg satpy/etc/composites/seviri.yaml for inspiration), pointing to the custom module containing your composites. Don't forget to add changes to the enhancement/generic.cfg file too.
4. Put your composites module on the python path.

With that, you should be able to load your new composite directly.

CHAPTER 6

Resampling

6.1 Resampling algorithms

The default resampling method in SatPy is nearest neighbor (`nearest`). There are also `bilinear` and Elliptical Weighted Averaging (`ewa`) available.

```
>>> local_scene = global_scene.resample("euro4", resampler="bilinear")
```

To make resampling faster next time (when resampling geostationary satellite data), it is possible to save the resampling coefficients and use more CPUs when calculating the coefficients on the first go:

```
>>> local_scene = global_scene.resample("euro4", resampler="bilinear",
...                                     nprocs=4, cache_dir="/var/tmp")
```

6.2 Create custom area definition

See `pyresample.geometry.AreaDefinition` for information on creating areas that can be passed to the resample method:

```
>>> from pyresample.geometry import AreaDefinition
>>> my_area = AreaDefinition(...)
>>> local_scene = global_scene.resample(my_area)
```

6.3 Create dynamic area definition

See `pyresample.geometry.DynamicAreaDefinition` for more information.

Examples coming soon...

6.4 Store area definitions

Area definitions can be added to a custom YAML file (see pyresample's documentation for more information) and loaded using pyresample's utility methods:

```
>>> from pyresample.utils import parse_area_file
>>> my_area = parse_area_file('my_areas.yaml', 'my_area')[0]
```

Examples coming soon...

CHAPTER 7

Writers

SatPy makes it possible to save datasets in multiple formats. For details on additional arguments and features available for a specific Writer see the table below. Most use cases will want to save datasets using the *save_datasets()* method:

```
>>> scn.save_datasets(writer='simple_image')
```

The `writer` parameter defaults to using the `geotiff` writer. One common parameter across almost all Writers is `file_pattern` and `base_dir` to help automate saving files with custom filenames:

```
>>> scn.save_datasets(
...     file_pattern='{name}_{start_time:%Y%m%d_%H%M%S}.tif',
...     base_dir='/tmp/my_ouput_dir')
```

Table 7.1: SatPy Writers

Description	Writer name	Status
GeoTIFF	*geotiff*	Nominal
Simple Image (PNG, JPEG, etc)	*simple_image*	Nominal
NinJo TIFF (using `pyninjotiff` package)	*ninjotiff*	Nominal
NetCDF (Standard CF)	*cf*	Pre-alpha
AWIPS II Tiled SCMI NetCDF4	`scmi`	Beta

7.1 Colorizing and Palettizing using user-supplied colormaps

Note: In the future this functionality will be added to the `Scene` object.

It is possible to create single channel "composites" that are then colorized using users' own colormaps. The colormaps are Numpy arrays with shape (num, 3), see the example below how to create the mapping file(s).

This example creates a 2-color colormap, and we interpolate the colors between the defined temperature ranges. Beyond those limits the image clipped to the specified colors.

```
>>> import numpy as np
>>> from satpy.composites import BWCompositor
>>> from satpy.enhancements import colorize
>>> from satpy.writers import to_image
>>> arr = np.array([[0, 0, 0], [255, 255, 255]])
>>> np.save("/tmp/binary_colormap.npy", arr)
>>> compositor = BWCompositor("test", standard_name="colorized_ir_clouds")
>>> composite = compositor((local_scene[10.8], ))
>>> img = to_image(composite)
>>> kwargs = {"palettes": [{"filename": "/tmp/binary_colormap.npy",
...           "min_value": 223.15, "max_value": 303.15}]}
>>> colorize(img, **kwargs)
>>> img.show()
```

Similarly it is possible to use discreet values without color interpolation using *palettize()* instead of *colorize()*

You can define several colormaps and ranges in the *palettes* list and they are merged together. See trollimage documentation for more information how colormaps and color ranges are merged.

The above example can be used in enhancements YAML config like this:

```
hot_or_cold:
  standard_name: hot_or_cold
  operations:
    - name: colorize
      method: &colorizefun !!python/name:satpy.enhancements.colorize ''
      kwargs:
        palettes:
          - {filename: /tmp/binary_colormap.npy, min_value: 223.15, max_value: 303.15}
```

CHAPTER 8

Developer's Guide

The below sections will walk through how to set up a development environment, make changes to the code, and test that they work. See the *How to contribute* section for more information on getting started and contributor expectations. Additional information for developer's can be found at the pages listed below.

8.1 How to contribute

Thank you for considering contributing to SatPy! SatPy's development team is made up of volunteers so any help we can get is very appreciated.

Contributions from users are what keep this community going. We welcome any contributions including bug reports, documentation fixes or updates, bug fixes, and feature requests. By contributing to SatPy you are providing code that everyone can use and benefit from.

The following guidelines will describe how the SatPy project structures its code contributions from discussion to code to package release.

For more information on contributing to open source projects see GitHub's Guide.

8.1.1 What can I do?

- Make sure you have a GitHub account.
- Submit a ticket for your issue, assuming one does not already exist.
- If you're uncomfortable using Git/GitHub, see Learn Git Branching or other online tutorials.
- If you are uncomfortable contributing to an open source project see:
  - How to Contribute to an Open Source Project on GitHub video series
  - Aaron Meurer's Git Workflow
  - How to Contribute to Open Source
- See what issues already exist. Issues marked good first issue or help wanted can be good issues to start with.

- Read the *Developer's Guide* for more details on contributing code.
- Fork the repository on GitHub and install the package in development mode.
- Update the SatPy documentation to make it clearer and more detailed.
- Contribute code to either fix a bug or add functionality and submit a Pull Request.
- Make an example Jupyter Notebook and add it to the available examples.

8.1.2 What if I break something?

Not possible. If something breaks because of your contribution it was our fault. When you submit your changes to be merged as a GitHub Pull Request they will be automatically tested and checked against coding style rules. Before they are merged they are reviewed by at least one maintainer of the SatPy project. If anything needs updating, we'll let you know.

8.1.3 What is expected?

You can expect the SatPy maintainers to help you. We are all volunteers, have jobs, and occasionally go on vacations. We will try our best to answer your questions as soon as possible. We will try our best to understand your use case and add the features you need. Although we strive to make SatPy useful for everyone there may be some feature requests that we can't allow if they would require breaking existing features. Other features may be best for a different package, PyTroll or otherwise. Regardless, we will help you find the best place for your feature and to make it possible to do what you want.

We, the SatPy maintainers, expect you to be patient, understanding, and respectful of both developers and users. SatPy can only be successful if everyone in the community feels welcome. We also expect you to put in as much work as you expect out of us. There is no dedicated PyTroll or SatPy support team, so there may be times when you need to do most of the work to solve your problem (trying different test cases, environments, etc).

Being respectful includes following the style of the existing code for any code submissions. Please follow PEP8 style guidelines and limit lines of code to 80 characters whenever possible and when it doesn't hurt readability. SatPy follows Google Style Docstrings for all code API documentation. When in doubt use the existing code as a guide for how coding should be done.

8.1.4 How do I get help?

The SatPy developers (and all other PyTroll package developers) monitor the:

- Mailing List
- Slack chat (get an invitation)
- GitHub issues

8.1.5 How do I submit my changes?

Any contributions should start with some form of communication (see above) to let the SatPy maintainers know how you plan to help. The larger the contribution the more important direct communication is so everyone can avoid duplicate code and wasted time. After talking to the SatPy developers any additional work like code or documentation changes can be provided as a GitHub Pull Request.

8.2 Migrating to xarray and dask

Many python developers dealing with meteorologic satellite data begin with using NumPy arrays directly. This work usually involves masked arrays, boolean masks, index arrays, and reshaping. Due to the libraries used by SatPy these operations can't always be done in the same way. This guide acts as a starting point for new SatPy developers in transitioning from NumPy's array operations to SatPy's operations, although they are very similar.

To provide the most functionality for users, SatPy uses the xarray library's `DataArray` object as the main representation for its data. DataArray objects can also benefit from the dask library. The combination of these libraries allow SatPy to easily distribute operations over multiple workers, lazy evaluate operations, and keep track additional metadata and coordinate information.

8.2.1 Lazy Operations

8.2.2 Indexing

8.2.3 Masks and fill values

8.2.4 Chunks

8.3 Coding guidelines

SatPy tries to follow PEP8 style guidelines for all of its python code. We also try to limit lines of code to 80 characters whenever possible and when it doesn't hurt readability. SatPy follows Google Style Docstrings for all code API documentation. When in doubt use the existing code as a guide for how coding should be done.

SatPy currently supports Python 2.7 and 3.4+. All code should be written to be compatible with these versions.

8.4 Development installation

See the *Installation Instructions* section for basic installation instructions. When it comes time to install SatPy it should be installed from a clone of the git repository and in development mode so that local file changes are automatically reflected in the python environment. We highly recommend making a separate conda environment or virtualenv for development.

First, if you plan on contributing back to the project you should fork the repository and clone your fork. The package can then be installed in development by doing:

```
pip install -e .
```

8.5 Running tests

SatPy tests are written using the python `unittest` module and the tests can be executed by running:

```
python setup.py test
```

8.6 Documentation

SatPy's documentation is built using Sphinx. All documentation lives in the `doc/` directory of the project repository. After editing the source files there the documentation can be generated locally:

```
cd doc
make html
```

The output of the make command should be checked for warnings and errors. If code has been changed (new functions or classes) then the API documentation files should be regenerated before running the above command:

```
sphinx-apidoc -f -T -o source/api ../satpy ../satpy/tests
```

CHAPTER 9

satpy package

9.1 Subpackages

9.1.1 satpy.composites package

Submodules

satpy.composites.abi module

Composite classes for the AHI instrument.

class satpy.composites.abi.**TrueColor**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

Ratio sharpened full resolution true color

class satpy.composites.abi.**TrueColor2km**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

True Color ABI compositor assuming all bands are the same resolution

satpy.composites.abi.**four_element_average**(*d*)

Average every 4 elements (2x2) in a 2D array

satpy.composites.abi.**simulated_green**(*c01*, *c02*, *c03*)

satpy.composites.ahi module

Composite classes for the AHI instrument.

class satpy.composites.ahi.**Expander**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

Expand the size of the composite.

Keyword Arguments **factor** (*int*) – Repeat both dimensions by this number

class satpy.composites.ahi.**GreenCorrector**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

Corrector of the AHI green band to compensate for the deficit of chlorophyl signal.

class satpy.composites.ahi.**Reducer2**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

Reduce the size of the composite.

class satpy.composites.ahi.**Reducer4**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

Reduce the size of the composite.

class satpy.composites.ahi.**Reducer8**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

Reduce the size of the composite.

satpy.composites.crefl_utils module

Shared utilities for correcting reflectance data using the 'crefl' algorithm.

Original code written by Ralph Kuehn with modifications by David Hoese and Martin Raspaud. Ralph's code was originally based on the C crefl code distributed for VIIRS and MODIS.

satpy.composites.crefl_utils.**chand**(*phi*, *muv*, *mus*, *taur*)

satpy.composites.crefl_utils.**csalbr**(*tau*)

satpy.composites.crefl_utils.**find_coefficient_index**(*sensor*, *wavelength_range*, *resolution=0*)

Return index in to coefficient arrays for this band's wavelength.

This function search through the *COEFF_INDEX_MAP* dictionary and finds the first key where the nominal wavelength of *wavelength_range* falls between the minimum wavelength and maximum wavelength of the key. *wavelength_range* can also be the standard name of the band. For example, "M05" for VIIRS or "1" for MODIS.

Parameters

- **sensor** – sensor of band to be corrected
- **wavelength_range** – 3-element tuple of (min wavelength, nominal wavelength, max wavelength)
- **resolution** – resolution of the band to be corrected

Returns index in to coefficient arrays like *aH2O*, *aO3*, etc. None is returned if no matching wavelength is found

satpy.composites.crefl_utils.**get_atm_variables**(*mus*, *muv*, *phi*, *height*, *coeffs*)

satpy.composites.crefl_utils.**get_coefficients**(*sensor*, *wavelength_range*, *resolution=0*)

Parameters

- **sensor** – sensor of the band to be corrected

- **wavelength_range** – 3-element tuple of (min wavelength, nominal wavelength, max wavelength)
- **resolution** – resolution of the band to be corrected

Returns aH2O, bH2O, aO3, taur0 coefficient values

satpy.composites.crefl_utils.**run_crefl**(*refl*, *coeffs*, *lon*, *lat*, *sensor_azimuth*, *sensor_zenith*, *solar_azimuth*, *solar_zenith*, *avg_elevation=None*, *percent=False*)

Run main crefl algorithm.

All input parameters are per-pixel values meaning they are the same size and shape as the input reflectance data, unless otherwise stated.

Parameters

- **reflectance_bands** – tuple of reflectance band arrays
- **coefficients** – tuple of coefficients for each band (see *get_coefficients*)
- **lon** – input swath longitude array
- **lat** – input swath latitude array
- **sensor_azimuth** – input swath sensor azimuth angle array
- **sensor_zenith** – input swath sensor zenith angle array
- **solar_azimuth** – input swath solar azimuth angle array
- **solar_zenith** – input swath solar zenith angle array
- **avg_elevation** – average elevation (usually pre-calculated and stored in CMG-DEM.hdf)
- **percent** – True if input reflectances are on a 0-100 scale instead of 0-1 scale (default: False)

satpy.composites.sar module

Composite classes for the VIIRS instrument.

class satpy.composites.sar.**SARIce**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

The SAR Ice composite.

satpy.composites.sar.**overlay**(*top*, *bottom*)

Blending two layers.

from: https://docs.gimp.org/en/gimp-concepts-layer-modes.html

satpy.composites.viirs module

Composite classes for the VIIRS instrument.

class satpy.composites.viirs.**AdaptiveDNB**(**args*, ***kwargs*)

Bases: *satpy.composites.viirs.HistogramDNB*

Adaptive histogram equalized DNB composite.

The logic for this code was taken from Polar2Grid and was originally developed by Eva Schiffer (SSEC).

This composite separates the DNB data in to 3 main regions: Day, Night, and Mixed. Each region is equalized separately to bring out the most information from the region due to the high dynamic range of the DNB data. Optionally, the mixed region can be separated in to multiple smaller regions by using the *mixed_degree_step* keyword.

class satpy.composites.viirs.**ERFDNB**(**args*, ***kwargs*)
Bases: *satpy.composites.CompositeBase*

Equalized DNB composite using the error function (erf).

The logic for this code was taken from Polar2Grid and was originally developed by Curtis Seaman and Steve Miller. The original code was written in IDL and is included as comments in the code below.

class satpy.composites.viirs.**HistogramDNB**(**args*, ***kwargs*)
Bases: *satpy.composites.CompositeBase*

Histogram equalized DNB composite.

The logic for this code was taken from Polar2Grid and was originally developed by Eva Schiffer (SSEC).

This composite separates the DNB data in to 3 main regions: Day, Night, and Mixed. Each region is equalized separately to bring out the most information from the region due to the high dynamic range of the DNB data. Optionally, the mixed region can be separated in to multiple smaller regions by using the *mixed_degree_step* keyword.

class satpy.composites.viirs.**NCCZinke**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)
Bases: *satpy.composites.CompositeBase*

Equalized DNB composite using the Zinke algorithm.

http://www.tandfonline.com/doi/full/10.1080/01431161.2017.1338838 DOI: 10.1080/01431161.2017.1338838

static gain_factor(*theta*)

class satpy.composites.viirs.**RatioSharpenedRGB**(**args*, ***kwargs*)
Bases: *satpy.composites.RGBCompositor*

class satpy.composites.viirs.**ReflectanceCorrector**(**args*, ***kwargs*)
Bases: *satpy.composites.CompositeBase*

CREFL modifier

Uses a python rewrite of the C CREFL code written for VIIRS and MODIS.

class satpy.composites.viirs.**SnowAge**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)
Bases: *satpy.composites.RGBCompositor*

Returns RGB snow product based on method presented at the second CSPP/IMAPP users' meeting at Eumetsat in Darmstadt on 14-16 April 2015

class satpy.composites.viirs.**VIIRSFog**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)
Bases: *satpy.composites.CompositeBase*

satpy.composites.viirs.**histogram_equalization**(*data*, *mask_to_equalize*, *number_of_bins=1000*, *std_mult_cutoff=4.0*, *do_zerotoone_normalization=True*, *valid_data_mask=None*, *clip_limit=None*, *slope_limit=None*, *do_log_scale=False*, *log_offset=None*, *local_radius_px=None*, *out=None*)

Perform a histogram equalization on the data selected by mask_to_equalize. The data will be separated into number_of_bins levels for equalization and outliers beyond +/- std_mult_cutoff*std will be ignored.

If do_zerotoone_normalization is True the data selected by mask_to_equalize will be returned in the 0 to 1 range. Otherwise the data selected by mask_to_equalize will be returned in the 0 to number_of_bins range.

