mpop documentation

Release v1.1.0

SMHI

December 17, 2015

Contents

The Meteorological Post-Processing package is a python library for generating RGB products for meteorological remote sensing. As such it can create RGB composites directly from satellite instrument channels, or take advantage of precomputed PGEs.

Get to the project page, with source and downloads.

It is designed to be easily extendable to support any meteorological satellite by the creation of plugins. In the base distribution, we provide support for Meteosat-7, -8, -9, -10, Himawari-6 (MTSAT-1R), Himawari-7 (MTSAT-2), GOES-11, GOES-12, GOES-13 through the use of mipp, and NOAA-15, -16, -17, -18, -19, Metop-A and -B through the use of AAPP.

Reprojection of data is also available through the use of pyresample.

CHAPTER 1

Quickstart

1.1 Loading data

Changed in version 2.0.0-alpha.1: New syntax

To work with weather satellite data, one has to create an instance of the `Scene` class. In order for mpop to get access to the data, either the current wording directory has to be set to the directory containing the data files, or the *base_dir* keyword argument has to be provided on scene creation:

```
>>> import os
>>> os.chdir("/home/a001673/data/satellite/Meteosat-10/seviri/lvl1.5/2015/04/20/HRIT")
>>> from mpop import Scene
>>> from datetime import datetime
>>> time_slot = datetime(2015, 4, 20, 10, 0)
>>> global_scene = Scene(platform_name="Meteosat-10", sensor="seviri", start_time=datetime(2015, 4, 2
```

or:

```
>>> from mpop.scene import Scene
>>> from datetime import datetime
>>> time_slot = datetime(2015, 4, 20, 10, 0)
>>> global_scene = Scene(platform_name="Meteosat-10", sensor="seviri", start_time=datetime(2015, 4, 2
>>>
```

For some platforms, it might be necessary to also specify an *end_time*:

```
>>> Scene(platform_name="SNPP", sensor="viirs", start_time=datetime(2015, 3, 11, 11, 20), end_time=da
```

Loading weather satellite data with mpop is as simple as calling 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)
```

As you can see, this loads the visible and IR channels provided as argument to the `load()` method as a list of wavelengths in micrometers. Another way to load the channels is to provide the names instead:

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

To have a look at the available bands you should be able to load with your *Scene* object, you can call the `available_datasets()` method:

```
>>> global_scene.available_datasets()

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

To access the loaded data:

```
>>> print global_scene[0.6]
```

or:

```
>>> print global_scene["VIS006"]
```

To visualize it:

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

To combine them:

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

1.2 Generating composites

The easiest way to generate composites is to *load* them:

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

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

```
>>> global_scene.available_composites()

[u'overview_sun',
 u'airmass',
 u'natural',
 u'night_fog',
 u'overview',
 u'green_snow',
 u'dust',
 u'fog',
 u'natural_sun',
 u'cloudtop',
 u'convection',
 u'ash']
```

To save a composite to disk:

```
>>> global_scene.save_dataset('overview', 'my_nice_overview.png')
```

One can also specify which writer to use for filenames with non-standard extensions

```
>>> global_scene.save_dataset('overview', 'my_nice_overview.stupidextension', writer='geotiff')
```

1.3 Resampling

Until now, we have used the channels directly as provided by the satellite, that is in satellite projection. Generating composites thus produces views in satellite projection, *i.e.* as viewed by the satellite.

Most often however, we will want to resample the data onto a specific area so that only the area of interest is depicted in the RGB composites.

Here is how we do that:

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

Now we have resampled channel data and composites onto the "eurol" area in the *local_scene* variable and we can operate as before to display and save RGB composites:

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

The image is automatically saved here in GeoTiff format.

