Pygranule Documentation

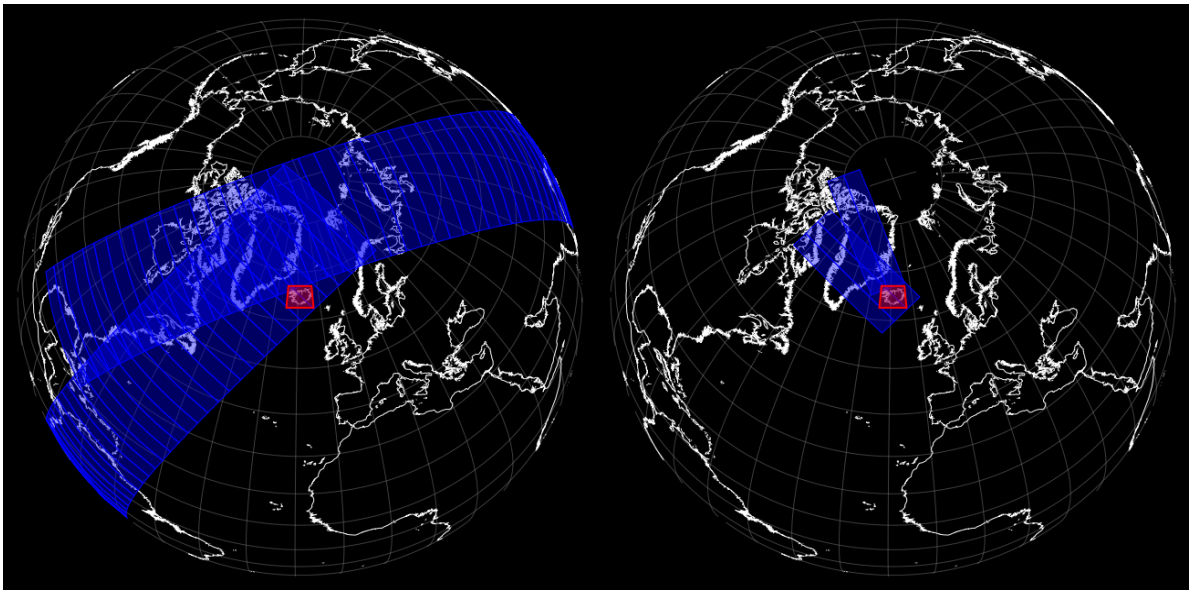
Release 0.1.0

Hrobjartur Thorsteinsson

March 03, 2016

Contents

pygranule is a package for validating, fetching and scheduling satellite data granules. The source code of the package can be found on google code, googlecode



The main purpose is to filter some satellite granule file names,

```
>>> files = ['/home/msg/archive/AVHRR/avhrr_20140225_133000_noaa19.hrp.bz2',
             '/home/msg/archive/AVHRR/avhrr_20140225_133100_noaa19.hrp.bz2',
             '/home/msg/archive/AVHRR/avhrr_20140225_133200_noaa19.hrp.bz2',
             ...,
             '/home/msg/archive/AVHRR/avhrr_20140225_152900_noaa19.hrp.bz2']
```

Using a configuration to describe the file set,

```
>>> config = {'config_name':"DummySatData",
              'sat_name':"NOAA 19",
              'instrument':"AVHRR",
              'file_source_pattern':"/home/msg/archive/AVHRR/avhrr_%Y%m%d_%H%M00_noaa19.hrp.bz2",
              'time_step':"00:01:00",
              'time_step_offset':"00:00:00",
              'area_of_interest':"(-25,62.5),(-25,67),(-13,67),(-13,62.5)"}
>>> gf = OrbitalGranuleFilter(config)
```

We can now do some on the fly operations on the files, e.g., show all the granules,

```
>>> gf.show( files )
```