Note: the data will be changed in place.

satpy.composites.viirs.**local_histogram_equalization**(*data*, *mask_to_equalize*, *valid_data_mask=None*, *number_of_bins=1000*, *std_mult_cutoff=3.0*, *do_zerotoone_normalization=True*, *local_radius_px=300*, *clip_limit=60.0*, *slope_limit=3.0*, *do_log_scale=True*, *log_offset=1e-05*, *out=None*)

equalize the provided data (in the mask_to_equalize) using adaptive histogram equalization tiles of width/height (2 * local_radius_px + 1) will be calculated and results for each pixel will be bilinerarly interpolated from the nearest 4 tiles when pixels fall near the edge of the image (there is no adjacent tile) the resultant interpolated sum from the available tiles will be multipled to account for the weight of any missing tiles (pixel total interpolated value = pixel available interpolated value / (1 - missing interpolation weight))

if do_zerotoone_normalization is True the data will be scaled so that all data in the mask_to_equalize falls between 0 and 1; otherwise the data in mask_to_equalize will all fall between 0 and number_of_bins

returns the equalized data

satpy.composites.viirs.**make_day_night_masks**(*solarZenithAngle*, *good_mask*, *highAngleCutoff*, *lowAngleCutoff*, *stepsDegrees=None*)

given information on the solarZenithAngle for each point, generate masks defining where the day, night, and mixed regions are

optionally provide the highAngleCutoff and lowAngleCutoff that define the limits of the terminator region (if no cutoffs are given the DEFAULT_HIGH_ANGLE and DEFAULT_LOW_ANGLE will be used)

optionally provide the stepsDegrees that define how many degrees each "mixed" mask in the terminator region should be (if no stepsDegrees is given, the whole terminator region will be one mask)

Module contents

Base classes for composite objects.

class satpy.composites.**Airmass**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

class satpy.composites.**BWCompositor**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

class satpy.composites.**CO2Corrector**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

class satpy.composites.**CloudCompositor**(*transition_min=258.15*, *transition_max=298.15*, *transition_gamma=3.0*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

class satpy.composites.**ColorizeCompositor**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.ColormapCompositor*

A compositor colorizing the data, interpolating the palette colors when needed.

class satpy.composites.**ColormapCompositor**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

A compositor that uses colormaps.

static build_colormap(*palette*, *dtype*, *info*)

Create the colormap from the *raw_palette* and the valid_range.

class satpy.composites.**CompositeBase**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.dataset.MetadataObject*

apply_modifier_info(*origin*, *destination*)

class satpy.composites.**CompositorLoader**(*ppp_config_dir='/home/docs/checkouts/readthedocs.org/user_builds/satpy/packages/satpy-0.8.1-py3.5.egg/satpy/etc'*)

Bases: object

Read composites using the configuration files on disk.

get_compositor(*key*, *sensor_names*)

get_modifier(*key*, *sensor_names*)

load_compositors(*sensor_names*)

Load all compositor configs for the provided sensors.

Parameters **sensor_names** (*list of strings*) – Sensor names that have matching `sensor_name.yaml` config files.

Returns

Where *comps* is a dictionary:

sensor_name -> composite ID -> compositor object

And *mods* is a dictionary:

sensor_name -> modifier name -> (modifier class, modifiers options)

Note that these dictionaries are copies of those cached in this object.

Return type (comps, mods)

load_sensor_composites(*sensor_name*)

Load all compositor configs for the provided sensor.

class satpy.composites.**Convection**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

class satpy.composites.**DayNightCompositor**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

A compositor that takes one composite on the night side, another on day side, and then blends them together.

class satpy.composites.**DifferenceCompositor**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

class satpy.composites.**Dust**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

class satpy.composites.**EffectiveSolarPathLengthCorrector**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.SunZenithCorrectorBase*

Special sun zenith correction with the method proposed by Li and Shibata (2006): https://doi.org/10.1175/JAS3682.1

class satpy.composites.**GenericCompositor**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

modes = {1: 'L', 2: 'LA', 3: 'RGB', 4: 'RGBA'}

exception satpy.composites.**IncompatibleAreas**

Bases: Exception

Error raised upon compositing things of different shapes.

exception satpy.composites.**IncompatibleTimes**

Bases: Exception

Error raised upon compositing things from different times.

class satpy.composites.**NIREmissivePartFromReflectance**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.NIRReflectance*

class satpy.composites.**NIRReflectance**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

class satpy.composites.**PSPAtmosphericalCorrection**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

class satpy.composites.**PSPRayleighReflectance**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

class satpy.composites.**PaletteCompositor**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.ColormapCompositor*

A compositor colorizing the data, not interpolating the palette colors.

class satpy.composites.**RGBCompositor**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

class satpy.composites.**RealisticColors**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.GenericCompositor*

class satpy.composites.**SunZenithCorrector**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.SunZenithCorrectorBase*

Standard sun zenith correction, 1/cos(sunz)

class satpy.composites.**SunZenithCorrectorBase**(*name*, *prerequisites=None*, *optional_prerequisites=None*, ***kwargs*)

Bases: *satpy.composites.CompositeBase*

Base class for sun zenith correction

coszen = <WeakValueDictionary>

satpy.composites.**check_times**(*projectables*)

satpy.composites.**enhance2dataset**(*dset*)
Apply enhancements to dataset *dset* and convert the image data back to Dataset object.

satpy.composites.**sub_arrays**(*proj1*, *proj2*)
Substract two DataArrays and combine their attrs.

9.1.2 satpy.enhancements package

Module contents

Enhancements.

satpy.enhancements.**cira_stretch**(*img*, ***kwargs*)
Logarithmic stretch adapted to human vision.

Applicable only for visible channels.

satpy.enhancements.**colorize**(*img*, ***kwargs*)
Colorize the given image.

satpy.enhancements.**create_colormap**(*palette*)
Create colormap of the given numpy file, color vector or colormap.

satpy.enhancements.**gamma**(*img*, ***kwargs*)
Perform gamma correction.

satpy.enhancements.**invert**(*img*, **args*)
Perform inversion.

satpy.enhancements.**lookup**(*img*, ***kwargs*)
Assign values to channels based on a table.

satpy.enhancements.**palettize**(*img*, ***kwargs*)
Palettize the given image (no color interpolation).

satpy.enhancements.**stretch**(*img*, ***kwargs*)
Perform stretch.

satpy.enhancements.**three_d_effect**(*img*, ***kwargs*)
Create 3D effect using convolution

9.1.3 satpy.readers package

Submodules

satpy.readers.aapp_l1b module

Reader for aapp level 1b data.

Options for loading:

- pre_launch_coeffs (False): use pre-launch coefficients if True, operational otherwise (if available).

http://research.metoffice.gov.uk/research/interproj/nwpsaf/aapp/ NWPSAF-MF-UD-003_Formats.pdf

class satpy.readers.aapp_l1b.**AVHRRAAPPL1BFile**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

calibrate(*dataset_ids*, *pre_launch_coeffs=False*, *calib_coeffs=None*)
Calibrate the data

end_time

get_angles(*angle_id*)
Get sun-satellite viewing angles

get_dataset(*key*, *info*)
Get a dataset from the file.

navigate()
Return the longitudes and latitudes of the scene.

read()
Read the data.

shape()

start_time

satpy.readers.aapp_l1b.**create_xarray**(*arr*)

satpy.readers.abi_l1b module

Advance Baseline Imager reader

class satpy.readers.abi_l1b.**NC_ABI_L1B**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

end_time

get_area_def(*key*)
Get the area definition of the data at hand.

get_dataset(*key*, *info*, *xslice=slice(None*, *None*, *None)*, *yslice=slice(None*, *None*, *None)*)
Load a dataset.

get_shape(*key*, *info*)
Get the shape of the data.

start_time

satpy.readers.acspo module

satpy.readers.ahi_hsd module

Advanced Himawari Imager (AHI) standard format data reader

http://www.data.jma.go.jp/mscweb/en/himawari89/space_segment/spsg_ahi.html

class satpy.readers.ahi_hsd.**AHIHSDFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

AHI standard format reader

calibrate(*data*, *calibration*)
Calibrate the data

convert_to_radiance(*data*)
Calibrate to radiance.

end_time

geo_mask(*lineslice=None*, *colslice=None*)
Masking the space pixels from geometry info.

get_area_def(*dsid*)

get_dataset(*key*, *info*, *out=None*, *xslice=slice(None, None, None)*, *yslice=slice(None, None, None)*)

get_lonlats(*key*, *info*, *lon_out*, *lat_out*)

get_shape(*dsid*, *ds_info*)

read_band(*key*, *info*, *out=None*, *xslice=slice(None, None, None)*, *yslice=slice(None, None, None)*)
Read the data

start_time

exception satpy.readers.ahi_hsd.**CalibrationError**
Bases: Exception

satpy.readers.ahi_hsd.**show**(*data*, *negate=False*)
Show the stretched data.

satpy.readers.amsr2_l1b module

Reader for AMSR2 L1B files in HDF5 format.

class satpy.readers.amsr2_l1b.**AMSR2L1BFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.hdf5_utils.HDF5FileHandler*

get_dataset(*ds_id*, *ds_info*)
Get output data and metadata of specified dataset

get_metadata(*ds_id*, *ds_info*)

get_shape(*ds_id*, *ds_info*)
Get output shape of specified dataset

satpy.readers.clavrx module

Interface to CLAVR-X HDF4 products.

class satpy.readers.clavrx.**CLAVRXFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.hdf4_utils.HDF4FileHandler*

available_dataset_ids()
Automatically determine datasets provided by this file

end_time

get_data_type(*dataset_id*, *ds_info*)

get_dataset(*dataset_id*, *ds_info*, *out=None*, *xslice=slice(None, None, None)*, *yslice=slice(None, None, None)*)

get_metadata(*dataset_id*, *ds_info*)

get_nadir_resolution(*sensor*)

get_platform(*platform*)

get_rows_per_scan(*sensor*)

get_sensor(*sensor*)

get_shape(*dataset_id*, *ds_info*)

nadir_resolution = {'modis': 1000, 'avhrr': 1050, 'viirs': 742}

platforms = {'SNPP': 'npp'}

rows_per_scan = {'modis': 10, 'viirs': 16}

sensors = {'AVHRR': 'avhrr', 'VIIRS': 'viirs', 'MODIS': 'modis'}

start_time

class satpy.readers.clavrx.**CLAVRXYAMLReader**(*config_files*, *filter_parameters=None*, *filter_filenames=True*, ***kwargs*)

Bases: *satpy.readers.yaml_reader.FileYAMLReader*

create_filehandlers(*filenames*)

load_ds_ids_from_files()

satpy.readers.eps_l1b module

Reader for eps level 1b data. Uses xml files as a format description.

class satpy.readers.eps_l1b.**EPSAVHRRFile**(*filename*, *filename_info*, *filetype_info*)

Bases: *satpy.readers.file_handlers.BaseFileHandler*

Eps level 1b reader for AVHRR data.

end_time

get_dataset(*key*, *info*)
Get calibrated channel data.

get_full_angles()
Get the interpolated lons/lats.

get_full_lonlats()
Get the interpolated lons/lats.

get_lonlat(*row*, *col*)
Get lons/lats for given indices. WARNING: if the lon/lats were not expanded, this will refer to the tiepoint data.

get_lonlats()

keys()
List of reader's keys.

platform_name

sensor_name

sensors = {'AVHR': 'avhrr-3'}

spacecrafts = {'M02': 'Metop-A', 'M01': 'Metop-B', 'M03': 'Metop-C'}

start_time

satpy.readers.eps_l1b.**create_xarray**(*arr*)

satpy.readers.eps_l1b.**radiance_to_bt**(*arr*, *wc_*, *a__*, *b__*)
Convert to BT.

satpy.readers.eps_l1b.**radiance_to_refl**(*arr*, *solar_flux*)
Convert to reflectances.

satpy.readers.eps_l1b.**read_raw**(*filename*)
Read *filename* without scaling it afterwards.

satpy.readers.fci_fdhsi module

Interface to MTG-FCI Retrieval NetCDF files

class satpy.readers.fci_fdhsi.**FCIFDHSIFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

MTG FCI FDHSI File Reader

calc_area_extent(*key*)
Calculate area extent for a dataset.

calibrate(*data*, *key*)
Data calibration.

end_time

get_area_def(*key*, *info=None*)
Calculate on-fly area definition for 0 degree geos-projection for a dataset

get_dataset(*key*, *info=None*)
Load a dataset

start_time

satpy.readers.file_handlers module

class satpy.readers.file_handlers.**BaseFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: object

combine_info(*all_infos*)
Combine metadata for multiple datasets.

When loading data from multiple files it can be non-trivial to combine things like start_time, end_time, start_orbit, end_orbit, etc.

By default this method will produce a dictionary containing all values that were equal across **all** provided info dictionaries.

Additionally it performs the logical comparisons to produce the following if they exist:

- start_time
- end_time

- start_orbit
- end_orbit

Also, concatenate the areas.

end_time

get_area_def(*dsid*)

get_bounding_box()
Get the bounding box of the files, as a (lons, lats) tuple.

The tuple return should a lons and lats list of coordinates traveling clockwise around the points available in the file.

get_dataset(*dataset_id*, *ds_info*, *out=None*, *xslice=slice(None, None, None)*, *yslice=slice(None, None, None)*)

get_shape(*dataset_id*, *ds_info*)

start_time

satpy.readers.gac_lac_l1 module

Reading and calibrating GAC and LAC avhrr data. Todo: - Fine grained calibration

class satpy.readers.gac_lac_l1.**GACLACFile**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

Reader for GAC and LAC data.

end_time

get_dataset(*key*, *info*)

start_time

satpy.readers.generic_image module

Reader for generic image (e.g. gif, png, jpg).

It returns a dataset without coordinates and calibration.

class satpy.readers.generic_image.**GenericImageFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

end_time

get_dataset(*key*, *info*, *out=None*)
Get a dataset from the file.

read(*filename*)

start_time

satpy.readers.geocat module

satpy.readers.ghrsst_osisaf module

satpy.readers.hdf4_caliopv3 module

class satpy.readers.hdf4_caliopv3.**HDF4BandReader**(*filename*, *filename_info*, *filetype_info*)

Bases: *satpy.readers.file_handlers.BaseFileHandler*

CALIOP v3 HDF4 reader.

end_time

get_dataset(*key*, *info*)
Read data from file and return the corresponding projectables.

get_end_time()
Get observation end time from file metadata.

get_filehandle()
Get HDF4 filehandle.

get_lonlats()
Get longitude and latitude arrays from the file.

get_sds_variable(*name*)
Read variable from the HDF4 file.

static parse_metadata_string(*metadata_string*)
Grab end time with regular expression.

start_time

satpy.readers.hdf4_utils module

Helpers for reading hdf4-based files.

class satpy.readers.hdf4_utils.**HDF4FileHandler**(*filename*, *filename_info*, *filetype_info*)

Bases: *satpy.readers.file_handlers.BaseFileHandler*

Small class for inspecting a HDF5 file and retrieve its metadata/header data.

collect_metadata(*name*, *obj*)

get(*item*, *default=None*)

satpy.readers.hdf5_utils module

Helpers for reading hdf5-based files.

class satpy.readers.hdf5_utils.**HDF5FileHandler**(*filename*, *filename_info*, *filetype_info*)

Bases: *satpy.readers.file_handlers.BaseFileHandler*

Small class for inspecting a HDF5 file and retrieve its metadata/header data.

collect_metadata(*name*, *obj*)

get(*item*, *default=None*)

satpy.readers.hdfeos_l1b module

Interface to Modis level 1b format. http://www.icare.univ-lille1.fr/wiki/index.php/MODIS_geolocation http://www.sciencedirect.com/science?_ob=MiamiImageURL&_imagekey=B6V6V-4700BJP-3-27&_cdi=5824&_user=671124&_check=y&_orig=search&_coverDate=11%2F30%2F2002&view=c&wchp=dGLzVlz-zSkWz&md5=bac5bc7a4f08007722ae793954f1dd63&ie=/sdarticle.pdf

class satpy.readers.hdfeos_l1b.**HDFEOSBandReader**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.hdfeos_l1b.HDFEOSFileReader*

get_dataset(*key*, *info*)
Read data from file and return the corresponding projectables.

get_height()

get_sata()

get_satz()

get_suna()

get_sunz()

res = {'1': 1000, 'H': 500, 'Q': 250}

class satpy.readers.hdfeos_l1b.**HDFEOSFileReader**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

end_time

read_mda(*attribute*)

start_time

class satpy.readers.hdfeos_l1b.**HDFEOSGeoReader**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.hdfeos_l1b.HDFEOSFileReader*

get_dataset(*key*, *info*, *out=None*, *xslice=None*, *yslice=None*)
Get the dataset designated by *key*.

load(*keys*, *interpolate=True*, *raw=False*)
Load the data.

satpy.readers.hdfeos_l1b.**calibrate_refl**(*subdata*, *uncertainty*, *indices*)
Calibration for reflective channels.

satpy.readers.hdfeos_l1b.**calibrate_tb**(*subdata*, *uncertainty*, *indices*, *band_names*)
Calibration for the emissive channels.

satpy.readers.hrit_base module

HRIT format reader

class satpy.readers.hrit_base.**HRITFileHandler**(*filename*, *filename_info*, *filetype_info*, *hdr_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

HRIT standard format reader.

end_time

get_area_def(*dsid*)
Get the area definition of the band.

get_area_extent(*size*, *offsets*, *factors*, *platform_height*)
Get the area extent of the file.

get_dataset(*key*, *info*)
Load a dataset.

get_shape(*dsid*, *ds_info*)

get_xy_from_linecol(*line*, *col*, *offsets*, *factors*)
Get the intermediate coordinates from line & col.

Intermediate coordinates are actually the instruments scanning angles.

read_band(*key*, *info*)
Read the data

start_time

satpy.readers.hrit_base.**dec10216**(*inbuf*)

```
/*
 * pack 4 10-bit words in 5 bytes into 4 16-bit words
 *
 * 0        1        2        3        4        5
 * 0123456789012345678901234567890123456789012
 * 0          1          2          3          4
 */
ip = &in_buffer[i];
op = &out_buffer[j];
op[0] = ip[0]*4 + ip[1]/64;
op[1] = (ip[1] & 0x3F)*16 + ip[2]/16;
op[2] = (ip[2] & 0x0F)*64 + ip[3]/4;
op[3] = (ip[3] & 0x03)*256 +ip[4];
```

satpy.readers.hrit_base.**make_time_cds_short**(*tcds_array*)

satpy.readers.hrit_electrol module

HRIT format reader.

References

ELECTRO-L GROUND SEGMENT MSU-GS INSTRUMENT, LRIT/HRIT Mission Specific Implementation, February 2012

class satpy.readers.hrit_electrol.**HRITGOMSEpilogueFileHandler**(*filename*, *filename_info*, *filetype_info*)

Bases: *satpy.readers.hrit_base.HRITFileHandler*

GOMS HRIT format reader.

read_epilogue()
Read the prologue metadata.

class satpy.readers.hrit_electrol.**HRITGOMSFileHandler**(*filename*, *filename_info*, *filetype_info*, *prologue*, *epilogue*)

Bases: *satpy.readers.hrit_base.HRITFileHandler*

GOMS HRIT format reader.

calibrate(*data*, *calibration*)
Calibrate the data.

get_area_def(*dsid*)
Get the area definition of the band.

get_dataset(*key*, *info*, *out=None*, *xslice=slice(None, None, None)*, *yslice=slice(None, None, None)*)
Get the data from the files.

class satpy.readers.hrit_electrol.**HRITGOMSPrologueFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.hrit_base.HRITFileHandler*

GOMS HRIT format reader.

process_prologue()
Reprocess prologue to correct types.

read_prologue()
Read the prologue metadata.

satpy.readers.hrit_electrol.**recarray2dict**(*arr*)

satpy.readers.hrit_goes module

HRIT format reader.