1.4 Making custom composites

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

```
>>> from mpop.composites import RGBCompositor
>>> compositor = RGBCompositor("myoverview", "bla", "")
>>> composite = compositor([local_scene[0.6],
...                         local_scene[0.8],
...                         local_scene[10.8]])
>>> from mpop.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.

CHAPTER 2

mpop package

2.1 Subpackages

2.1.1 mpop.composites package

Submodules

mpop.composites.viirs module

Module contents

Base classes for composite objects.

class mpop.composites.**Airmass**(*name*, *prerequisites=[]*, *optional_prerequisites=[]*, *metadata_requirements=[]*, ***kwargs*)
Bases: *mpop.composites.RGBCompositor*

__call__(*projectables*, **args*, ***kwargs*)
Make an airmass RGB image composite.

Channels	Temp	Gamma
WV6.2 - WV7.3	-25 to 0 K	gamma 1
IR9.7 - IR10.8	-40 to 5 K	gamma 1
WV6.2	243 to 208 K	gamma 1

class mpop.composites.**CompositeBase**(*name*, *prerequisites=[]*, *optional_prerequisites=[]*, *metadata_requirements=[]*, ***kwargs*)
Bases: *mpop.projectable.InfoObject*

class mpop.composites.**CompositeReader**(*composite_names*, *sensor_names=None*)
Bases: object

Read composites using the configuration files on disk.

class mpop.composites.**Convection**(*name*, *prerequisites=[]*, *optional_prerequisites=[]*, *metadata_requirements=[]*, ***kwargs*)
Bases: *mpop.composites.RGBCompositor*

__call__(*projectables*, **args*, ***kwargs*)
Make a Severe Convection RGB image composite.

Channels	Span	Gamma
WV6.2 - WV7.3	-30 to 0 K	gamma 1
IR3.9 - IR10.8	0 to 55 K	gamma 1
IR1.6 - VIS0.6	-70 to 20 %	gamma 1

class mpop.composites.**Dust**(*name*, *prerequisites=[]*, *optional_prerequisites=[]*, *metadata_requirements=[]*, ***kwargs*)

Bases: *mpop.composites.RGBCompositor*

__call__(*projectables*, **args*, ***kwargs*)

Make a Dust RGB image composite.

Channels	Temp	Gamma
IR12.0 - IR10.8	-4 to 2 K	gamma 1
IR10.8 - IR8.7	0 to 15 K	gamma 2.5
IR10.8	261 to 289 K	gamma 1

exception mpop.composites.**IncompatibleAreas**

Bases: exceptions.Exception

Error raised upon compositing things of different shapes.

class mpop.composites.**RGBCompositor**(*name*, *prerequisites=[]*, *optional_prerequisites=[]*, *metadata_requirements=[]*, ***kwargs*)

Bases: *mpop.composites.CompositeBase*

class mpop.composites.**SunCorrectedRGB**(*name*, *prerequisites=[]*, *optional_prerequisites=[]*, *metadata_requirements=[]*, ***kwargs*)

Bases: *mpop.composites.RGBCompositor*

class mpop.composites.**SunZenithNormalize**

Bases: object

coszen = {}

mpop.composites.**load_compositors**(*composite_names=None*, *sensor_names=None*, *ppp_config_dir='/home/docs/checkouts/readthedocs.org/user_builds/mpop/envs/featu simplify/lib/python2.7/site-packages/mpop-1.1.0-py2.7.egg/etc'*, ***kwargs*)

Load the requested *composite_names*.

Parameters

- **composite_names** – The name of the desired composites
- **sensor_names** – The name of the desired sensors to load composites for
- **ppp_config_dir** – The config directory

Returns A list of loaded compositors

2.1.2 mpop.readers package

Submodules

mpop.readers.eps_l1b module

mpop.readers.mipp_xrit module

mpop.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 mpop.readers.viirs_sdr.**HDF5MetaData**(*filename*, ***kwargs*)
Bases: object

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

collect_metadata(*name*, *obj*)

class mpop.readers.viirs_sdr.**SDRFileReader**(*file_type*, *filename*, *file_keys*, ***kwargs*)
Bases: *mpop.readers.GenericFileReader*

VIIRS HDF5 File Reader

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

begin_orbit_number

create_file_handle(*filename*, ***kwargs*)

end_orbit_number

geofilename

get_file_units(*item*)

get_shape(*item*)

get_swath_data(*item*, *data_out=None*, *mask_out=None*, *dataset_id=None*)
Get swath data, apply proper scalings, and apply proper masks.

platform_name

ring_lonlats()

scale_swath_data(*data*, *mask*, *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

class mpop.readers.viirs_sdr.**VIIRSSDRReader**(*default_file_reader=<class 'mpop.readers.viirs_sdr.SDRFileReader'>*, *default_config_filename='readers/viirs_sdr.cfg'*, ***kwargs*)
Bases: *mpop.readers.ConfigBasedReader*

Module contents

Shared objects of the various reader classes.

class mpop.readers.**ConfigBasedReader**(*default_file_reader=None*, ***kwargs*)
Bases: *mpop.readers.Reader*

file_key_class
alias of *FileKey*

identify_file_types(*filenames*, *default_file_reader=None*)
Identify the type of a file by its filename or by its contents.

Uses previously loaded information from the configuration file.

load(*datasets_to_load*, *metadata=None*, ***dataset_info*)