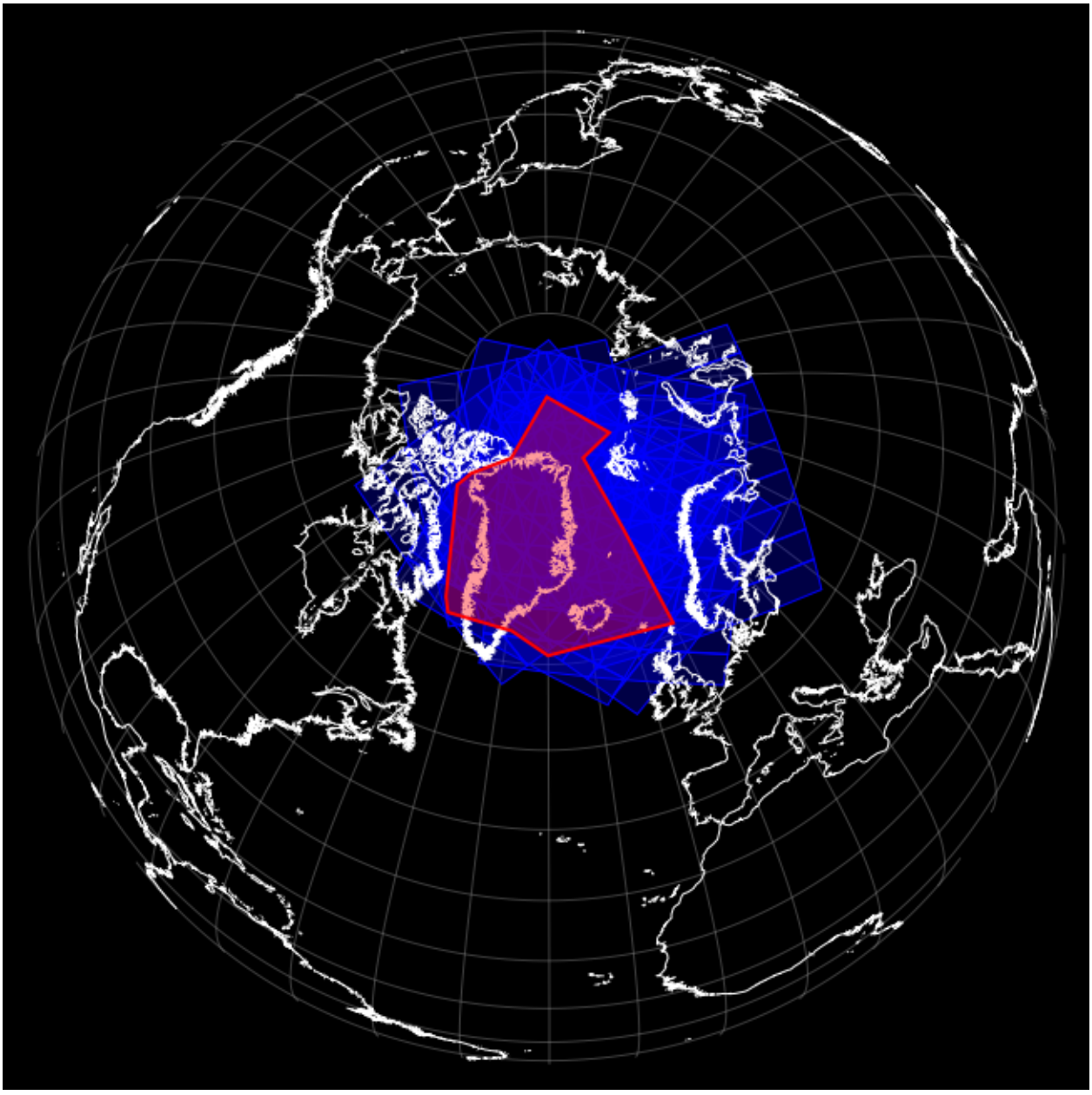
or filter the granules, allowing only those that match the configuration, returns an iterable file paths container but with a handle back to the parent filter, allowing handy attributes such as, `show()`,

```
>>> gf( files ).show()
```

Filters can also interact with local and remote file systems to access granule file names,