References

LRIT/HRIT Mission Specific Implementation, February 2012 GVARRDL98.pdf 05057_SPE_MSG_LRIT_HRI

exception satpy.readers.hrit_goes.**CalibrationError**
Bases: Exception

class satpy.readers.hrit_goes.**HRITGOESFileHandler**(*filename*, *filename_info*, *filetype_info*, *prologue*)
Bases: *satpy.readers.hrit_base.HRITFileHandler*

GOES HRIT format reader.

calibrate(*data*, *calibration*)
Calibrate the data.

get_area_def(*dsid*)
Get the area definition of the band.

get_dataset(*key*, *info*)
Get the data from the files.

class satpy.readers.hrit_goes.**HRITGOESPrologueFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.hrit_base.HRITFileHandler*

GOES HRIT format reader

process_prologue()
Reprocess prologue to correct types.

read_prologue()
Read the prologue metadata.

satpy.readers.hrit_goes.**make_gvar_float**(*float_val*)

satpy.readers.hrit_goes.**make_sgs_time**(*sgs_time_array*)

satpy.readers.hrit_goes.**recarray2dict**(*arr*)

satpy.readers.hrit_goes.**show**(*data*, *negate=False*)
Show the stretched data.

satpy.readers.hrit_jma module

HRIT format reader for JMA data.

References

JMA HRIT - Mission Specific Implementation http://www.jma.go.jp/jma/jma-eng/satellite/introduction/4_2HRIT.pdf

class satpy.readers.hrit_jma.**HRITJMAFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.hrit_base.HRITFileHandler*

JMA HRIT format reader

calibrate(*data*, *calibration*)
Calibrate the data.

get_area_def(*dsid*)
Get the area definition of the band.

get_dataset(*key*, *info*)
Get the dataset designated by *key*.

satpy.readers.hrit_jma.**recarray2dict**(*arr*)

satpy.readers.hrit_jma.**show**(*data*, *negate=False*)
Show the stretched data.

satpy.readers.hrit_msg module

HRIT format reader.

References

MSG Level 1.5 Image Data FormatDescription

TODO: - HRV navigation

class satpy.readers.hrit_msg.**HRITMSGEpilogueFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.hrit_base.HRITFileHandler*

MSG HRIT format reader

read_epilogue()
Read the prologue metadata.

class satpy.readers.hrit_msg.**HRITMSGFileHandler**(*filename*, *filename_info*, *filetype_info*, *prologue*, *epilogue*)

Bases: *satpy.readers.hrit_base.HRITFileHandler*

MSG HRIT format reader

calibrate(*data*, *calibration*)
Calibrate the data.

convert_to_radiance(*data*)
Calibrate to radiance.

end_time

get_area_def(*dsid*)
Get the area definition of the band.

get_area_extent(*size*, *offsets*, *factors*, *platform_height*)
Get the area extent of the file.

get_dataset(*key*, *info*)

get_xy_from_linecol(*line*, *col*, *offsets*, *factors*)
Get the intermediate coordinates from line & col.

Intermediate coordinates are actually the instruments scanning angles.

start_time

class satpy.readers.hrit_msg.**HRITMSGPrologueFileHandler**(*filename*, *filename_info*, *filetype_info*)

Bases: *satpy.readers.hrit_base.HRITFileHandler*

MSG HRIT format reader

process_prologue()
Reprocess prologue to correct types.

read_prologue()
Read the prologue metadata.

satpy.readers.hrit_msg.**make_time_cds_expanded**(*tcds_array*)

satpy.readers.hrit_msg.**recarray2dict**(*arr*)

satpy.readers.hrit_msg.**show**(*data*, *negate=False*)
Show the stretched data.

satpy.readers.hrpt module

Reading and calibrating hrpt avhrr data. Todo: - AMSU - Compare output with AAPP

Reading: http://www.ncdc.noaa.gov/oa/pod-guide/ncdc/docs/klm/html/c4/sec4-1.htm#t413-1

Calibration: http://www.ncdc.noaa.gov/oa/pod-guide/ncdc/docs/klm/html/c7/sec7-1.htm

class satpy.readers.hrpt.**HRPTFile**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

Reader for HRPT Minor Frame, 10 bits data expanded to 16 bits.

end_time

get_dataset(*key*, *info*)

get_lonlats()

get_telemetry()

read()

start_time

satpy.readers.hrpt.**bfield**(*array*, *bit*)
return the bit array.

satpy.readers.hrpt.**geo_interpolate**(*lons32km*, *lats32km*)

satpy.readers.hrpt.**time_seconds**(*tc_array*, *year*)
Return the time object from the timecodes

satpy.readers.iasi_l2 module

IASI L2 HDF5 files.

class satpy.readers.iasi_l2.**IASIL2HDF5**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

File handler for IASI L2 HDF5 files.

end_time

get_dataset(*key*, *info*)
Load a dataset

start_time

satpy.readers.iasi_l2.**read_dataset**(*fid*, *key*, *info*)
Read dataset

satpy.readers.iasi_l2.**read_geo**(*fid*, *key*, *info*)
Read geolocation and related datasets.

satpy.readers.li_l2 module

Interface to MTG-LI L2 product NetCDF files

The reader is based on preliminary test data provided by EUMETSAT. The data description is described in the "LI L2 Product User Guide [LIL2PUG] Draft version" documentation.

class satpy.readers.li_l2.**LIFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

MTG LI File Reader

end_time

get_area_def(*key*, *info=None*)
Projection information are hard coded for 0 degree geos projection Test dataset doen't provide the values in the file container. Only fill values are inserted

get_dataset(*key*, *info=None*, *out=None*, *xslice=None*, *yslice=None*)
Load a dataset

get_shape(*dsid*, *ds_info*)

start_time

satpy.readers.maia module

Reader for NWPSAF AAPP MAIA Cloud product.

https://nwpsaf.eu/site/software/aapp/

Documentation reference:

> [NWPSAF-MF-UD-003] DATA Formats [NWPSAF-MF-UD-009] MAIA version 4 Scientific User Manual

class satpy.readers.maia.**MAIAFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

end_time

get_dataset(*key*, *info*, *out=None*)
Get a dataset from the file.

get_platform(*platform*)

read(*filename*)

start_time

satpy.readers.msg_base module

Utilities and eventually also base classes for MSG HRIT/Native data reading

satpy.readers.msg_base.**convert_to_radiance**(*data*, *gain*, *offset*)
Calibrate to radiance.

satpy.readers.msg_base.**dec10216**(*data*)
Unpacking the 10 bit data to 16 bit

satpy.readers.msg_base.**erads2bt**(*data*, *wavenumber*, *alpha*, *beta*)

satpy.readers.msg_base.**get_cds_time**(*days*, *msecs*)
Get the datetime object of the time since epoch given in days and milliseconds of day

satpy.readers.msg_base.**srads2bt**(*data*, *wavenumber*, *a__*, *b__*, *c__*)

satpy.readers.msg_base.**tl15**(*data*, *wavenumber*)
Compute the L15 temperature.

satpy.readers.msg_base.**vis_calibrate**(*data*, *solar_irradiance*)

satpy.readers.native_msg module

A reader for the EUMETSAT MSG native format

https://www.eumetsat.int/website/wcm/idc/idcplg?IdcService=GET_FILE&dDocName=PDF_FG15_MSG-NATIVE-FORMAT-15&RevisionSelectionMethod=LatestReleased&Rendition=Web

exception satpy.readers.native_msg.**CalibrationError**
Bases: Exception

class satpy.readers.native_msg.**NativeMSGFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

Native MSG format reader

calibrate(*data*, *key*)
Calibrate the data.

convert_to_radiance(*data*, *key_name*)
Calibrate to radiance.

end_time

get_area_def(*dsid*)

get_dataset(*key*, *info*, *xslice=slice(None, None, None)*, *yslice=slice(None, None, None)*)

start_time

satpy.readers.native_msg_hdr module

Definition of Header Records for the MSG Level 1.5 data (hrit or native)

Warning: *impf_configuration* in *L15DataHeaderRecord* class needs to be fixed!

class satpy.readers.native_msg_hdr.**GSDTRecords**
Bases: object

MSG Ground Segment Data Type records.

Reference Document: MSG Ground Segment Design Specification (GSDS)

gp_cpu_address

gp_fac_env

gp_fac_id

gp_pk_header

gp_pk_sh1

gp_sc_id

gp_su_id

gp_svce_type

time_cds

time_cds_expanded

time_cds_short

class satpy.readers.native_msg_hdr.**L15DataHeaderRecord**
Bases: *satpy.readers.native_msg_hdr.GSDTRecords*

Reference Document: MSG Level 1.5 Image Data Format Description

celestial_events

geometric_processing

get(*umarf=True*)

image_acquisition

image_description

impf_configuration

radiometric_processing

satellite_status

class satpy.readers.native_msg_hdr.**L15MainProductHeaderRecord**
Bases: *satpy.readers.native_msg_hdr.L15PhData*

Reference Document: MSG Level 1.5 Native Format File Definition

get()

l15_ph_data_identification

class satpy.readers.native_msg_hdr.**L15PhData**
Bases: object

l15_ph_data

class satpy.readers.native_msg_hdr.**L15SecondaryProductHeaderRecord**
Bases: *satpy.readers.native_msg_hdr.L15PhData*

Reference Document: MSG Level 1.5 Native Format File Definition

get()

class satpy.readers.native_msg_hdr.**Msg15NativeHeaderRecord**
Bases: object

get()

satpy.readers.nc_nwcsaf module

Nowcasting SAF common PPS&MSG NetCDF/CF format reader

class satpy.readers.nc_nwcsaf.**NcNWCSAF**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

NWCSAF PPS&MSG NetCDF reader.

end_time
Return the end time of the object.

get_area_def(*dsid*)
Get the area definition of the datasets in the file.

Only applicable for MSG products!

get_dataset(*dsid*, *info*)
Load a dataset.

start_time
Return the start time of the object.

satpy.readers.nc_nwcsaf.**remove_empties**(*variable*)
Remove empty objects from the *variable*'s attrs.

satpy.readers.nc_nwcsaf_msg module

Nowcasting SAF MSG NetCDF4 format reader

class satpy.readers.nc_nwcsaf_msg.**NcNWCSAFMSG**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

NWCSAF MSG NetCDF reader.

end_time

get_area_def(*dsid*)
Get the area definition of the datasets in the file.

get_dataset(*key*, *info*)
Load a dataset.

start_time

satpy.readers.nc_olci module

Sentinel-3 OLCI reader

class satpy.readers.nc_olci.**NCOLCI1B**(*filename*, *filename_info*, *filetype_info*, *cal*)
Bases: *satpy.readers.nc_olci.NCOLCIChannelBase*

get_dataset(*key*, *info*)
Load a dataset.

class satpy.readers.nc_olci.**NCOLCI2**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.nc_olci.NCOLCIChannelBase*

get_dataset(*key*, *info*)
Load a dataset

class satpy.readers.nc_olci.**NCOLCIAngles**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

datasets = {'solar_zenith_angle': 'SZA', 'solar_azimuth_angle': 'SAA', 'satellite_az

end_time

get_dataset(*key*, *info*)
Load a dataset.

start_time

class satpy.readers.nc_olci.**NCOLCIBase**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

end_time

get_dataset(*key*, *info*)
Load a dataset.

start_time

class satpy.readers.nc_olci.**NCOLCICal**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.nc_olci.NCOLCIBase*

class satpy.readers.nc_olci.**NCOLCIChannelBase**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.nc_olci.NCOLCIBase*

class satpy.readers.nc_olci.**NCOLCIGeo**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.nc_olci.NCOLCIBase*

satpy.readers.nc_slstr module

Compact viirs format.

class satpy.readers.nc_slstr.**NCSLSTR1B**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

end_time

get_dataset(*key*, *info*)
Load a dataset.

start_time

class satpy.readers.nc_slstr.**NCSLSTRAngles**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

datasets = {'solar_zenith_angle': 'solar_zenith_tn', 'solar_azimuth_angle': 'solar_a

end_time

get_dataset(*key*, *info*)
Load a dataset

start_time

view = 'n'

class satpy.readers.nc_slstr.**NCSLSTRGeo**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

end_time

get_dataset(*key*, *info*)
Load a dataset

start_time

satpy.readers.netcdf_utils module

satpy.readers.nucaps module

satpy.readers.omps_edr module

Interface to OMPS EDR format

class satpy.readers.omps_edr.**EDREOSFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.omps_edr.EDRFileHandler*

class satpy.readers.omps_edr.**EDRFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.hdf5_utils.HDF5FileHandler*

adjust_scaling_factors(*factors*, *file_units*, *output_units*)

end_orbit_number

get_dataset(*dataset_id*, *ds_info*)

get_metadata(*dataset_id*, *ds_info*)

get_shape(*ds_id*, *ds_info*)

platform_name

sensor_name

start_orbit_number

satpy.readers.safe_msi module

satpy.readers.safe_sar_c module

SAFE SAR-C format.

class satpy.readers.safe_sar_c.**SAFEGRD**(*filename*, *filename_info*, *filetype_info*, *calfh*, *noisefh*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

Measurement file reader.

end_time

get_dataset(*key*, *info*)
Load a dataset.

get_gcps()
Read GCP from the GDAL band.

Parameters

- **band** (*gdal band*) – Measurement band which comes with GCP's
- **coordinates** (*tuple*) – A tuple with longitude and latitude arrays

Returns Pixel and Line indices 1d arrays gcp_coords (tuple): longitude and latitude 1d arrays

Return type points (tuple)

get_gdal_filehandle()
Try to create the filehandle using gdal.

get_lonlats()
Obtain GCPs and construct latitude and longitude arrays.

Parameters

- **band** (*gdal band*) – Measurement band which comes with GCP's
- **array_shape** (*tuple*) – The size of the data array

Returns A tuple with longitude and latitude arrays

Return type coordinates (tuple)

read_band(*blocksize=4096*)
Read the band in blocks.

start_time

class satpy.readers.safe_sar_c.**SAFEXML**(*filename*, *filename_info*, *filetype_info*, *header_file=None*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

XML file reader for the SAFE format.

end_time

get_calibration(*name*, *shape*)
Get the calibration array.

get_calibration_constant()
Load the calibration constant.

get_dataset(*key*, *info*)
Load a dataset.

get_metadata()
Convert the xml metadata to dict.

get_noise_correction(*shape*)
Get the noise correction array.

static interpolate_xml_array(*data*, *low_res_coords*, *shape*)
Interpolate arbitrary size dataset to a full sized grid.

static read_xml_array(*elts*, *variable_name*)
Read an array from an xml elements *elts*.

start_time

satpy.readers.safe_sar_c.**dictify**(*r*, *root=True*)
Convert an ElementTree into a dict.

satpy.readers.safe_sar_c.**interpolate_slice**(*slice_rows*, *slice_cols*, *interpolator*)
Interpolate the given slice of the larger array.

satpy.readers.safe_sar_c.**interpolate_xarray**(*xpoints*, *ypoints*, *values*, *shape*, *kind='cubic'*, *blocksize=4096*)
Interpolate, generating a dask array.

satpy.readers.safe_sar_c.**interpolate_xarray_linear**(*xpoints*, *ypoints*, *values*, *shape*, *blocksize=4096*)
Interpolate linearly, generating a dask array.

satpy.readers.scatsat1_l2b module

ScatSat-1 L2B Reader, distributed by Eumetsat in HDF5 format

class satpy.readers.scatsat1_l2b.**SCATSAT1L2BFileHandler**(*filename*, *filename_info*, *filetype_info*)
Bases: *satpy.readers.file_handlers.BaseFileHandler*

get_dataset(*key*, *info*)

satpy.readers.utils module

Helper functions for area extent calculations.

satpy.readers.utils.**get_area_slices**(*data_area*, *area_to_cover*)
Compute the slice to read from an *area* based on an *area_to_cover*.

satpy.readers.utils.**get_geostationary_angle_extent**(*geos_area*)
Get the max earth (vs space) viewing angles in x and y.

satpy.readers.utils.**get_geostationary_bounding_box**(*geos_area*, *nb_points=50*)
Get the bbox in lon/lats of the valid pixels inside *geos_area*.

Parameters **nb_points** – Number of points on the polygon

satpy.readers.utils.**get_sub_area**(*area*, *xslice*, *yslice*)
Apply slices to the area_extent and size of the area.

satpy.readers.utils.**np2str**(*value*)
Convert an *numpy.string_* to str.

Parameters **value** (*ndarray*) – scalar or 1-element numpy array to convert

Raises ValueError – if value is array larger than 1-element or it is not of type *numpy.string_* or it is not a numpy array

satpy.readers.viirs_compact module

Compact viirs format.

class satpy.readers.viirs_compact.**VIIRSCompactFileHandler**(*filename*, *filename_info*, *filetype_info*)

Bases: satpy.readers.file_handlers.BaseFileHandler

angles(*azi_name*, *zen_name*)

end_time

get_bounding_box()

get_dataset(*key*, *info*)
Load a dataset

navigate()

read_dataset(*dataset_key*, *info*)

read_geo(*key*, *info*)
Read angles.

start_time

satpy.readers.viirs_compact.**expand_array**(*data*, *scans*, *c_align*, *c_exp*, *scan_size=16*, *tpz_size=16*, *nties=200*, *track_offset=0.5*, *scan_offset=0.5*)

Expand *data* according to alignment and expansion.

satpy.readers.viirs_compact.**navigate_dnb**(*h5f*)

satpy.readers.viirs_compact.**read_dnb**(*h5f*)

satpy.readers.viirs_l1b module

satpy.readers.viirs_sdr module

Interface to VIIRS SDR format

Format documentation: http://npp.gsfc.nasa.gov/science/sciencedocuments/082012/474-00001-03_CDFCBVolIII_RevC.pdf

class satpy.readers.viirs_sdr.**VIIRSSDRFileHandler**(*filename*, *filename_info*, *filetype_info*)

Bases: satpy.readers.hdf5_utils.HDF5FileHandler

VIIRS HDF5 File Reader

adjust_scaling_factors(*factors*, *file_units*, *output_units*)

end_orbit_number

end_time

get_bounding_box()
Get the bounding box of this file.

get_dataset(*dataset_id*, *ds_info*)

get_file_units(*dataset_id*, *ds_info*)

get_shape(*ds_id*, *ds_info*)

platform_name

scale_swath_data(*data*, *scaling_factors*)
Scale swath data using scaling factors and offsets.