load_metadata(*datasets_to_load*, *metadata_to_load*)
Load the specified metadata for the specified datasets.

Returns dictionary of dictionaries

load_section_calibration(*section_name*, *section_options*)

load_section_file_key(*section_name*, *section_options*)

load_section_file_type(*section_name*, *section_options*)

load_section_navigation(*section_name*, *section_options*)

splittable_dataset_options = ['file_patterns', 'navigation', 'standard_name', 'units', 'file_type', 'file_key']

class mpop.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.

__setitem__(*key*, *value*)
Support assigning 'Projectable' objects or dictionaries of metadata.

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

get_key(*key*)

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

get_keys_by_datasetid(*did*)

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

mpop.readers.**DatasetID**
alias of Dataset

class mpop.readers.**FileKey**
Bases: *mpop.readers.FileKey*

class mpop.readers.**GenericFileReader**(*file_type*, *filename*, *file_keys*, ***kwargs*)
Bases: object

begin_orbit_number

create_file_handle(*filename*, ***kwargs*)

end_orbit_number

end_time

geofilename

get_file_units(*item*)

get_shape(*item*)

get_swath_data(*item*, *data_out=None*, *mask_out=None*, *dataset_id=None*)

get_units(*item*)

platform_name

ring_lonlats

sensor_name

start_time

class mpop.readers.**MultiFileReader**(*file_type*, *file_readers*, *file_keys*, ***kwargs*)
Bases: object

begin_orbit_number

end_orbit_number

end_time

filenames

geofilenames

get_swath_data(*item*, *filename=None*, *dataset_id=None*)

get_units(*item*)

load_metadata(*item*, *join_method='append'*, *axis=0*)

platform_name

sensor_name

start_time

class mpop.readers.**Reader**(*name=None*, *file_patterns=None*, *filenames=None*, *description=''*, *start_time=None*, *end_time=None*, *area=None*, *sensor=None*, ***kwargs*)
Bases: *mpop.plugin_base.Plugin*

Reader plugins. They should have a *pformat* attribute, and implement the *load* method. This is an abstract class to be inherited.

add_filenames(**filenames*)

dataset_names
Names of all datasets configured for this reader.

get_dataset_key(*key*, *calibration=None*, *resolution=None*, *polarization=None*, *aslist=False*)
Get the fully qualified dataset corresponding to *key*, either by name or centerwavelength.

If *key* is a *DatasetID* object its name is searched if it exists, otherwise its wavelength is used.

load(*datasets_to_load*)
Loads the *datasets_to_load* into the scene object.

load_metadata(*datasets_to_load*, *metadata_to_load*)
Load the specified metadata for the specified datasets.

Returns dictionary of dictionaries

load_section_dataset(*section_name*, *section_options*)

load_section_metadata(*section_name*, *section_options*)

load_section_reader(*section_name*, *section_options*)

sensor_names
Sensors supported by this reader.

splittable_dataset_options = ['file_patterns', 'navigation', 'standard_name', 'units']

class mpop.readers.**ReaderFinder**(*ppp_config_dir=None*, *base_dir=None*, ***info*)
Bases: object

Finds readers given a scene, filenames, sensors, and/or a reader_name

static assign_matching_files(*reader_info*, **files*, ***kwargs*)
Assign *files* to the *reader_info*

get_filenames(*reader_info*, *base_dir=None*)
Get the filenames from disk given the patterns in *reader_info*. This assumes that the scene info contains start_time at least (possibly end_time too).

2.1.3 mpop.writers package

Submodules

mpop.writers.geotiff module

mpop.writers.simple_image module

class mpop.writers.simple_image.**PillowWriter**(***kwargs*)
Bases: *mpop.writers.Writer*

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

Module contents

Shared objects of the various writer classes.

For now, this includes enhancement configuration utilities.

class mpop.writers.**EnhancementDecisionTree**(**config_files*, ***kwargs*)
Bases: object

add_config_to_tree(**config_files*)

any_key = None

find_match(***kwargs*)

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

load_section_writer(*section_name*, *section_options*)

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

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

mpop.writers.**get_enhanced_image**(*dataset*, *enhancer=None*, *fill_value=None*, *ppp_config_dir=None*, *enhancement_config_file=None*)

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

mpop.writers.**to_image**(*dataset*, *copy=True*, ***kwargs*)

2.2 Submodules

2.3 mpop.plugin_base module

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

class mpop.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.

get_section_type(*section_name*)

load_config(*conf*)

2.4 mpop.projectable module