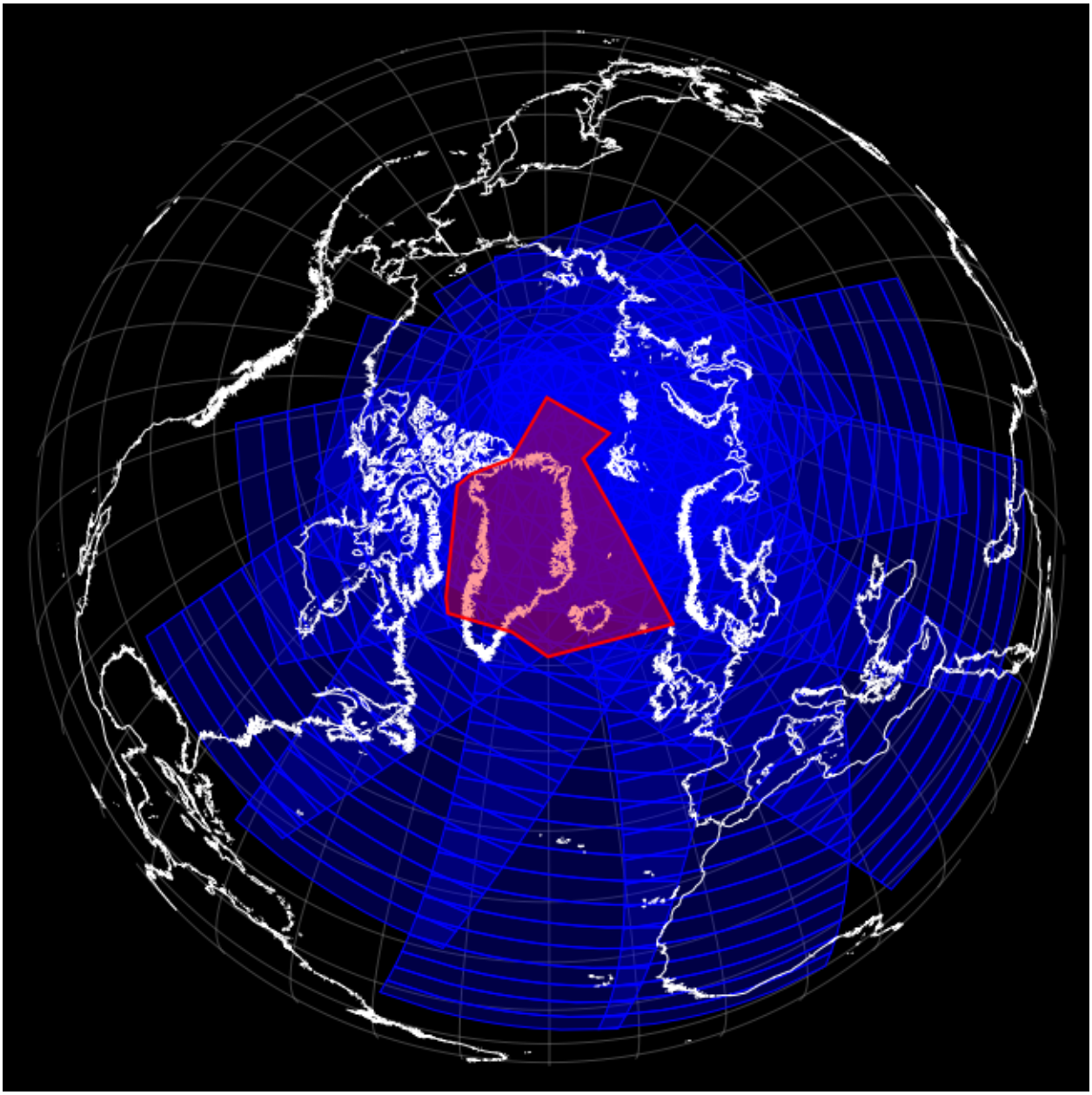
```
>>> config = {'config_name':"EARS NOAA 19 AVHRR",
              'sat_name':"NOAA 19",
              'instrument':"AVHRR",
              'protocol':"sftp",
              'server':"msg@msg01.vedur.is",
              'file_source_pattern':"/home/msg/archive/AVHRR/avhrr_%Y%m%d_%H%M00_noaa19.hrp.bz2",
```

```
                'time_step':"00:01:00",
                'time_step_offset':"00:00:00",
                'area_of_interest':"(0.0,73.0),(0.0,61.0),(-30.0,61.0),
                                    (-39,63.5),(-55.666,63.5),(-57.75,65),
                                    (-76,76),(-75,78),(-60,82),(0,90),(30,82),(0,82)"}
>>> F = OrbitalGranuleFilter(config)
>>> F().show()
```