Multi-granule (a.k.a. aggregated) files will have more than the usual two values.

sensor_name

start_orbit_number

start_time

class satpy.readers.viirs_sdr.**VIIRSSDRReader**(*config_files*, *use_tc=True*, ***kwargs*)
Bases: *satpy.readers.yaml_reader.FileYAMLReader*

Custom file reader for finding VIIRS SDR geolocation at runtime.

satpy.readers.xmlformat module

Reads a format from an xml file to create dtypes and scaling factor arrays.

class satpy.readers.xmlformat.**XMLFormat**(*filename*)
Bases: object

XMLFormat object.

apply_scales(*array*)
Apply scales to *array*.

dtype(*key*)
Get the dtype for the format object.

satpy.readers.xmlformat.**parse_format**(*xml_file*)
Parse the xml file to create types, scaling factor types, and scales.

satpy.readers.xmlformat.**process_array**(*elt*, *ascii=False*)
Process an 'array' tag.

satpy.readers.xmlformat.**process_delimiter**(*elt*, *ascii=False*)
Process a 'delimiter' tag.

satpy.readers.xmlformat.**process_field**(*elt*, *ascii=False*)
Process a 'field' tag.

satpy.readers.xmlformat.**to_dtype**(*val*)
Parse *val* to return a dtype.

satpy.readers.xmlformat.**to_scaled_dtype**(*val*)
Parse *val* to return a dtype.

satpy.readers.xmlformat.**to_scales**(*val*)
Parse *val* to return an array of scale factors.

satpy.readers.yaml_reader module

class satpy.readers.yaml_reader.**AbstractYAMLReader**(*config_files*)
Bases: object

all_dataset_ids

all_dataset_names

available_dataset_ids

available_dataset_names

static dfilter_from_key(*dfilter*, *key*)
Create a dataset filter from a *key*.

end_time
End time of the reader.

filter_ds_ids(*dataset_ids*, *dfilter*)
Filter *dataset_ids* based on *dfilter*.

filter_selected_filenames(*filenames*)
Filter provided filenames by parameters in reader configuration.

Returns: iterable of usable files

get_dataset_key(*key*, *dfilter=None*, *aslist=False*)
Get the fully qualified dataset id corresponding to *key*.

Can be either by name or centerwavelength. If *key* is a *DatasetID* object its name is searched if it exists, otherwise its wavelength is used.

get_ds_ids_by_id(*dsid*, *ids=None*)
Get the dataset ids matching a given a dataset id.

get_ds_ids_by_name(*name*, *ids=None*)
Get the datasets ids by *name*.

get_ds_ids_by_wavelength(*wavelength*, *ids=None*)
Get the dataset ids matching a given a *wavelength*.

load(*dataset_keys*)
Load *dataset_keys*.

load_ds_ids_from_config()
Get the dataset ids from the config.

select_files_from_directory(*directory=None*)
Find files for this reader in *directory*.

If directory is None or ‘’, look in the current directory.

select_files_from_pathnames(*filenames*)
Select the files from *filenames* this reader can handle.

start_time
Start time of the reader.

supports_sensor(*sensor*)
Check if *sensor* is supported.

Returns True is *sensor* is None.

class satpy.readers.yaml_reader.**FileYAMLReader**(*config_files*, *filter_parameters=None*, *filter_filenames=True*, ***kwargs*)
Bases: *satpy.readers.yaml_reader.AbstractYAMLReader*

Implementation of the YAML reader.

available_dataset_ids

static check_file_covers_area(*file_handler*, *check_area*)
Checks if the file covers the current area.

If the file doesn't provide any bounding box information or 'area' was not provided in *filter_parameters*, the check returns True.

create_filehandlers(*filenames*)
Organize the filenames into file types and create file handlers.

end_time

static filename_items_for_filetype(*filenames*, *filetype_info*)
Iterator over the filenames matching *filetype_info*.

filter_fh_by_metadata(*filehandlers*)
Filter out filehandlers using provide filter parameters.

filter_filenames_by_info(*filename_items*)
Filter out file using metadata from the filenames.

Currently only uses start and end time. If only start time is available from the filename, keep all the filename that have a start time before the requested end time.

filter_selected_filenames(*filenames*)

find_required_filehandlers(*requirements*, *filename_info*)
Find the necessary fhs for the current filehandler.

We assume here requirements are available.

load(*dataset_keys*, *previous_datasets=None*)
Load *dataset_keys*.

If *previous_datasets* is provided, do not reload those.

metadata_matches(*sample_dict*, *file_handler=None*)

new_filehandler_instances(*filetype_info*, *filename_items*)
Generate new filehandler instances.

new_filehandlers_for_filetype(*filetype_info*, *filenames*)
Create filehandlers for a given filetype.

sorted_filetype_items()
Sort the instance's filetypes in using order.

start_time

time_matches(*fstart*, *fend*)

class satpy.readers.yaml_reader.**Shuttle**(*data*, *mask*, *info*)
Bases: tuple

data
Alias for field number 0

info
Alias for field number 2

mask
Alias for field number 1

satpy.readers.yaml_reader.**get_filebase**(*path*, *pattern*)
Get the end of *path* of same length as *pattern*.

satpy.readers.yaml_reader.**listify_string**(*something*)
Takes *something* and make it a list.

something is either a list of strings or a string, in which case the function returns a list containing the string. If *something* is None, an empty list is returned.

satpy.readers.yaml_reader.**match_filenames**(*filenames*, *pattern*)
Get the filenames matching *pattern*.

Module contents

Shared objects of the various reader classes.

class satpy.readers.**DatasetDict**(**args*, ***kwargs*)
Bases: dict

Special dictionary object that can handle dict operations based on dataset name, wavelength, or DatasetID.

Note: Internal dictionary keys are *DatasetID* objects.

get_best_choice(*key*, *choices*)

get_item(*name_or_wl*, *resolution=None*, *polarization=None*, *calibration=None*, *modifiers=None*)

get_key(*key*)

get_keys(*name_or_wl*, *resolution=None*, *polarization=None*, *calibration=None*, *modifiers=None*)

get_keys_by_datasetid(*did*)

keys(*names=False*, *wavelengths=False*)

exception satpy.readers.**MalformedConfigError**
Bases: Exception

satpy.readers.**configs_for_reader**(*reader=None*, *ppp_config_dir=None*)
Generator of reader configuration files for one or more readers

Parameters

- **reader** (*Optional[str]*) – Yield configs only for this reader
- **ppp_config_dir** (*Optional[str]*) – Additional configuration directory to search for reader configuration files.

Returns: Generator of lists of configuration files

satpy.readers.**find_files_and_readers**(*start_time=None*, *end_time=None*, *base_dir=None*, *reader=None*, *sensor=None*, *ppp_config_dir='/home/docs/checkouts/readthedocs.org/user_builds/satpy/envs/*
packages/satpy-0.8.1-py3.5.egg/satpy/etc', *filter_parameters=None*, *reader_kwargs=None*)
Find on-disk files matching the provided parameters.

Use *start_time* and/or *end_time* to limit found filenames by the times in the filenames (not the internal file metadata). Files are matched if they fall anywhere within the range specified by these parameters.

Searching is **NOT** recursive.

The returned dictionary can be passed directly to the *Scene* object through the *filenames* keyword argument.

Parameters

- **start_time** (*datetime*) – Limit used files by starting time.

- **end_time** (*datetime*) – Limit used files by ending time.
- **base_dir** (*str*) – The directory to search for files containing the data to load. Defaults to the current directory.
- **reader** (*str or list*) – The name of the reader to use for loading the data or a list of names.
- **sensor** (*str or list*) – Limit used files by provided sensors.
- **ppp_config_dir** (*str*) – The directory containing the configuration files for SatPy.
- **filter_parameters** (*dict*) – Filename pattern metadata to filter on. *start_time* and *end_time* are automatically added to this dictionary. Shortcut for *reader_kwargs['filter_parameters']*.
- **reader_kwargs** (*dict*) – Keyword arguments to pass to specific reader instances to further configure file searching.

Returns: Dictionary mapping reader name string to list of filenames

satpy.readers.**load_reader**(*reader_configs*, ***reader_kwargs*)
Import and setup the reader from *reader_info*

satpy.readers.**load_readers**(*filenames=None*, *reader=None*, *reader_kwargs=None*, *ppp_config_dir='/home/docs/checkouts/readthedocs.org/user_builds/satpy/envs/latest/lib/python3.5/site-packages/satpy-0.8.1-py3.5.egg/satpy/etc'*)
Create specified readers and assign files to them.

Parameters

- **filenames** (*iterable or dict*) – A sequence of files that will be used to load data from. A `dict` object should map reader names to a list of filenames for that reader.
- **reader** (*str or list*) – The name of the reader to use for loading the data or a list of names.
- **filter_parameters** (*dict*) – Specify loaded file filtering parameters. Shortcut for *reader_kwargs['filter_parameters']*.
- **reader_kwargs** (*dict*) – Keyword arguments to pass to specific reader instances.
- **ppp_config_dir** (*str*) – The directory containing the configuration files for satpy.

Returns: Dictionary mapping reader name to reader instance

satpy.readers.**read_reader_config**(*config_files*, *loader=<class 'yaml.loader.Loader'>*)
Read the reader *config_files* and return the info extracted.

9.1.4 satpy.writers package

Submodules

satpy.writers.cf_writer module

Writer for netCDF4/CF.

class satpy.writers.cf_writer.**CFWriter**(*name=None*, *fill_value=None*, *file_pattern=None*, *base_dir=None*, ***kwargs*)
Bases: *satpy.writers.Writer*

Writer producing NetCDF/CF compatible datasets.

static da2cf(*dataarray*, *epoch='seconds since 1970-01-01 00:00:00 +00:00'*)
Convert the dataarray to something cf-compatible.

save_dataset(*dataset*, *filename=None*, *fill_value=None*, ***kwargs*)
Save the *dataset* to a given *filename*.

save_datasets(*datasets*, *filename*, ***kwargs*)
Save all datasets to one or more files.

satpy.writers.cf_writer.**area2cf**(*dataarray*, *strict=False*)

satpy.writers.cf_writer.**area2gridmapping**(*dataarray*)

satpy.writers.cf_writer.**area2lonlat**(*dataarray*)
Convert an area to longitudes and latitudes.

satpy.writers.cf_writer.**create_grid_mapping**(*area*)
Create the grid mapping instance for *area*.

satpy.writers.cf_writer.**geos2cf**(*proj_dict*)
Return the cf grid mapping for the geos projection.

satpy.writers.cf_writer.**get_extra_ds**(*dataset*)
Get the extra datasets associated to *dataset*.

satpy.writers.cf_writer.**laea2cf**(*proj_dict*)
Return the cf grid mapping for the laea projection.

satpy.writers.cf_writer.**omerc2cf**(*proj_dict*)
Return the cf grid mapping for the omerc projection.

satpy.writers.geotiff module

GeoTIFF writer objects for creating GeoTIFF files from *Dataset* objects.

class satpy.writers.geotiff.**GeoTIFFWriter**(*floating_point=False*, *tags=None*, ***kwargs*)
Bases: *satpy.writers.ImageWriter*

Writer to save GeoTIFF images.

Basic example from Scene:

scn.save_datasets(writer='geotiff')

Un-enhanced float geotiff with NaN for fill values:

scn.save_datasets(writer='geotiff', floating_point=True, enhancement_config=False,
fill_value=np.nan)

GDAL_OPTIONS = ('tfw', 'rpb', 'rpctxt', 'interleave', 'tiled', 'blockxsize', 'blockysi

save_image(*img*, *filename=None*, *floating_point=False*, ***kwargs*)
Save the image to the given *filename* in geotiff format. *floating_point* allows the saving of 'L' mode images in floating point format if set to True.

satpy.writers.ninjotiff module

GeoTIFF writer objects for creating GeoTIFF files from *Dataset* objects.

class satpy.writers.ninjotiff.**NinjoTIFFWriter**(*floating_point=False*, *tags=None*, ***kwargs*)
Bases: *satpy.writers.ImageWriter*

GDAL_OPTIONS = ('tfw', 'rpb', 'rpctxt', 'interleave', 'tiled', 'blockxsize', 'blockysi

save_image(*img*, *filename=None*, ***kwargs*)
Save the image to the given *filename* in ninjotiff format.

satpy.writers.scmi module

satpy.writers.simple_image module

class satpy.writers.simple_image.**PillowWriter**(***kwargs*)
Bases: *satpy.writers.ImageWriter*

save_image(*img*, *filename=None*, ***kwargs*)

Module contents

Shared objects of the various writer classes.

For now, this includes enhancement configuration utilities.

class satpy.writers.**DecisionTree**(*decision_dicts*, *attrs*, ***kwargs*)
Bases: object

add_config_to_tree(**decision_dicts*)

any_key = None

find_match(***kwargs*)

class satpy.writers.**EnhancementDecisionTree**(**decision_dicts*, ***kwargs*)
Bases: *satpy.writers.DecisionTree*

add_config_to_tree(**decision_dict*)

find_match(***kwargs*)

class satpy.writers.**Enhancer**(*ppp_config_dir=None*, *enhancement_config_file=None*)
Bases: object

Helper class to get enhancement information for images.

add_sensor_enhancements(*sensor*)

apply(*img*, ***info*)

get_sensor_enhancement_config(*sensor*)

class satpy.writers.**ImageWriter**(*name=None*, *fill_value=None*, *file_pattern=None*, *enhancement_config=None*, *base_dir=None*, ***kwargs*)
Bases: *satpy.writers.Writer*

save_dataset(*dataset*, *filename=None*, *fill_value=None*, *overlay=None*, *decorate=None*, ***kwargs*)
Saves the *dataset* to a given *filename*.

save_image(*img*, *filename=None*, ***kwargs*)

class satpy.writers.**Writer**(*name=None*, *fill_value=None*, *file_pattern=None*, *base_dir=None*, ***kwargs*)
Bases: *satpy.plugin_base.Plugin*

Writer plugins. They must implement the *save_image* method. This is an abstract class to be inherited.

create_filename_parser(*base_dir*)

get_filename(***kwargs*)

save_dataset(*dataset*, *filename=None*, *fill_value=None*, ***kwargs*)
Saves the *dataset* to a given *filename*.

save_datasets(*datasets*, ***kwargs*)
Save all datasets to one or more files.

Subclasses can use this method to save all datasets to one single file or optimize the writing of individual datasets. By default this simply calls *save_dataset* for each dataset provided.

satpy.writers.**add_decorate**(*orig*, ***decorate*)
Decorate an image with text and/or logos/images.

This call adds text/logos in order as given in the input to keep the alignment features available in pydecorate.

An example of the decorate config:

```
decorate = {
    'decorate': [
        {'logo': {'logo_path': <path to a logo>, 'height': 143, 'bg': 'white',
↪'bg_opacity': 255}},
        {'text': {'txt': start_time_txt,
                  'align': {'top_bottom': 'bottom', 'left_right': 'right'},
                  'font': <path to ttf font>,
                  'font_size': 22,
                  'height': 30,
                  'bg': 'black',
                  'bg_opacity': 255,
                  'line': 'white'}}
    ]
}
```

Any numbers of text/logo in any order can be added to the decorate list, but the order of the list is kept as described above.

Note that a feature given in one element, eg. bg (which is the background color) will also apply on the next elements unless a new value is given.

align is a special keyword telling where in the image to start adding features, top_bottom is either top or bottom and left_right is either left or right.

satpy.writers.**add_logo**(*orig*, *dc*, *img*, *logo=None*)
Add logos or other images to an image using the pydecorate function add_logo All the features in pydecorate are available

See documentation of pydecorate

satpy.writers.**add_overlay**(*orig*, *area*, *coast_dir*, *color=(0, 0, 0)*, *width=0.5*, *resolution=None*, *level_coast=1*, *level_borders=1*)
Add coastline and political borders to image, using *color* (tuple of integers between 0 and 255). Warning: Loses the masks ! *resolution* is chosen automatically if None (default), otherwise it should be one of: +---+----------+ | 'f' | Full resolution | 0.04 km | | 'h' | High resolution | 0.2 km | | 'i' | Intermediate resolution | 1.0 km | | 'l' | Low resolution | 5.0 km | | 'c' | Crude resolution | 25 km | +---+----------+

satpy.writers.**add_text**(*orig*, *dc*, *img*, *text=None*)
Add text to an image using the pydecorate function add_text All the features in pydecorate are available

See documentation of pydecorate

satpy.writers.**get_enhanced_image**(*dataset*, *enhancer=None*, *fill_value=None*, *ppp_config_dir=None*, *enhancement_config_file=None*, *overlay=None*, *decorate=None*)

satpy.writers.**show**(*dataset*, ***kwargs*)
Display the dataset as an image.

satpy.writers.**to_image**(*dataset*, *copy=False*, ***kwargs*)

9.2 Submodules

9.3 satpy.config module

SatPy Configuration directory and file handling

satpy.config.**config_search_paths**(*filename*, **search_dirs*, ***kwargs*)

satpy.config.**get_config**(*filename*, **search_dirs*, ***kwargs*)
Blends the different configs, from package defaults to .

satpy.config.**get_config_path**(*filename*, **search_dirs*)
Get the appropriate path for a filename, in that order: filename, ., PPP_CONFIG_DIR, package's etc dir.

satpy.config.**get_environ_ancpath**(*default='.'*)

satpy.config.**get_environ_config_dir**(*default='/home/docs/checkouts/readthedocs.org/user_builds/satpy/envs/latest/lib/packages/satpy-0.8.1-py3.5.egg/satpy/etc'*)

satpy.config.**glob_config**(*pattern*, **search_dirs*)
Return glob results for all possible configuration locations.