Projectable objects.

class mpop.projectable.**Dataset**
Bases: MaskedArray

copy()

is_loaded()

class mpop.projectable.**InfoObject**(***attributes*)
Bases: object

class mpop.projectable.**Projectable**
Bases: *mpop.projectable.Dataset*

resample(*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
- **string corresponding to the name of the area as defined in the area file.** (*a*) –
- ****kwargs** – The extra parameters to pass to the resampling functions.

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

2.5 mpop.resample module

Shortcuts to resampling stuff.

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

The base resampler class. Abstract.

__call__(**args*, ***kwargs*)
Shortcut for the *resample()* method

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

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

precompute(***kwargs*)
Do the precomputation

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

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

Resample using nearest neighbour.

caches = OrderedDict()

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

get_hash(***kwargs*)
Get hash for the current resample with the given *kwargs*.

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

precompute(*radius_of_influence=10000*, *epsilon=0*, *reduce_data=True*, *nprocs=1*, *segments=None*, *cache_dir=False*, ***kwargs*)

mpop.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 mpop's configuration file.

mpop.resample.**get_area_file**()

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

2.6 mpop.scene module

Scene objects to hold satellite data.

class mpop.scene.**Scene**(*filenames=None*, *ppp_config_dir=None*, *reader_name=None*, *base_dir=None*, ***info*)

Bases: *mpop.projectable.InfoObject*

The almighty scene class.

available_composites()
Return the list of available composites for the current scene.

available_datasets(*reader_name=None*)
Return the available datasets, globally or just for *reader_name* if specified.

compute(**requirements*)
Compute all the composites contained in *requirements*.

get_writer(*writer='geotiff'*, ***kwargs*)

get_writer_by_ext(*extension*, ***kwargs*)

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

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

load_writer_config(*config_files*, ***kwargs*)

read(*dataset_keys*, *calibration=None*, *resolution=None*, *polarization=None*, *metadata=None*, ***kwargs*)
Read the composites called *dataset_keys* or their prerequisites.

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

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

save_datasets(*writer='geotiff'*, ***kwargs*)
Save all the datasets present in a scene to disk using *writer*.

show(*dataset_id*)
Show the *dataset* on screen as an image.

unload(*keepables=None*)
Unload all loaded composites.

2.7 mpop.tools module

Helper functions for eg. performing Sun zenith angle correction.

mpop.tools.**sunzen_corr_cos**(*data*, *cos_zen*, *limit=80.0*)
Perform Sun zenith angle correction to the given *data* using cosine of the zenith angle (*cos_zen*). The correction is limited to *limit* degrees (default: 80.0 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.

2.8 mpop.utils module

Module defining various utilities.

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

Empty handler.

emit(*record*)
Record a message.

class mpop.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

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

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

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

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

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

mpop.utils.**strftime**(*utctime*, *format_string*)
Like datetime.strftime, except it works with string formatting conversion specifier items on windows, making the assumption that all conversion specifiers use mapping keys.

E.g.: >>> from datetime import datetime >>> t = datetime.utcnow() >>> a = "blabla%Y%d%m-%H%M%S-%(value)s" >>> strftime(t, a) 'blabla20120911-211448-%(value)s'

2.9 Module contents

MPOP Package initializer.

mpop.**config_search_paths**(*filename*, **search_dirs*, ***kwargs*)

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

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

mpop.**get_environ_config_dir**(*default='/home/docs/checkouts/readthedocs.org/user_builds/mpop/envs/feature-simplify/lib/python2.7/site-packages/mpop-1.1.0-py2.7.egg/etc'*)

mpop.**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.

mpop.**runtime_import**(*object_path*)
Import at runtime

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

m

Symbols

A

B

C

D

E

N

O

P

R

S

T

U

V

W