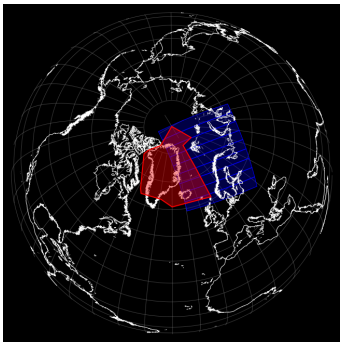
Now, if you wanted to turn off area of interest filtering to show all the granules in the source file system, then,

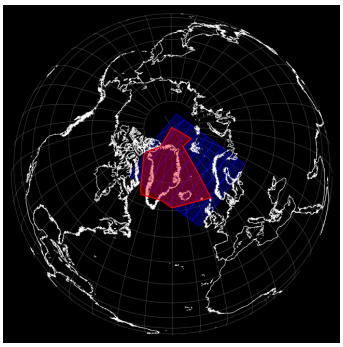
```
>>> F(with_aoi=False).show()
```

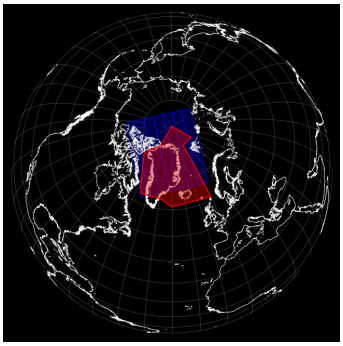


Targeted granules can also be readily split into chunks (satellite passes),

```
>>> for chunk in F().split():
>>>     chunk.show()
```







... Off course granule filter will be able to do much much more ...

Contents:

CHAPTER 1

Installation

You can download the getsat source code from googlecodes,:

```
$> git clone https://code.google.com/p/get-sat/
```

and then run:

```
$> python setup.py install
```

1.1 TODO

TODO.

CHAPTER 2

Usage (DRAFT)

The pygranule package has been developed to help abstract the acquisiton of satellite data granules. Procedures for collecting satellite earth observation data for a particular target region (area of interest) is a necessary fist step an automated processing chain.

Satellite Earth observing systems and instruments are varied. Many systems observe the earth continuously, either with a fixed repetition, or based on continuous scans (swaths) as the satellite moves across the Earth.

Commonly such earth observation data is delivered in a segmented way, separated into data granules that represent different time periods of data recorded by the satellite instrument. Furthermore, some instrument data is also segmented into data channels, and data subsets (e.g. MODIS HDF-EOS and SEVIRI-HRIT data).

The little nuances in data granulation and subsetting of different instruments can easily make acquisition code inhomogeneous and cluttered.

2.1 Acquisition config files

Simple configuration files are created to describe the data acquisition. The file specifies among other things the satellite, instrument type, target area, filename pattern, granulation and data subsetting.

From the config file pygranule filter should be able to evaluate if a particular incoming data granule is part of the target data set.

TODO: Eventually user provided trigger functions will be made available to perform actions on the data, including moving/downloading the data sets to a data archive.

Periodic type acquisition:

```
[HRIT_SEVIRI_MSG-3]
sat_name: MSG-3
sat_id:
instrument: SEVIRI
type: periodic
protocol: scp
server: msg01.vedur.is
file_source_pattern: /home/msg/archive/SEVIRI/HRIT/H-000-MSG3__-MSG3_________-{0}___-00000{1}___-%Y%m%
time_step: 00:15:00
time_step_offset: 00:00:00
subsets: {IR_108:{1..8}, WV_073:{1,2,3,4,5,6,7,8}}
```

Orbital type acquisition:

```
[EARS_AVHRR_NOAA-19]
sat_name: NOAA-19
sat_id:
instrument: AVHRR
type: orbital
protocol: scp
server: msg01.vedur.is
file_source_pattern: /home/msg/archive/AVHRR/avhrr_%Y%m%d_%H%M00_noaa19.hrp.bz2
time_stamp_alignment: 0.0 #beginning
time_step: 00:01:00
time_step_offset: 00:00:00
area_of_interest: (-25,62.5),(-25,67),(-13,67),(-13,62.5)
```

2.2 GranuleFilter

Granule filters are objects based on an acquisition configuration. They are used to validate and filter granule filenames.

They check if granules fit a particular file name pattern, subsets and granulation. They also verify that a granule observes (intersects) the area of interest.