Note: This method does not check the configuration "base" directory if the pattern includes a subdirectory.
This is done for performance since this is usually used to find *all* configs for a certain component.

satpy.config.**recursive_dict_update**(*d*, *u*)
Recursive dictionary update using

Copied from:

http://stackoverflow.com/questions/3232943/update-value-of-a-nested-dictionary-of-varying-depth

satpy.config.**runtime_import**(*object_path*)
Import at runtime

9.4 satpy.dataset module

Dataset objects.

class satpy.dataset.**Dataset**
Bases: object

Placeholder for the deprecated class.

class satpy.dataset.**DatasetID**
Bases: *satpy.dataset.DatasetID*

Identifier for all *Dataset* objects.

DatasetID is a namedtuple that holds identifying and classifying information about a Dataset. There are two identifying elements, `name` and `wavelength`. These can be used to generically refer to a Dataset. The other elements of a DatasetID are meant to further distinguish a Dataset from the possible variations it may have. For example multiple Datasets may be called by one `name` but may exist in multiple resolutions or with different calibrations such as "radiance" and "reflectance". If an element is *None* then it is considered not applicable.

A DatasetID can also be used in SatPy to query for a Dataset. This way a fully qualified DatasetID can be found even if some of the DatasetID elements are unknown. In this case a *None* signifies something that is unknown or not applicable to the requested Dataset.

Parameters

- **name** (*str*) – String identifier for the Dataset
- **wavelength** (*float, tuple*) – Single float wavelength when querying for a Dataset. Otherwise 3-element tuple of floats specifying the minimum, nominal, and maximum wavelength for a Dataset. *None* if not applicable.
- **resolution** (*int, float*) – Per data pixel/area resolution. If resolution varies across the Dataset then nadir view resolution is preferred. Usually this is in meters, but for lon/lat gridded data angle degrees may be used.
- **polarization** (*str*) – 'V' or 'H' polarizations of a microwave channel. *None* if not applicable.
- **calibration** (*str*) – String identifying the calibration level of the Dataset (ex. 'radiance', 'reflectance', etc). *None* if not applicable.
- **modifiers** (*tuple*) – Tuple of strings identifying what corrections or other modifications have been performed on this Dataset (ex. 'sunz_corrected', 'rayleigh_corrected', etc). *None* or empty tuple if not applicable.

classmethod from_dict(*d*, ***kwargs*)
Convert a dict to an ID.

static name_match(*a*, *b*)
Return if two string names are equal.

Parameters

- **a** (*str*) – DatasetID.name or other string
- **b** (*str*) – DatasetID.name or other string

to_dict(*trim=True*)
Convert the ID to a dict.

static wavelength_match(*a*, *b*)
Return if two wavelengths are equal.

Parameters

- **a** (*tuple or scalar*) – (min wl, nominal wl, max wl) or scalar wl
- **b** (*tuple or scalar*) – (min wl, nominal wl, max wl) or scalar wl

class satpy.dataset.**MetadataObject**(***attributes*)
Bases: object

A general metadata object.

id
Return the DatasetID of the object.

satpy.dataset.**combine_metadata**(**metadata_objects*)
Combine the metadata of two or more Datasets.

Parameters ***metadata_objects** – MetadataObject or dict objects to combine

Returns the combined metadata

satpy.dataset.**dataset_walker**(*datasets*)
Walk through *datasets* and their ancillary data.

Yields datasets and their parent.

satpy.dataset.**replace_anc**(*dataset*, *parent_dataset*)
Replace *dataset* the *parent_dataset*'s *ancillary_variables* field.

9.5 satpy.multiscene module

MultiScene object to blend satellite data.

class satpy.multiscene.**MultiScene**(*layers*)
Bases: object

blend(*blend_function=<function stack>*)
Blend the datasets into one scene.

load(**args*, ***kwargs*)
Load the required datasets from the multiple scenes.

resample(*destination*, ***kwargs*)
Resample the multiscene.

satpy.multiscene.**stack**(*datasets*)
First dataset at the bottom.

satpy.multiscene.**stack_time**(*datasets*)
Oldest time at the bottom.

9.6 satpy.node module

Nodes to build trees.

class satpy.node.**DependencyTree**(*readers*, *compositors*, *modifiers*)
Bases: *satpy.node.Node*

Structure to discover and store *Dataset* dependencies

Used primarily by the *Scene* object to organize dependency finding. Dependencies are stored used a series of *Node* objects which this class is a subclass of.

add_child(*parent*, *child*)

add_leaf(*ds_id*, *parent=None*)

copy()
Copy the this node tree

Note all references to readers are removed. This is meant to avoid tree copies accessing readers that would return incompatible (Area) data. Theoretically it should be possible for tree copies to request compositor or modifier information as long as they don't depend on any datasets not already existing in the dependency tree.

find_dependencies(*dataset_keys*, *calibration=None*, *polarization=None*, *resolution=None*)
Create the dependency tree.

Parameters

- **dataset_keys** (*iterable*) – Strings or DatasetIDs to find dependencies for
- **calibration** (*iterable or None*) –

Returns Root node of the dependency tree and a set of unknown datasets

Return type (*Node*, set)

get_compositor(*key*)

get_modifier(*comp_id*)

leaves(*nodes=None*, *unique=True*)
Get the leaves of the tree starting at this root.

Parameters

- **nodes** (*iterable*) – limit leaves for these node names
- **unique** – only include individual leaf nodes once

Returns list of leaf nodes

trunk(*nodes=None*, *unique=True*)
Get the trunk nodes of the tree starting at this root.

Parameters

- **nodes** (*iterable*) – limit trunk nodes to the names specified or the children of them that are also trunk nodes.
- **unique** – only include individual trunk nodes once

Returns list of trunk nodes

class satpy.node.**Node**(*name*, *data=None*)
Bases: object

A node object.

add_child(*obj*)
Add a child to the node.

copy()

display(*previous=0*, *include_data=False*)
Display the node.

flatten(*d=None*)
Flatten tree structure to a one level dictionary.

Parameters **d** (*dict, optional*) – output dictionary to update

Returns

Node.name -> Node. The returned dictionary includes the current Node and all its children.

Return type dict

is_leaf

leaves(*unique=True*)
Get the leaves of the tree starting at this root.

trunk(*unique=True*)
Get the trunk of the tree starting at this root.

9.7 satpy.plugin_base module

The *satpy.plugin_base* module defines the plugin API.

class satpy.plugin_base.**Plugin**(*ppp_config_dir=None*, *default_config_filename=None*, *config_files=None*, ***kwargs*)
Bases: object

The base plugin class. It is not to be used as is, it has to be inherited by other classes.

load_yaml_config(*conf*)

9.8 satpy.resample module

Shortcuts to resampling stuff.

class satpy.resample.**BaseResampler**(*source_geo_def*, *target_geo_def*)
Bases: object

The base resampler class. Abstract.

caches = <WeakValueDictionary>

compute(*data*, ***kwargs*)
Do the actual resampling

dump(*filename*)
Dump the projection info to *filename*.

get_hash(*source_geo_def=None*, *target_geo_def=None*, ***kwargs*)
Get hash for the current resample with the given *kwargs*.

static hash_area(*area*)
Get (and set) the hash for the *area*.

precompute(***kwargs*)
Do the precomputation.

This is an optional step if the subclass wants to implement more complex features like caching or can share some calculations between multiple datasets to be processed.

resample(*data*, *cache_dir=False*, *mask_area=True*, ***kwargs*)
Resample the *data*, saving the projection info on disk if *precompute* evaluates to True.

Parameters **mask_area** – Provide data mask to *precompute* method to mask invalid data values in geolocation.

class satpy.resample.**BilinearResampler**(*source_geo_def*, *target_geo_def*)
Bases: *satpy.resample.BaseResampler*

Resample using bilinear.

compute(*data*, *fill_value=None*, ***kwargs*)
Resample the given data using bilinear interpolation

precompute(*mask=None*, *radius_of_influence=50000*, *reduce_data=True*, *nprocs=1*, *segments=None*, *cache_dir=False*, ***kwargs*)
Create bilinear coefficients and store them for later use.

Note: The *mask* keyword should be provided if geolocation may be valid where data points are invalid. This defaults to the *mask* attribute of the *data* numpy masked array passed to the *resample* method.

class satpy.resample.**EWAResampler**(*source_geo_def*, *target_geo_def*, *swath_usage=0*, *grid_coverage=0*, ***kwargs*)
Bases: *satpy.resample.BaseResampler*

compute(*data*, *fill_value=0*, *weight_count=10000*, *weight_min=0.01*, *weight_distance_max=1.0*, *weight_sum_min=-1.0*, *maximum_weight_mode=False*, ***kwargs*)

precompute(*mask=None*, *cache_dir=False*, ***kwargs*)
Generate row and column arrays and store it for later use.

Note: The *mask* keyword should be provided if geolocation may be valid where data points are invalid. This defaults to the *mask* attribute of the *data* numpy masked array passed to the *resample* method.

class satpy.resample.**KDTreeResampler**(*source_geo_def*, *target_geo_def*)
Bases: *satpy.resample.BaseResampler*

Resample using nearest neighbour.

compute(*data*, *weight_funcs=None*, *fill_value=None*, *with_uncert=False*, ***kwargs*)

precompute(*mask=None*, *radius_of_influence=10000*, *epsilon=0*, *reduce_data=True*, *nprocs=1*, *segments=None*, *cache_dir=False*, ***kwargs*)
Create a KDTree structure and store it for later use.

Note: The *mask* keyword should be provided if geolocation may be valid where data points are invalid. This defaults to the *mask* attribute of the *data* numpy masked array passed to the *resample* method.

satpy.resample.**get_area_def**(*area_name*)
Get the definition of *area_name* from file.

The file is defined to use is to be placed in the $PPP_CONFIG_DIR directory, and its name is defined in satpy's configuration file.

satpy.resample.**get_area_file**()
Find area file(s) to use.

The files are to be named *areas.yaml* or *areas.def*.

satpy.resample.**get_frozen_area**(*to_freeze*, *ref*)
Freeze the *to_freeze* area according to *ref* if applicable, otherwise return *to_freeze* as an area definition instance.

satpy.resample.**mask_source_lonlats**(*source_def*, *mask*)
Mask source longitudes and latitudes to match data mask

satpy.resample.**resample**(*source_area*, *data*, *destination_area*, *resampler_class=<class 'satpy.resample.KDTreeResampler'>*, ***kwargs*)
Do the resampling.

satpy.resample.**resample_dataset**(*dataset*, *destination_area*, ***kwargs*)
Resample the current projectable and return the resampled one.

Parameters

- **destination_area** – The destination onto which to project the data, either a full blown area definition or a string corresponding to the name of the area as defined in the area file.
- ****kwargs** – The extra parameters to pass to the resampling functions.

Returns A resampled projectable, with updated .attrs["area"] field.

9.9 satpy.scene module

Scene objects to hold satellite data.

class satpy.scene.**Scene**(*filenames=None*, *reader=None*, *filter_parameters=None*, *reader_kwargs=None*, *ppp_config_dir='/home/docs/checkouts/readthedocs.org/user_builds/satpy/e packages/satpy-0.8.1-py3.5.egg/satpy/etc'*, *base_dir=None*, *sensor=None*, *start_time=None*, *end_time=None*, *area=None*)

Bases: *satpy.dataset.MetadataObject*

The Almighty Scene Class.

Example usage:

```
from satpy import Scene
from glob import glob

# create readers and open files
scn = Scene(filenames=glob('/path/to/files/*'), reader='viirs_sdr')

# load datasets from input files
scn.load(['I01', 'I02'])

# resample from satellite native geolocation to builtin 'eurol' Area
new_scn = scn.resample('eurol')

# save all resampled datasets to geotiff files in the current directory
new_scn.save_datasets()
```

all_composite_ids(*sensor_names=None*)
Get all composite IDs that are configured.

Returns generator of configured composite names

all_composite_names(*sensor_names=None*)

all_dataset_ids(*reader_name=None*, *composites=False*)
Get names of all datasets from loaded readers or *reader_name* if specified..

Returns list of all dataset names

all_dataset_names(*reader_name=None*, *composites=False*)

all_modifier_names()

available_composite_ids(*available_datasets=None*)
Get names of compositors that can be generated from the available datasets.

Returns generator of available compositor's names

available_composite_names(*available_datasets=None*)

available_dataset_ids(*reader_name=None*, *composites=False*)
Get names of available datasets, globally or just for *reader_name* if specified, that can be loaded.

Available dataset names are determined by what each individual reader can load. This is normally determined by what files are needed to load a dataset and what files have been provided to the scene/reader.

Returns list of available dataset names

available_dataset_names(*reader_name=None*, *composites=False*)
Get the list of the names of the available datasets.

compute(*nodes=None*)
Compute all the composites contained in *requirements*.

create_reader_instances(*filenames=None*, *reader=None*, *reader_kwargs=None*)
Find readers and return their instances.

end_time
Return the end time of the file.

get_writer(*writer='geotiff'*, ***kwargs*)
Get the writer instance.

get_writer_by_ext(*extension*, ***kwargs*)
Find the writer matching the *extension*.

images()
Generate images for all the datasets from the scene.

iter_by_area()
Generate datasets grouped by Area.

Returns generator of (area_obj, list of dataset objects)

keys(***kwargs*)

load(*wishlist*, *calibration=None*, *resolution=None*, *polarization=None*, *compute=True*, *unload=True*, ***kwargs*)
Read, compute and unload.

load_writer_config(*config_files*, ***kwargs*)
Load the writer config for *config_files*.

missing_datasets
DatasetIDs that have not been loaded.

read(*nodes=None*, ***kwargs*)
Load datasets from the necessary reader.

Parameters

- **nodes** (*iterable*) – DependencyTree Node objects
- ****kwargs** – Keyword arguments to pass to the reader's *load* method.

Returns DatasetDict of loaded datasets

read_composites(*compositor_nodes*)
Read (generate) composites.

read_datasets(*dataset_nodes*, ***kwargs*)
Read the given datasets from file.

resample(*destination*, *datasets=None*, *compute=True*, *unload=True*, ***resample_kwargs*)
Resample the datasets and return a new scene.

save_dataset(*dataset_id*, *filename=None*, *writer=None*, *overlay=None*, ***kwargs*)
Save the *dataset_id* to file using *writer* (default: geotiff).

save_datasets(*writer='geotiff'*, *datasets=None*, ***kwargs*)
Save all the datasets present in a scene to disk using *writer*.

show(*dataset_id*, *overlay=None*)
Show the *dataset* on screen as an image.

start_time
Return the start time of the file.

unload(*keepables=None*)
Unload all unneeded datasets.

Datasets are considered unneeded if they weren't directly requested or added to the Scene by the user or they are no longer needed to compute composites that have yet to be computed.

Parameters **keepables** (*iterable*) – DatasetIDs to keep whether they are needed or not.

9.10 satpy.utils module

Module defining various utilities.

class satpy.utils.**NullHandler**(*level=0*)
Bases: logging.Handler

Empty handler.

emit(*record*)
Record a message.

class satpy.utils.**OrderedConfigParser**(**args*, ***kwargs*)
Bases: object

Intercepts read and stores ordered section names. Cannot use inheritance and super as ConfigParser use old style classes.

read(*filename*)
Reads config file

sections()
Get sections from config file

satpy.utils.**angle2xyz**(*azi*, *zen*)
Convert azimuth and zenith to cartesian.

satpy.utils.**atmospheric_path_length_correction**(*data*, *cos_zen*, *limit=88.0*)
Perform Sun zenith angle correction.

This function uses the correction method proposed by Li and Shibata (2006): https://doi.org/10.1175/JAS3682.1

The correction is limited to *limit* degrees (default: 88.0 degrees). For larger zenith angles, the correction is the same as at the *limit*. Both *data* and *cos_zen* are given as 2-dimensional Numpy arrays or Numpy MaskedArrays, and they should have equal shapes.

satpy.utils.**debug_on**()
Turn debugging logging on.

satpy.utils.**ensure_dir**(*filename*)
Checks if the dir of f exists, otherwise create it.

satpy.utils.**get_logger**(*name*)
Return logger with null handle

satpy.utils.**logging_off**()
Turn logging off.

satpy.utils.**logging_on**(*level=30*)
Turn logging on.

satpy.utils.**lonlat2xyz**(*lon*, *lat*)
Convert lon lat to cartesian.

satpy.utils.**proj_units_to_meters**(*proj_str*)
Convert projection units from kilometers to meters.

satpy.utils.**sunzen_corr_cos**(*data*, *cos_zen*, *limit=88.0*)
Perform Sun zenith angle correction.

The correction is based on the provided cosine of the zenith angle (*cos_zen*). The correction is limited to *limit* degrees (default: 88.0 degrees). For larger zenith angles, the correction is the same as at the *limit*. Both *data* and *cos_zen* are given as 2-dimensional Numpy arrays or Numpy MaskedArrays, and they should have equal shapes.

satpy.utils.**xyz2angle**(*x*, *y*, *z*)
Convert cartesian to azimuth and zenith.

satpy.utils.**xyz2lonlat**(*x*, *y*, *z*)
Convert cartesian to lon lat.

9.11 satpy.version module

Version file.

9.12 Module contents

SatPy Package initializer.

Table 9.1: SatPy Readers

Description	Reader name	Status
MSG (Meteosat 8 to 11) Seviri data in HRIT format	*hrit_msg*	Nominal
MSG (Meteosat 8 to 11) SEVIRI data in native format	*native_msg*	No support for reading sub-section of the full disk. HRV data cannot be remapped.
Himawari 8 and 9 AHI data in HSD format	*ahi_hsd*	Nominal
Himawari 8 and 9 AHI data in HRIT format	*hrit_jma*	Nominal
GOES 16 imager data in netcdf format	*abi_l1b*	Nominal
GOES 11 to 15 imager data in HRIT format	*hrit_goes*	Nominal
Electro-L N2 MSU-GS data in HRIT format	*hrit_electrol*	Nominal
NOAA 15 to 19, Metop A to C AVHRR data in AAPP format	*avhrr_aapp_l1b*	Nominal
Metop A to C AVHRR in native level 1 format	*avhrr_eps_l1b*	Nominal
Tiros-N, NOAA 7 to 19 AVHRR data in GAC and LAC format	*gac_lac_l1b*	Nominal

Continued on next page

Table 9.1 – continued from previous page

Description	Reader name	Status
NOAA 15 to 19 AVHRR data in raw HRPT format	*avhrr_hrpt*	Nominal
GCOM-W1 AMSR2 data in HDF5 format	*amsr2_l1b*	Nominal
MTG FCI data in netcdf format	*fci_fdhsi*	Nominal
Callipso Caliop data in EOS-hdf4 format	*hdf4_caliopv3*	Nominal
Terra and Aqua MODIS data in EOS-hdf4 level-1 format as produced by IMAPP and IPOPP or downloaded from LAADS	*hdfeos_l1b*	Nominal
NWCSAF MSG 2016 products in netCDF4 format	*nc_nwcsaf_msg*	Nominal
NWCSAF PPS 2014 products in netCDF4 format	*nc_nwcsaf_pps*	Not yet support for remapped netCDF products. Only the standard swath based output is supported. CPP products not supported yet
Sentinel-1 A and B SAR-C data in SAFE format	*safe_sar_c*	Nominal
Sentinel-2 A and B MSI data in SAFE format	*safe_msi*	Nominal
Sentinel-3 A and B OLCI data in netCDF4 format	*nc_olci*	Nominal
Sentinel-3 A and B SLSTR data in netCDF4 format	*nc_slstr*	Nominal
OSISAF SST data in GHRSST (netcdf) format	*ghrsst_osisaf*	Nominal
NUCAPS EDR Retrieval in NetCDF4 format	*nucaps*	Nominal
NOAA Level 2 ACSPO SST data in netCDF4 format	*acspo*	Nominal
GEOstationary Cloud Algorithm Test-bed (GEOCAT)	*geocat*	Nominal
The Clouds from AVHRR Extended (CLAVR-x)	*clavrx*	Nominal
SNPP VIIRS data in hdf5 SDR format	*viirs_sdr*	Nominal
SNPP VIIRS data in netCDF4 L1B format	*viirs_sdr*	Nominal
SNPP VIIRS data in hdf5 Compact format	*viirs_compact*	Nominal
AAPP MAIA VIIRS and AVHRR products in hdf5 format	*maia*	Nominal

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

satpy, 70
satpy.composites, 29
satpy.composites.abi, 25
satpy.composites.ahi, 25
satpy.composites.crefl_utils, 26
satpy.composites.sar, 27
satpy.composites.viirs, 27
satpy.config, 61
satpy.dataset, 61
satpy.enhancements, 32
satpy.multiscene, 63
satpy.node, 63
satpy.plugin_base, 65
satpy.readers, 56
satpy.readers.aapp_l1b, 32
satpy.readers.abi_l1b, 33
satpy.readers.ahi_hsd, 33
satpy.readers.amsr2_l1b, 34
satpy.readers.clavrx, 34
satpy.readers.eps_l1b, 35
satpy.readers.fci_fdhsi, 36
satpy.readers.file_handlers, 36
satpy.readers.gac_lac_l1, 37
satpy.readers.generic_image, 37
satpy.readers.hdf4_caliopv3, 38
satpy.readers.hdf4_utils, 38
satpy.readers.hdf5_utils, 38
satpy.readers.hdfeos_l1b, 39
satpy.readers.hrit_base, 39
satpy.readers.hrit_electrol, 40
satpy.readers.hrit_goes, 41
satpy.readers.hrit_jma, 42
satpy.readers.hrit_msg, 42
satpy.readers.hrpt, 43
satpy.readers.iasi_l2, 44
satpy.readers.li_l2, 44
satpy.readers.maia, 45
satpy.readers.msg_base, 45
satpy.readers.native_msg, 45
satpy.readers.native_msg_hdr, 46
satpy.readers.nc_nwcsaf, 47
satpy.readers.nc_nwcsaf_msg, 47
satpy.readers.nc_olci, 48
satpy.readers.nc_slstr, 48
satpy.readers.omps_edr, 49
satpy.readers.safe_sar_c, 50
satpy.readers.scatsat1_l2b, 51
satpy.readers.utils, 51
satpy.readers.viirs_compact, 52
satpy.readers.viirs_sdr, 52
satpy.readers.xmlformat, 53
satpy.readers.yaml_reader, 53
satpy.resample, 65
satpy.scene, 67
satpy.utils, 69
satpy.version, 70
satpy.writers, 59
satpy.writers.cf_writer, 57
satpy.writers.geotiff, 58
satpy.writers.ninjotiff, 58
satpy.writers.simple_image, 59

Index

A

AbstractYAMLReader (class in satpy.readers.yaml_reader), 53
AdaptiveDNB (class in satpy.composites.viirs), 27
add_child() (satpy.node.DependencyTree method), 63
add_child() (satpy.node.Node method), 64
add_config_to_tree() (satpy.writers.DecisionTree method), 59
add_config_to_tree() (satpy.writers.EnhancementDecisionTree method), 59
add_decorate() (in module satpy.writers), 60
add_leaf() (satpy.node.DependencyTree method), 63
add_logo() (in module satpy.writers), 60
add_overlay() (in module satpy.writers), 60
add_sensor_enhancements() (satpy.writers.Enhancer method), 59
add_text() (in module satpy.writers), 60
adjust_scaling_factors() (satpy.readers.omps_edr.EDRFileHandler method), 49
adjust_scaling_factors() (satpy.readers.viirs_sdr.VIIRSSDRFileHandler method), 52
AHIHSDFileHandler (class in satpy.readers.ahi_hsd), 33
Airmass (class in satpy.composites), 29
all_composite_ids() (satpy.scene.Scene method), 67
all_composite_names() (satpy.scene.Scene method), 67
all_dataset_ids (satpy.readers.yaml_reader.AbstractYAMLReader attribute), 53
all_dataset_ids() (satpy.scene.Scene method), 67
all_dataset_names (satpy.readers.yaml_reader.AbstractYAMLReader attribute), 54
all_dataset_names() (satpy.scene.Scene method), 67
all_modifier_names() (satpy.scene.Scene method), 67
AMSR2L1BFileHandler (class in satpy.readers.amsr2_l1b), 34
angle2xyz() (in module satpy.utils), 69
angles() (satpy.readers.viirs_compact.VIIRSCompactFileHandler method), 52
any_key (satpy.writers.DecisionTree attribute), 59
apply() (satpy.writers.Enhancer method), 59
apply_modifier_info() (satpy.composites.CompositeBase method), 30
apply_scales() (satpy.readers.xmlformat.XMLFormat method), 53
area2cf() (in module satpy.writers.cf_writer), 58
area2gridmapping() (in module satpy.writers.cf_writer), 58
area2lonlat() (in module satpy.writers.cf_writer), 58
atmospheric_path_length_correction() (in module satpy.utils), 69
available_composite_ids() (satpy.scene.Scene method), 67
available_composite_names() (satpy.scene.Scene method), 67
available_dataset_ids (satpy.readers.yaml_reader.AbstractYAMLReader attribute), 54
available_dataset_ids (satpy.readers.yaml_reader.FileYAMLReader attribute), 54
available_dataset_ids() (satpy.readers.clavrx.CLAVRXFileHandler method), 34
available_dataset_ids() (satpy.scene.Scene method), 67
available_dataset_names (satpy.readers.yaml_reader.AbstractYAMLReader attribute), 54
available_dataset_names() (satpy.scene.Scene method), 67
AVHRRAAPPL1BFile (class in satpy.readers.aapp_l1b), 33

B

BaseFileHandler (class in satpy.readers.file_handlers), 36
BaseResampler (class in satpy.resample), 65
bfield() (in module satpy.readers.hrpt), 44
BilinearResampler (class in satpy.resample), 65
blend() (satpy.multiscene.MultiScene method), 63
build_colormap() (satpy.composites.ColormapCompositor static method), 30
BWCompositor (class in satpy.composites), 29

C

caches (satpy.resample.BaseResampler attribute), 65

calc_area_extent() (satpy.readers.fci_fdhsi.FCIFDHSIFileHandler method), 36
calibrate() (satpy.readers.aapp_l1b.AVHRRAAPPL1BFile method), 33
calibrate() (satpy.readers.ahi_hsd.AHIHSDFileHandler method), 34
calibrate() (satpy.readers.fci_fdhsi.FCIFDHSIFileHandler method), 36
calibrate() (satpy.readers.hrit_electrol.HRITGOMSFileHandler method), 41
calibrate() (satpy.readers.hrit_goes.HRITGOESFileHandler method), 41
calibrate() (satpy.readers.hrit_jma.HRITJMAFileHandler method), 42
calibrate() (satpy.readers.hrit_msg.HRITMSGFileHandler method), 43
calibrate() (satpy.readers.native_msg.NativeMSGFileHandler method), 45
calibrate_refl() (in module satpy.readers.hdfeos_l1b), 39
calibrate_tb() (in module satpy.readers.hdfeos_l1b), 39
CalibrationError, 34, 41, 45
celestial_events (satpy.readers.native_msg_hdr.L15DataHeaderRecord attribute), 46
CFWriter (class in satpy.writers.cf_writer), 57
chand() (in module satpy.composites.crefl_utils), 26
check_file_covers_area() (satpy.readers.yaml_reader.FileYAMLReader static method), 54
check_times() (in module satpy.composites), 32
cira_stretch() (in module satpy.enhancements), 32
CLAVRXFileHandler (class in satpy.readers.clavrx), 34
CLAVRXYAMLReader (class in satpy.readers.clavrx), 35
CloudCompositor (class in satpy.composites), 29
CO2Corrector (class in satpy.composites), 29
collect_metadata() (satpy.readers.hdf4_utils.HDF4FileHandler method), 38
collect_metadata() (satpy.readers.hdf5_utils.HDF5FileHandler method), 38
colorize() (in module satpy.enhancements), 32
ColorizeCompositor (class in satpy.composites), 29
ColormapCompositor (class in satpy.composites), 30
combine_info() (satpy.readers.file_handlers.BaseFileHandler method), 36
combine_metadata() (in module satpy.dataset), 62
CompositeBase (class in satpy.composites), 30
CompositorLoader (class in satpy.composites), 30
compute() (satpy.resample.BaseResampler method), 65
compute() (satpy.resample.BilinearResampler method), 65
compute() (satpy.resample.EWAResampler method), 66
compute() (satpy.resample.KDTreeResampler method), 66
compute() (satpy.scene.Scene method), 67
config_search_paths() (in module satpy.config), 61
configs_for_reader() (in module satpy.readers), 56
Convection (class in satpy.composites), 30
convert_to_radiance() (in module satpy.readers.msg_base), 45
convert_to_radiance() (satpy.readers.ahi_hsd.AHIHSDFileHandler method), 34
convert_to_radiance() (satpy.readers.hrit_msg.HRITMSGFileHandler method), 43
convert_to_radiance() (satpy.readers.native_msg.NativeMSGFileHandler method), 46
copy() (satpy.node.DependencyTree method), 63
copy() (satpy.node.Node method), 64
coszen (satpy.composites.SunZenithCorrectorBase attribute), 32
create_colormap() (in module satpy.enhancements), 32
create_filehandlers() (satpy.readers.clavrx.CLAVRXYAMLReader method), 35
create_filehandlers() (satpy.readers.yaml_reader.FileYAMLReader method), 55
create_filename_parser() (satpy.writers.Writer method), 59
create_grid_mapping() (in module satpy.writers.cf_writer), 58
create_reader_instances() (satpy.scene.Scene method), 68
create_xarray() (in module satpy.readers.aapp_l1b), 33
create_xarray() (in module satpy.readers.eps_l1b), 36
csalbr() (in module satpy.composites.crefl_utils), 26

D

da2cf() (satpy.writers.cf_writer.CFWriter static method), 57
data (satpy.readers.yaml_reader.Shuttle attribute), 55
Dataset (class in satpy.dataset), 61
dataset_walker() (in module satpy.dataset), 63
DatasetDict (class in satpy.readers), 56
DatasetID (class in satpy.dataset), 61
datasets (satpy.readers.nc_olci.NCOLCIAngles attribute), 48
datasets (satpy.readers.nc_slstr.NCSLSTRAngles attribute), 49
DayNightCompositor (class in satpy.composites), 30
debug_on() (in module satpy.utils), 69
dec10216() (in module satpy.readers.hrit_base), 40
dec10216() (in module satpy.readers.msg_base), 45
DecisionTree (class in satpy.writers), 59
DependencyTree (class in satpy.node), 63
dfilter_from_key() (satpy.readers.yaml_reader.AbstractYAMLReader static method), 54
dictify() (in module satpy.readers.safe_sar_c), 51
DifferenceCompositor (class in satpy.composites), 30
display() (satpy.node.Node method), 64
dtype() (satpy.readers.xmlformat.XMLFormat method), 53
dump() (satpy.resample.BaseResampler method), 65

Dust (class in satpy.composites), 30

E

EDREOSFileHandler (class in satpy.readers.omps_edr), 49
EDRFileHandler (class in satpy.readers.omps_edr), 49
EffectiveSolarPathLengthCorrector (class in satpy.composites), 31
emit() (satpy.utils.NullHandler method), 69
end_orbit_number (satpy.readers.omps_edr.EDRFileHandler attribute), 49
end_orbit_number (satpy.readers.viirs_sdr.VIIRSSDRFileHandler attribute), 52
end_time (satpy.readers.aapp_l1b.AVHRRAAPPL1BFile attribute), 33
end_time (satpy.readers.abi_l1b.NC_ABI_L1B attribute), 33
end_time (satpy.readers.ahi_hsd.AHIHSDFileHandler attribute), 34
end_time (satpy.readers.clavrx.CLAVRXFileHandler attribute), 34
end_time (satpy.readers.eps_l1b.EPSAVHRRFile attribute), 35
end_time (satpy.readers.fci_fdhsi.FCIFDHSIFileHandler attribute), 36
end_time (satpy.readers.file_handlers.BaseFileHandler attribute), 37
end_time (satpy.readers.gac_lac_l1.GACLACFile attribute), 37
end_time (satpy.readers.generic_image.GenericImageFileHandler attribute), 37
end_time (satpy.readers.hdf4_caliopv3.HDF4BandReader attribute), 38
end_time (satpy.readers.hdfeos_l1b.HDFEOSFileReader attribute), 39
end_time (satpy.readers.hrit_base.HRITFileHandler attribute), 39
end_time (satpy.readers.hrit_msg.HRITMSGFileHandler attribute), 43
end_time (satpy.readers.hrpt.HRPTFile attribute), 43
end_time (satpy.readers.iasi_l2.IASIL2HDF5 attribute), 44
end_time (satpy.readers.li_l2.LIFileHandler attribute), 44
end_time (satpy.readers.maia.MAIAFileHandler attribute), 45
end_time (satpy.readers.native_msg.NativeMSGFileHandler attribute), 46
end_time (satpy.readers.nc_nwcsaf.NcNWCSAF attribute), 47
end_time (satpy.readers.nc_nwcsaf_msg.NcNWCSAFMSG attribute), 47
end_time (satpy.readers.nc_olci.NCOLCIAngles attribute), 48
end_time (satpy.readers.nc_olci.NCOLCIBase attribute), 48
end_time (satpy.readers.nc_slstr.NCSLSTR1B attribute), 48
end_time (satpy.readers.nc_slstr.NCSLSTRAngles attribute), 49
end_time (satpy.readers.nc_slstr.NCSLSTRGeo attribute), 49
end_time (satpy.readers.safe_sar_c.SAFEGRD attribute), 50
end_time (satpy.readers.safe_sar_c.SAFEXML attribute), 50
end_time (satpy.readers.viirs_compact.VIIRSCompactFileHandler attribute), 52
end_time (satpy.readers.viirs_sdr.VIIRSSDRFileHandler attribute), 52
end_time (satpy.readers.yaml_reader.AbstractYAMLReader attribute), 54
end_time (satpy.readers.yaml_reader.FileYAMLReader attribute), 55
end_time (satpy.scene.Scene attribute), 68
enhance2dataset() (in module satpy.composites), 32
EnhancementDecisionTree (class in satpy.writers), 59
Enhancer (class in satpy.writers), 59
ensure_dir() (in module satpy.utils), 69
EPSAVHRRFile (class in satpy.readers.eps_l1b), 35
erads2bt() (in module satpy.readers.msg_base), 45
ERFDNB (class in satpy.composites.viirs), 28
EWAResampler (class in satpy.resample), 66
expand_array() (in module satpy.readers.viirs_compact), 52
Expander (class in satpy.composites.ahi), 25

F

FCIFDHSIFileHandler (class in satpy.readers.fci_fdhsi), 36
filename_items_for_filetype() (satpy.readers.yaml_reader.FileYAMLReader static method), 55
FileYAMLReader (class in satpy.readers.yaml_reader), 54
filter_ds_ids() (satpy.readers.yaml_reader.AbstractYAMLReader method), 54
filter_fh_by_metadata() (satpy.readers.yaml_reader.FileYAMLReader method), 55
filter_filenames_by_info() (satpy.readers.yaml_reader.FileYAMLReader method), 55
filter_selected_filenames() (satpy.readers.yaml_reader.AbstractYAMLReader method), 54
filter_selected_filenames() (satpy.readers.yaml_reader.FileYAMLReader method), 55

find_coefficient_index() (in module satpy.composites.crefl_utils), 26
find_dependencies() (satpy.node.DependencyTree method), 63
find_files_and_readers() (in module satpy.readers), 56
find_match() (satpy.writers.DecisionTree method), 59
find_match() (satpy.writers.EnhancementDecisionTree method), 59
find_required_filehandlers() (satpy.readers.yaml_reader.FileYAMLReader method), 55
flatten() (satpy.node.Node method), 64
four_element_average() (in module satpy.composites.abi), 25
from_dict() (satpy.dataset.DatasetID class method), 62