2.2.1 loading filters from configs

First off, all acquisition config files should be placed in the same folder, and the PYGRANULE_CONFIG_PATH environment variable should point to this folder, e.g.,

```
$ export PYGRANULE_CONFIG_PATH=$HOME/etc/pygranule/
```

or when testing in the source code directory,

```
$ export PYGRANULE_CONFIG_PATH=$PWD/etc/
```

We can then load these configurations as GranuleFilter objects,

```
>>> from pygranule import get_granule_filters
>>> filters = get_granule_filters()
```

Every valid configuration file will then be loaded as part of an a GranuleFilter. The filters can be printed to show their configuration,

```
>>> for c in filter:
>>>     print filter[c]
```

2.2.2 creating a filter by hand

Alternatively GranuleFilters can be created by hand or in custom code,

```
>>> from pygranule import OrbitalGranuleFilter
>>>
>>> config = {'config_name':"NOAA_19_AVHRR",
              'sat_name':"NOAA 19",
              'file_source_pattern':"/home/msg/archive/AVHRR/avhrr_%Y%m%d_%H%M00_noaa19.hrp.bz2",
              'time_step':"00:01:00",
              'time_step_offset':"00:00:00",
```

```
            'area_of_interest':"(-25,62.5),(-25,67),(-13,67),(-13,62.5)"}
>>> gf = OrbitalGranuleFilter(config)
```

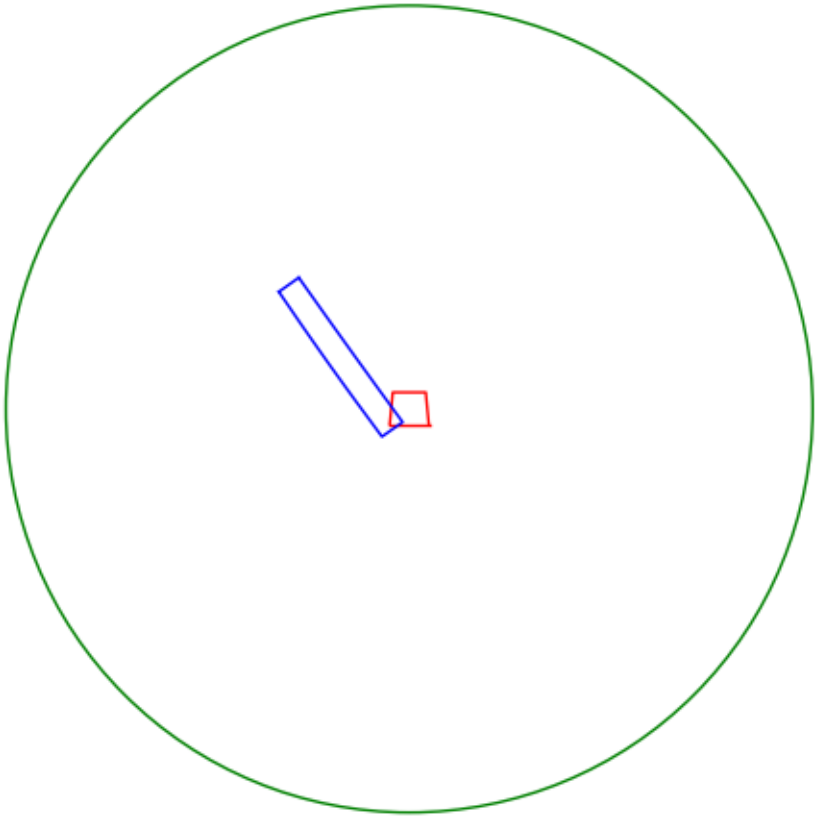
2.2.3 validating a granule filename

Incoming filenames can be evaluated against the filter configuration. A successful match is a filename that matches in pattern, subset, granulation (time_step) and samples the area of interest:

```
>>> filename="/home/msg/archive/AVHRR/avhrr_20140204_141500_noaa19.hrp.bz2"
>>> print "filename match:", gf.validate(filename)
--> filename match: True
```

Orbital type GranuleFilters have an OrbitalLayer instance (see below). The orbital layer accesses information from an orbital toolkit (pyorbital) to evaluate the intersection of the granule and AOI. With the orbital layer, we can plot the granule and AOI to check our result visually:

```
>>> t = gf.file_name_parser.time_from_filename(filename)
>>> gf.orbital_layer.show_swath(t,1)
```