G

GACLACFile (class in satpy.readers.gac_lac_l1), 37
gain_factor() (satpy.composites.viirs.NCCZinke static method), 28
gamma() (in module satpy.enhancements), 32
GDAL_OPTIONS (satpy.writers.geotiff.GeoTIFFWriter attribute), 58
GDAL_OPTIONS (satpy.writers.ninjotiff.NinjoTIFFWriter attribute), 58
GenericCompositor (class in satpy.composites), 31
GenericImageFileHandler (class in satpy.readers.generic_image), 37
geo_interpolate() (in module satpy.readers.hrpt), 44
geo_mask() (satpy.readers.ahi_hsd.AHIHSDFileHandler method), 34
geometric_processing (satpy.readers.native_msg_hdr.L15DataHeaderRecord attribute), 46
geos2cf() (in module satpy.writers.cf_writer), 58
GeoTIFFWriter (class in satpy.writers.geotiff), 58
get() (satpy.readers.hdf4_utils.HDF4FileHandler method), 38
get() (satpy.readers.hdf5_utils.HDF5FileHandler method), 38
get() (satpy.readers.native_msg_hdr.L15DataHeaderRecord method), 46
get() (satpy.readers.native_msg_hdr.L15MainProductHeaderRecord method), 47
get() (satpy.readers.native_msg_hdr.L15SecondaryProductHeaderRecord method), 47
get() (satpy.readers.native_msg_hdr.Msg15NativeHeaderRecord method), 47
get_angles() (satpy.readers.aapp_l1b.AVHRRAAPPL1BFile method), 33
get_area_def() (in module satpy.resample), 66
get_area_def() (satpy.readers.abi_l1b.NC_ABI_L1B method), 33
get_area_def() (satpy.readers.ahi_hsd.AHIHSDFileHandler method), 34
get_area_def() (satpy.readers.fci_fdhsi.FCIFDHSIFileHandler method), 36
get_area_def() (satpy.readers.file_handlers.BaseFileHandler method), 37
get_area_def() (satpy.readers.hrit_base.HRITFileHandler method), 39
get_area_def() (satpy.readers.hrit_electrol.HRITGOMSFileHandler method), 41
get_area_def() (satpy.readers.hrit_goes.HRITGOESFileHandler method), 41
get_area_def() (satpy.readers.hrit_jma.HRITJMAFileHandler method), 42
get_area_def() (satpy.readers.hrit_msg.HRITMSGFileHandler method), 43
get_area_def() (satpy.readers.li_l2.LIFileHandler method), 44
get_area_def() (satpy.readers.native_msg.NativeMSGFileHandler method), 46
get_area_def() (satpy.readers.nc_nwcsaf.NcNWCSAF method), 47
get_area_def() (satpy.readers.nc_nwcsaf_msg.NcNWCSAFMSG method), 48
get_area_extent() (satpy.readers.hrit_base.HRITFileHandler method), 39
get_area_extent() (satpy.readers.hrit_msg.HRITMSGFileHandler method), 43
get_area_file() (in module satpy.resample), 66
get_area_slices() (in module satpy.readers.utils), 51
get_atm_variables() (in module satpy.composites.crefl_utils), 26
get_best_choice() (satpy.readers.DatasetDict method), 56
get_bounding_box() (satpy.readers.file_handlers.BaseFileHandler method), 37
get_bounding_box() (satpy.readers.viirs_compact.VIIRSCompactFileHandl method), 52
get_bounding_box() (satpy.readers.viirs_sdr.VIIRSSDRFileHandler method), 52
get_calibration() (satpy.readers.safe_sar_c.SAFEXML method), 50
get_calibration_constant() (satpy.readers.safe_sar_c.SAFEXML method), 50
get_cds_time() (in module satpy.readers.msg_base), 45
get_coefficients() (in module satpy.composites.crefl_utils), 26
get_compositor() (satpy.composites.CompositorLoader method), 30
get_compositor() (satpy.node.DependencyTree method), 64
get_config() (in module satpy.config), 61
get_config_path() (in module satpy.config), 61
get_data_type() (satpy.readers.clavrx.CLAVRXFileHandler method), 34
get_dataset() (satpy.readers.aapp_l1b.AVHRRAAPPL1BFile

method), 33
get_dataset() (satpy.readers.abi_l1b.NC_ABI_L1B method), 33
get_dataset() (satpy.readers.ahi_hsd.AHIHSDFileHandler method), 34
get_dataset() (satpy.readers.amsr2_l1b.AMSR2L1BFileHandler method), 34
get_dataset() (satpy.readers.clavrx.CLAVRXFileHandler method), 35
get_dataset() (satpy.readers.eps_l1b.EPSAVHRRFile method), 35
get_dataset() (satpy.readers.fci_fdhsi.FCIFDHSIFileHandler method), 36
get_dataset() (satpy.readers.file_handlers.BaseFileHandler method), 37
get_dataset() (satpy.readers.gac_lac_l1.GACLACFile method), 37
get_dataset() (satpy.readers.generic_image.GenericImageFileHandler method), 37
get_dataset() (satpy.readers.hdf4_caliopv3.HDF4BandReader method), 38
get_dataset() (satpy.readers.hdfeos_l1b.HDFEOSBandReader method), 39
get_dataset() (satpy.readers.hdfeos_l1b.HDFEOSGeoReader method), 39
get_dataset() (satpy.readers.hrit_base.HRITFileHandler method), 40
get_dataset() (satpy.readers.hrit_electrol.HRITGOMSFileHandler method), 41
get_dataset() (satpy.readers.hrit_goes.HRITGOESFileHandler method), 41
get_dataset() (satpy.readers.hrit_jma.HRITJMAFileHandler method), 42
get_dataset() (satpy.readers.hrit_msg.HRITMSGFileHandler method), 43
get_dataset() (satpy.readers.hrpt.HRPTFile method), 43
get_dataset() (satpy.readers.iasi_l2.IASIL2HDF5 method), 44
get_dataset() (satpy.readers.li_l2.LIFileHandler method), 44
get_dataset() (satpy.readers.maia.MAIAFileHandler method), 45
get_dataset() (satpy.readers.native_msg.NativeMSGFileHandler method), 46
get_dataset() (satpy.readers.nc_nwcsaf.NcNWCSAF method), 47
get_dataset() (satpy.readers.nc_nwcsaf_msg.NcNWCSAFMSG method), 48
get_dataset() (satpy.readers.nc_olci.NCOLCI1B method), 48
get_dataset() (satpy.readers.nc_olci.NCOLCI2 method), 48
get_dataset() (satpy.readers.nc_olci.NCOLCIAngles method), 48
get_dataset() (satpy.readers.nc_olci.NCOLCIBase method), 48
get_dataset() (satpy.readers.nc_slstr.NCSLSTR1B method), 49
get_dataset() (satpy.readers.nc_slstr.NCSLSTRAngles method), 49
get_dataset() (satpy.readers.nc_slstr.NCSLSTRGeo method), 49
get_dataset() (satpy.readers.omps_edr.EDRFileHandler method), 49
get_dataset() (satpy.readers.safe_sar_c.SAFEGRD method), 50
get_dataset() (satpy.readers.safe_sar_c.SAFEXML method), 50
get_dataset() (satpy.readers.scatsat1_l2b.SCATSAT1L2BFileHandler method), 51
get_dataset() (satpy.readers.viirs_compact.VIIRSCompactFileHandler method), 52
get_dataset() (satpy.readers.viirs_sdr.VIIRSSDRFileHandler method), 52
get_dataset_key() (satpy.readers.yaml_reader.AbstractYAMLReader method), 54
get_ds_ids_by_id() (satpy.readers.yaml_reader.AbstractYAMLReader method), 54
get_ds_ids_by_name() (satpy.readers.yaml_reader.AbstractYAMLReader method), 54
get_ds_ids_by_wavelength() (satpy.readers.yaml_reader.AbstractYAMLReader method), 54
get_end_time() (satpy.readers.hdf4_caliopv3.HDF4BandReader method), 38
get_enhanced_image() (in module satpy.writers), 60
get_environ_ancpath() (in module satpy.config), 61
get_environ_config_dir() (in module satpy.config), 61
get_extra_ds() (in module satpy.writers.cf_writer), 58
get_file_units() (satpy.readers.viirs_sdr.VIIRSSDRFileHandler method), 52
get_filebase() (in module satpy.readers.yaml_reader), 55
get_filehandle() (satpy.readers.hdf4_caliopv3.HDF4BandReader method), 38
get_filename() (satpy.writers.Writer method), 59
get_frozen_area() (in module satpy.resample), 66
get_full_angles() (satpy.readers.eps_l1b.EPSAVHRRFile method), 35
get_full_lonlats() (satpy.readers.eps_l1b.EPSAVHRRFile method), 35
get_gcps() (satpy.readers.safe_sar_c.SAFEGRD method), 50
get_gdal_filehandle() (satpy.readers.safe_sar_c.SAFEGRD method), 50
get_geostationary_angle_extent() (in module satpy.readers.utils), 51
get_geostationary_bounding_box() (in module satpy.readers.utils), 51

get_hash() (satpy.resample.BaseResampler method), 65
get_height() (satpy.readers.hdfeos_l1b.HDFEOSBandReader method), 39
get_item() (satpy.readers.DatasetDict method), 56
get_key() (satpy.readers.DatasetDict method), 56
get_keys() (satpy.readers.DatasetDict method), 56
get_keys_by_datasetid() (satpy.readers.DatasetDict method), 56
get_logger() (in module satpy.utils), 69
get_lonlat() (satpy.readers.eps_l1b.EPSAVHRRFile method), 35
get_lonlats() (satpy.readers.ahi_hsd.AHIHSDFileHandler method), 34
get_lonlats() (satpy.readers.eps_l1b.EPSAVHRRFile method), 35
get_lonlats() (satpy.readers.hdf4_caliopv3.HDF4BandReader method), 38
get_lonlats() (satpy.readers.hrpt.HRPTFile method), 43
get_lonlats() (satpy.readers.safe_sar_c.SAFEGRD method), 50
get_metadata() (satpy.readers.amsr2_l1b.AMSR2L1BFileHandler method), 34
get_metadata() (satpy.readers.clavrx.CLAVRXFileHandler method), 35
get_metadata() (satpy.readers.omps_edr.EDRFileHandler method), 49
get_metadata() (satpy.readers.safe_sar_c.SAFEXML method), 50
get_modifier() (satpy.composites.CompositorLoader method), 30
get_modifier() (satpy.node.DependencyTree method), 64
get_nadir_resolution() (satpy.readers.clavrx.CLAVRXFileHandler method), 35
get_noise_correction() (satpy.readers.safe_sar_c.SAFEXML method), 51
get_platform() (satpy.readers.clavrx.CLAVRXFileHandler method), 35
get_platform() (satpy.readers.maia.MAIAFileHandler method), 45
get_rows_per_scan() (satpy.readers.clavrx.CLAVRXFileHandler method), 35
get_sata() (satpy.readers.hdfeos_l1b.HDFEOSBandReader method), 39
get_satz() (satpy.readers.hdfeos_l1b.HDFEOSBandReader method), 39
get_sds_variable() (satpy.readers.hdf4_caliopv3.HDF4BandReader method), 38
get_sensor() (satpy.readers.clavrx.CLAVRXFileHandler method), 35
get_sensor_enhancement_config() (satpy.writers.Enhancer method), 59
get_shape() (satpy.readers.abi_l1b.NC_ABI_L1B method), 33
get_shape() (satpy.readers.ahi_hsd.AHIHSDFileHandler method), 34
get_shape() (satpy.readers.amsr2_l1b.AMSR2L1BFileHandler method), 34
get_shape() (satpy.readers.clavrx.CLAVRXFileHandler method), 35
get_shape() (satpy.readers.file_handlers.BaseFileHandler method), 37
get_shape() (satpy.readers.hrit_base.HRITFileHandler method), 40
get_shape() (satpy.readers.li_l2.LIFileHandler method), 44
get_shape() (satpy.readers.omps_edr.EDRFileHandler method), 49
get_shape() (satpy.readers.viirs_sdr.VIIRSSDRFileHandler method), 53
get_sub_area() (in module satpy.readers.utils), 51
get_suna() (satpy.readers.hdfeos_l1b.HDFEOSBandReader method), 39
get_sunz() (satpy.readers.hdfeos_l1b.HDFEOSBandReader method), 39
get_telemetry() (satpy.readers.hrpt.HRPTFile method), 44
get_writer() (satpy.scene.Scene method), 68
get_writer_by_ext() (satpy.scene.Scene method), 68
get_xy_from_linecol() (satpy.readers.hrit_base.HRITFileHandler method), 40
get_xy_from_linecol() (satpy.readers.hrit_msg.HRITMSGFileHandler method), 43
glob_config() (in module satpy.config), 61
gp_cpu_address (satpy.readers.native_msg_hdr.GSDTRecords attribute), 46
gp_fac_env (satpy.readers.native_msg_hdr.GSDTRecords attribute), 46
gp_fac_id (satpy.readers.native_msg_hdr.GSDTRecords attribute), 46
gp_pk_header (satpy.readers.native_msg_hdr.GSDTRecords attribute), 46
gp_pk_sh1 (satpy.readers.native_msg_hdr.GSDTRecords attribute), 46
gp_sc_id (satpy.readers.native_msg_hdr.GSDTRecords attribute), 46
gp_su_id (satpy.readers.native_msg_hdr.GSDTRecords attribute), 46
gp_svce_type (satpy.readers.native_msg_hdr.GSDTRecords attribute), 46
GreenCorrector (class in satpy.composites.ahi), 26
GSDTRecords (class in satpy.readers.native_msg_hdr), 46

H

hash_area() (satpy.resample.BaseResampler static method), 65
HDF4BandReader (class in satpy.readers.hdf4_caliopv3), 38

HDF4FileHandler (class in satpy.readers.hdf4_utils), 38
HDF5FileHandler (class in satpy.readers.hdf5_utils), 38
HDFEOSBandReader (class in satpy.readers.hdfeos_l1b), 39
HDFEOSFileReader (class in satpy.readers.hdfeos_l1b), 39
HDFEOSGeoReader (class in satpy.readers.hdfeos_l1b), 39
histogram_equalization() (in module satpy.composites.viirs), 28
HistogramDNB (class in satpy.composites.viirs), 28
HRITFileHandler (class in satpy.readers.hrit_base), 39
HRITGOESFileHandler (class in satpy.readers.hrit_goes), 41
HRITGOESPrologueFileHandler (class in satpy.readers.hrit_goes), 41
HRITGOMSEpilogueFileHandler (class in satpy.readers.hrit_electrol), 40
HRITGOMSFileHandler (class in satpy.readers.hrit_electrol), 40
HRITGOMSPrologueFileHandler (class in satpy.readers.hrit_electrol), 41
HRITJMAFileHandler (class in satpy.readers.hrit_jma), 42
HRITMSGEpilogueFileHandler (class in satpy.readers.hrit_msg), 42
HRITMSGFileHandler (class in satpy.readers.hrit_msg), 42
HRITMSGPrologueFileHandler (class in satpy.readers.hrit_msg), 43
HRPTFile (class in satpy.readers.hrpt), 43

I

IASIL2HDF5 (class in satpy.readers.iasi_l2), 44
id (satpy.dataset.MetadataObject attribute), 62
image_acquisition (satpy.readers.native_msg_hdr.L15DataHeaderRecord attribute), 46
image_description (satpy.readers.native_msg_hdr.L15DataHeaderRecord attribute), 46
images() (satpy.scene.Scene method), 68
ImageWriter (class in satpy.writers), 59
impf_configuration (satpy.readers.native_msg_hdr.L15DataHeaderRecord attribute), 46
IncompatibleAreas, 31
IncompatibleTimes, 31
info (satpy.readers.yaml_reader.Shuttle attribute), 55
interpolate_slice() (in module satpy.readers.safe_sar_c), 51
interpolate_xarray() (in module satpy.readers.safe_sar_c), 51
interpolate_xarray_linear() (in module satpy.readers.safe_sar_c), 51
interpolate_xml_array() (satpy.readers.safe_sar_c.SAFEXML static method), 51
invert() (in module satpy.enhancements), 32
is_leaf (satpy.node.Node attribute), 64
iter_by_area() (satpy.scene.Scene method), 68

K

KDTreeResampler (class in satpy.resample), 66
keys() (satpy.readers.DatasetDict method), 56
keys() (satpy.readers.eps_l1b.EPSAVHRRFile method), 35
keys() (satpy.scene.Scene method), 68

L

l15_ph_data (satpy.readers.native_msg_hdr.L15PhData attribute), 47
l15_ph_data_identification (satpy.readers.native_msg_hdr.L15MainProductHeaderRecord attribute), 47
L15DataHeaderRecord (class in satpy.readers.native_msg_hdr), 46
L15MainProductHeaderRecord (class in satpy.readers.native_msg_hdr), 47
L15PhData (class in satpy.readers.native_msg_hdr), 47
L15SecondaryProductHeaderRecord (class in satpy.readers.native_msg_hdr), 47
laea2cf() (in module satpy.writers.cf_writer), 58
leaves() (satpy.node.DependencyTree method), 64
leaves() (satpy.node.Node method), 64
LIFileHandler (class in satpy.readers.li_l2), 44
listify_string() (in module satpy.readers.yaml_reader), 55
load() (satpy.multiscene.MultiScene method), 63
load() (satpy.readers.hdfeos_l1b.HDFEOSGeoReader method), 39
load() (satpy.readers.yaml_reader.AbstractYAMLReader method), 54
load() (satpy.readers.yaml_reader.FileYAMLReader method), 55
load() (satpy.scene.Scene method), 68
load_compositors() (satpy.composites.CompositorLoader method), 30
load_ds_ids_from_config() (satpy.readers.yaml_reader.AbstractYAMLReader method), 54
load_ds_ids_from_files() (satpy.readers.clavrx.CLAVRXYAMLReader method), 35
load_reader() (in module satpy.readers), 57
load_readers() (in module satpy.readers), 57
load_sensor_composites() (satpy.composites.CompositorLoader method), 30
load_writer_config() (satpy.scene.Scene method), 68
load_yaml_config() (satpy.plugin_base.Plugin method), 65

local_histogram_equalization() (in module satpy.composites.viirs), 29
logging_off() (in module satpy.utils), 69
logging_on() (in module satpy.utils), 69
lonlat2xyz() (in module satpy.utils), 69
lookup() (in module satpy.enhancements), 32

M

MAIAFileHandler (class in satpy.readers.maia), 45
make_day_night_masks() (in module satpy.composites.viirs), 29
make_gvar_float() (in module satpy.readers.hrit_goes), 42
make_sgs_time() (in module satpy.readers.hrit_goes), 42
make_time_cds_expanded() (in module satpy.readers.hrit_msg), 43
make_time_cds_short() (in module satpy.readers.hrit_base), 40
MalformedConfigError, 56
mask (satpy.readers.yaml_reader.Shuttle attribute), 55
mask_source_lonlats() (in module satpy.resample), 66
match_filenames() (in module satpy.readers.yaml_reader), 56
metadata_matches() (satpy.readers.yaml_reader.FileYAMLReader method), 55
MetadataObject (class in satpy.dataset), 62
missing_datasets (satpy.scene.Scene attribute), 68
modes (satpy.composites.GenericCompositor attribute), 31
Msg15NativeHeaderRecord (class in satpy.readers.native_msg_hdr), 47
MultiScene (class in satpy.multiscene), 63

N

nadir_resolution (satpy.readers.clavrx.CLAVRXFileHandler attribute), 35
name_match() (satpy.dataset.DatasetID static method), 62
NativeMSGFileHandler (class in satpy.readers.native_msg), 45
navigate() (satpy.readers.aapp_l1b.AVHRRAAPPL1BFile method), 33
navigate() (satpy.readers.viirs_compact.VIIRSCompactFileHandler method), 52
navigate_dnb() (in module satpy.readers.viirs_compact), 52
NC_ABI_L1B (class in satpy.readers.abi_l1b), 33
NCCZinke (class in satpy.composites.viirs), 28
NcNWCSAF (class in satpy.readers.nc_nwcsaf), 47
NcNWCSAFMSG (class in satpy.readers.nc_nwcsaf_msg), 47
NCOLCI1B (class in satpy.readers.nc_olci), 48
NCOLCI2 (class in satpy.readers.nc_olci), 48
NCOLCIAngles (class in satpy.readers.nc_olci), 48
NCOLCIBase (class in satpy.readers.nc_olci), 48
NCOLCICal (class in satpy.readers.nc_olci), 48
NCOLCIChannelBase (class in satpy.readers.nc_olci), 48
NCOLCIGeo (class in satpy.readers.nc_olci), 48
NCSLSTR1B (class in satpy.readers.nc_slstr), 48
NCSLSTRAngles (class in satpy.readers.nc_slstr), 49
NCSLSTRGeo (class in satpy.readers.nc_slstr), 49
new_filehandler_instances() (satpy.readers.yaml_reader.FileYAMLReader method), 55
new_filehandlers_for_filetype() (satpy.readers.yaml_reader.FileYAMLReader method), 55
NinjoTIFFWriter (class in satpy.writers.ninjotiff), 58
NIREmissivePartFromReflectance (class in satpy.composites), 31
NIRReflectance (class in satpy.composites), 31
Node (class in satpy.node), 64
np2str() (in module satpy.readers.utils), 51
NullHandler (class in satpy.utils), 69

O

omerc2cf() (in module satpy.writers.cf_writer), 58
OrderedConfigParser (class in satpy.utils), 69
overlay() (in module satpy.composites.sar), 27

P

PaletteCompositor (class in satpy.composites), 31
palettize() (in module satpy.enhancements), 32
parse_format() (in module satpy.readers.xmlformat), 53
parse_metadata_string() (satpy.readers.hdf4_caliopv3.HDF4BandReader static method), 38
PillowWriter (class in satpy.writers.simple_image), 59
platform_name (satpy.readers.eps_l1b.EPSAVHRRFile attribute), 35
platform_name (satpy.readers.omps_edr.EDRFileHandler attribute), 49
platform_name (satpy.readers.viirs_sdr.VIIRSSDRFileHandler attribute), 53
platforms (satpy.readers.clavrx.CLAVRXFileHandler attribute), 35
Plugin (class in satpy.plugin_base), 65
precompute() (satpy.resample.BaseResampler method), 65
precompute() (satpy.resample.BilinearResampler method), 65
precompute() (satpy.resample.EWAResampler method), 66
precompute() (satpy.resample.KDTreeResampler method), 66
process_array() (in module satpy.readers.xmlformat), 53
process_delimiter() (in module satpy.readers.xmlformat), 53
process_field() (in module satpy.readers.xmlformat), 53
process_prologue() (satpy.readers.hrit_electrol.HRITGOMSPrologueFileHa method), 41

process_prologue() (satpy.readers.hrit_goes.HRITGOESPrologueFileHandler method), 41
process_prologue() (satpy.readers.hrit_msg.HRITMSGPrologueFileHandler method), 43
proj_units_to_meters() (in module satpy.utils), 69
PSPAtmosphericalCorrection (class in satpy.composites), 31
PSPRayleighReflectance (class in satpy.composites), 31

R

radiance_to_bt() (in module satpy.readers.eps_l1b), 36
radiance_to_refl() (in module satpy.readers.eps_l1b), 36
radiometric_processing (satpy.readers.native_msg_hdr.L15DataHeaderRecord attribute), 46
RatioSharpenedRGB (class in satpy.composites.viirs), 28
read() (satpy.readers.aapp_l1b.AVHRRAAPPL1BFile method), 33
read() (satpy.readers.generic_image.GenericImageFileHandler method), 37
read() (satpy.readers.hrpt.HRPTFile method), 44
read() (satpy.readers.maia.MAIAFileHandler method), 45
read() (satpy.scene.Scene method), 68
read() (satpy.utils.OrderedConfigParser method), 69
read_band() (satpy.readers.ahi_hsd.AHIHSDFileHandler method), 34
read_band() (satpy.readers.hrit_base.HRITFileHandler method), 40
read_band() (satpy.readers.safe_sar_c.SAFEGRD method), 50
read_composites() (satpy.scene.Scene method), 68
read_dataset() (in module satpy.readers.iasi_l2), 44
read_dataset() (satpy.readers.viirs_compact.VIIRSCompactFileHandler method), 52
read_datasets() (satpy.scene.Scene method), 68
read_dnb() (in module satpy.readers.viirs_compact), 52
read_epilogue() (satpy.readers.hrit_electrol.HRITGOMSEpilogueFileHandler method), 40
read_epilogue() (satpy.readers.hrit_msg.HRITMSGEpilogueFileHandler method), 42
read_geo() (in module satpy.readers.iasi_l2), 44
read_geo() (satpy.readers.viirs_compact.VIIRSCompactFileHandler method), 52
read_mda() (satpy.readers.hdfeos_l1b.HDFEOSFileReader method), 39
read_prologue() (satpy.readers.hrit_electrol.HRITGOMSPrologueFileHandler method), 41
read_prologue() (satpy.readers.hrit_goes.HRITGOESPrologueFileHandler method), 41
read_prologue() (satpy.readers.hrit_msg.HRITMSGPrologueFileHandler method), 43
read_raw() (in module satpy.readers.eps_l1b), 36
read_reader_config() (in module satpy.readers), 57
read_xml_array() (satpy.readers.safe_sar_c.SAFEXML static method), 51
RealisticColors (class in satpy.composites), 31
recarray2dict() (in module satpy.readers.hrit_electrol), 41
recarray2dict() (in module satpy.readers.hrit_goes), 42
recarray2dict() (in module satpy.readers.hrit_jma), 42
recarray2dict() (in module satpy.readers.hrit_msg), 43
recursive_dict_update() (in module satpy.config), 61
Reducer2 (class in satpy.composites.ahi), 26
Reducer4 (class in satpy.composites.ahi), 26
Reducer8 (class in satpy.composites.ahi), 26
ReflectanceCorrector (class in satpy.composites.viirs), 28
remove_empties() (in module satpy.readers.nc_nwcsaf), 47
replace_anc() (in module satpy.dataset), 63
res (satpy.readers.hdfeos_l1b.HDFEOSBandReader attribute), 39
resample() (in module satpy.resample), 66
resample() (satpy.multiscene.MultiScene method), 63
resample() (satpy.resample.BaseResampler method), 65
resample() (satpy.scene.Scene method), 68
resample_dataset() (in module satpy.resample), 66
RGBCompositor (class in satpy.composites), 31
rows_per_scan (satpy.readers.clavrx.CLAVRXFileHandler attribute), 35
run_crefl() (in module satpy.composites.crefl_utils), 27
runtime_import() (in module satpy.config), 61

S

SAFEGRD (class in satpy.readers.safe_sar_c), 50
SAFEXML (class in satpy.readers.safe_sar_c), 50
SARIce (class in satpy.composites.sar), 27
satellite_status (satpy.readers.native_msg_hdr.L15DataHeaderRecord attribute), 47
satpy (module), 70
satpy.composites (module), 29
satpy.composites.abi (module), 25
satpy.composites.ahi (module), 25
satpy.composites.crefl_utils (module), 26
satpy.composites.sar (module), 27
satpy.composites.viirs (module), 27
satpy.config (module), 61
satpy.dataset (module), 61
satpy.enhancements (module), 32
satpy.multiscene (module), 63
satpy.node (module), 63
satpy.plugin_base (module), 65
satpy.readers (module), 56
satpy.readers.aapp_l1b (module), 32
satpy.readers.abi_l1b (module), 33
satpy.readers.ahi_hsd (module), 33
satpy.readers.amsr2_l1b (module), 34
satpy.readers.clavrx (module), 34
satpy.readers.eps_l1b (module), 35
satpy.readers.fci_fdhsi (module), 36
satpy.readers.file_handlers (module), 36

satpy.readers.gac_lac_l1 (module), 37
satpy.readers.generic_image (module), 37
satpy.readers.hdf4_caliopv3 (module), 38
satpy.readers.hdf4_utils (module), 38
satpy.readers.hdf5_utils (module), 38
satpy.readers.hdfeos_l1b (module), 39
satpy.readers.hrit_base (module), 39
satpy.readers.hrit_electrol (module), 40
satpy.readers.hrit_goes (module), 41
satpy.readers.hrit_jma (module), 42
satpy.readers.hrit_msg (module), 42
satpy.readers.hrpt (module), 43
satpy.readers.iasi_l2 (module), 44
satpy.readers.li_l2 (module), 44
satpy.readers.maia (module), 45
satpy.readers.msg_base (module), 45
satpy.readers.native_msg (module), 45
satpy.readers.native_msg_hdr (module), 46
satpy.readers.nc_nwcsaf (module), 47
satpy.readers.nc_nwcsaf_msg (module), 47
satpy.readers.nc_olci (module), 48
satpy.readers.nc_slstr (module), 48
satpy.readers.omps_edr (module), 49
satpy.readers.safe_sar_c (module), 50
satpy.readers.scatsat1_l2b (module), 51
satpy.readers.utils (module), 51
satpy.readers.viirs_compact (module), 52
satpy.readers.viirs_sdr (module), 52
satpy.readers.xmlformat (module), 53
satpy.readers.yaml_reader (module), 53
satpy.resample (module), 65
satpy.scene (module), 67
satpy.utils (module), 69
satpy.version (module), 70
satpy.writers (module), 59
satpy.writers.cf_writer (module), 57
satpy.writers.geotiff (module), 58
satpy.writers.ninjotiff (module), 58
satpy.writers.simple_image (module), 59
save_dataset() (satpy.scene.Scene method), 68
save_dataset() (satpy.writers.cf_writer.CFWriter method), 58
save_dataset() (satpy.writers.ImageWriter method), 59
save_dataset() (satpy.writers.Writer method), 60
save_datasets() (satpy.scene.Scene method), 68
save_datasets() (satpy.writers.cf_writer.CFWriter method), 58
save_datasets() (satpy.writers.Writer method), 60
save_image() (satpy.writers.geotiff.GeoTIFFWriter method), 58
save_image() (satpy.writers.ImageWriter method), 59
save_image() (satpy.writers.ninjotiff.NinjoTIFFWriter method), 59
save_image() (satpy.writers.simple_image.PillowWriter method), 59
scale_swath_data() (satpy.readers.viirs_sdr.VIIRSSDRFileHandler method), 53
SCATSAT1L2BFileHandler (class in satpy.readers.scatsat1_l2b), 51
Scene (class in satpy.scene), 67
sections() (satpy.utils.OrderedConfigParser method), 69
select_files_from_directory() (satpy.readers.yaml_reader.AbstractYAMLReader method), 54
select_files_from_pathnames() (satpy.readers.yaml_reader.AbstractYAMLReader method), 54
sensor_name (satpy.readers.eps_l1b.EPSAVHRRFile attribute), 35
sensor_name (satpy.readers.omps_edr.EDRFileHandler attribute), 49
sensor_name (satpy.readers.viirs_sdr.VIIRSSDRFileHandler attribute), 53
sensors (satpy.readers.clavrx.CLAVRXFileHandler attribute), 35
sensors (satpy.readers.eps_l1b.EPSAVHRRFile attribute), 36
shape() (satpy.readers.aapp_l1b.AVHRRAAPPL1BFile method), 33
show() (in module satpy.readers.ahi_hsd), 34
show() (in module satpy.readers.hrit_goes), 42
show() (in module satpy.readers.hrit_jma), 42
show() (in module satpy.readers.hrit_msg), 43
show() (in module satpy.writers), 61
show() (satpy.scene.Scene method), 68
Shuttle (class in satpy.readers.yaml_reader), 55
simulated_green() (in module satpy.composites.abi), 25
SnowAge (class in satpy.composites.viirs), 28
sorted_filetype_items() (satpy.readers.yaml_reader.FileYAMLReader method), 55
spacecrafts (satpy.readers.eps_l1b.EPSAVHRRFile attribute), 36
srads2bt() (in module satpy.readers.msg_base), 45
stack() (in module satpy.multiscene), 63
stack_time() (in module satpy.multiscene), 63
start_orbit_number (satpy.readers.omps_edr.EDRFileHandler attribute), 49
start_orbit_number (satpy.readers.viirs_sdr.VIIRSSDRFileHandler attribute), 53
start_time (satpy.readers.aapp_l1b.AVHRRAAPPL1BFile attribute), 33
start_time (satpy.readers.abi_l1b.NC_ABI_L1B attribute), 33
start_time (satpy.readers.ahi_hsd.AHIHSDFileHandler attribute), 34
start_time (satpy.readers.clavrx.CLAVRXFileHandler attribute), 35

start_time (satpy.readers.eps_l1b.EPSAVHRRFile attribute), 36
start_time (satpy.readers.fci_fdhsi.FCIFDHSIFileHandler attribute), 36
start_time (satpy.readers.file_handlers.BaseFileHandler attribute), 37
start_time (satpy.readers.gac_lac_l1.GACLACFile attribute), 37
start_time (satpy.readers.generic_image.GenericImageFileHandler attribute), 37
start_time (satpy.readers.hdf4_caliopv3.HDF4BandReader attribute), 38
start_time (satpy.readers.hdfeos_l1b.HDFEOSFileReader attribute), 39
start_time (satpy.readers.hrit_base.HRITFileHandler attribute), 40
start_time (satpy.readers.hrit_msg.HRITMSGFileHandler attribute), 43
start_time (satpy.readers.hrpt.HRPTFile attribute), 44
start_time (satpy.readers.iasi_l2.IASIL2HDF5 attribute), 44
start_time (satpy.readers.li_l2.LIFileHandler attribute), 44
start_time (satpy.readers.maia.MAIAFileHandler attribute), 45
start_time (satpy.readers.native_msg.NativeMSGFileHandler attribute), 46
start_time (satpy.readers.nc_nwcsaf.NcNWCSAF attribute), 47
start_time (satpy.readers.nc_nwcsaf_msg.NcNWCSAFMSG attribute), 48
start_time (satpy.readers.nc_olci.NCOLCIAngles attribute), 48
start_time (satpy.readers.nc_olci.NCOLCIBase attribute), 48
start_time (satpy.readers.nc_slstr.NCSLSTR1B attribute), 49
start_time (satpy.readers.nc_slstr.NCSLSTRAngles attribute), 49
start_time (satpy.readers.nc_slstr.NCSLSTRGeo attribute), 49
start_time (satpy.readers.safe_sar_c.SAFEGRD attribute), 50
start_time (satpy.readers.safe_sar_c.SAFEXML attribute), 51
start_time (satpy.readers.viirs_compact.VIIRSCompactFileHandler attribute), 52
start_time (satpy.readers.viirs_sdr.VIIRSSDRFileHandler attribute), 53
start_time (satpy.readers.yaml_reader.AbstractYAMLReader attribute), 54
start_time (satpy.readers.yaml_reader.FileYAMLReader attribute), 55
start_time (satpy.scene.Scene attribute), 68
stretch() (in module satpy.enhancements), 32
sub_arrays() (in module satpy.composites), 32
sunzen_corr_cos() (in module satpy.utils), 70
SunZenithCorrector (class in satpy.composites), 31
SunZenithCorrectorBase (class in satpy.composites), 32
supports_sensor() (satpy.readers.yaml_reader.AbstractYAMLReader method), 54

T

three_d_effect() (in module satpy.enhancements), 32
time_cds (satpy.readers.native_msg_hdr.GSDTRecords attribute), 46
time_cds_expanded (satpy.readers.native_msg_hdr.GSDTRecords attribute), 46
time_cds_short (satpy.readers.native_msg_hdr.GSDTRecords attribute), 46
time_matches() (satpy.readers.yaml_reader.FileYAMLReader method), 55
time_seconds() (in module satpy.readers.hrpt), 44
tl15() (in module satpy.readers.msg_base), 45
to_dict() (satpy.dataset.DatasetID method), 62
to_dtype() (in module satpy.readers.xmlformat), 53
to_image() (in module satpy.writers), 61
to_scaled_dtype() (in module satpy.readers.xmlformat), 53
to_scales() (in module satpy.readers.xmlformat), 53
TrueColor (class in satpy.composites.abi), 25
TrueColor2km (class in satpy.composites.abi), 25
trunk() (satpy.node.DependencyTree method), 64
trunk() (satpy.node.Node method), 65

U

unload() (satpy.scene.Scene method), 68

V

view (satpy.readers.nc_slstr.NCSLSTRAngles attribute), 49
VIIRSCompactFileHandler (class in satpy.readers.viirs_compact), 52
VIIRSFog (class in satpy.composites.viirs), 28
VIIRSSDRFileHandler (class in satpy.readers.viirs_sdr), 52
VIIRSSDRReader (class in satpy.readers.viirs_sdr), 53
vis_calibrate() (in module satpy.readers.msg_base), 45

W

wavelength_match() (satpy.dataset.DatasetID static method), 62
Writer (class in satpy.writers), 59

X

XMLFormat (class in satpy.readers.xmlformat), 53
xyz2angle() (in module satpy.utils), 70
xyz2lonlat() (in module satpy.utils), 70