2.2.4 filtering filenames

Perhaps the most common use for the filter would be to listing a directory for some filenames, then filtering out those that match the configuration:

```
>>> files = ["blabla",
             "H-000-MSG3__-MSG3________-WV_073___-000009___-201401231300",
             "/home/msg/archive/AVHRR/avhrr_20140225_133400_noaa19.hrp.bz2",
             "/home/msg/archive/AVHRR/avhrr_20140225_133500_noaa19.hrp.bz2",
             "/home/msg/archive/AVHRR/avhrr_20140225_133600_noaa19.hrp.bz2",
             "/home/msg/archive/AVHRR/avhrr_20140225_133700_noaa19.hrp.bz2",
             "/home/msg/archive/AVHRR/avhrr_20140225_160000_noaa19.hrp.bz2",
             "H-000-MSG3__-MSG3________-IR_108___-000003___-201401231300"]
>>>
>>> wanted_files = gf(files)
>>> print wanted_files
-->
["/home/msg/archive/AVHRR/avhrr_20140225_133400_noaa19.hrp.bz2",
 "/home/msg/archive/AVHRR/avhrr_20140225_133500_noaa19.hrp.bz2",
 "/home/msg/archive/AVHRR/avhrr_20140225_133600_noaa19.hrp.bz2"]
```

2.3 FileNameParser

Filename parser objects handle the parsing of a filename strings, verifying form, extracting time, subset names. The file name parser together with the orbital layer are used by GranuleFilter to verify filenames and area of interest intersects.

A filename parser must as bare minimum be instanciated with a format string as follows,

```
>>> fnp = FileNameParser("H-000-MSG3__-MSG3________-IR_108___-000001___-%Y%m%d%H%M")
```

The datetime formatting characters %Y%m%d%H%M should look familiar to python programmers. If not, please refer to the datetime documentation. The above parser handles parsing files that match that string exactly. Note how the above pattern is completely fixed at checking data channel (subset) 'IR_108', sub-subset '00001'. While in MSG3 SEVIRI data has a number of different channel subsets, with different combinations of subsubset segements.

We can actually create a parser that takes account of these subsets options by supplying the parser with a string listing out a tree of subsets and their sub-subsets:

```
>>> fnp = FileNameParser("H-000-MSG3__-MSG3________-{0}___-00000{1}___-%Y%m%d%H%M",
                         "{IR_108:{1..8}, WV_073:{1,2,3,4,5,6,7,8}}")
```

The placement of the subset strings in the pattern are denoted using the python string formatting style. This filename parser now accepts filenames such as,

```
H-000-MSG3__-MSG3________-IR_108___-000003___-201402121230
H-000-MSG3__-MSG3________-WV_073___-000007___-201402121233
```

but rejects

```
H-000-MSG3__-MSG3________-IR_108___-000009___-201402121230
H-000-MSG3__-MSG3________-IR_120___-000001___-201402121230
```

2.3.1 filenames from time

Using the above parser, we can generate all possible filenames from the pattern based on a given time,

```
>>> t = datetime(2014,1,23,13,55)
>>> fnp.filenames_from_time(t)
>>> print fnp
-->
['H-000-MSG3__-MSG3________-IR_108___-000001___-201401231355',
```

```
 'H-000-MSG3__-MSG3________-IR_108___-000003___-201401231355',
 'H-000-MSG3__-MSG3________-IR_108___-000002___-201401231355',
 'H-000-MSG3__-MSG3________-IR_108___-000005___-201401231355',
 'H-000-MSG3__-MSG3________-IR_108___-000004___-201401231355',
 'H-000-MSG3__-MSG3________-IR_108___-000007___-201401231355',
 'H-000-MSG3__-MSG3________-IR_108___-000006___-201401231355',
 'H-000-MSG3__-MSG3________-IR_108___-000008___-201401231355',
 'H-000-MSG3__-MSG3________-WV_073___-000001___-201401231355',
 'H-000-MSG3__-MSG3________-WV_073___-000003___-201401231355',
 'H-000-MSG3__-MSG3________-WV_073___-000002___-201401231355',
 'H-000-MSG3__-MSG3________-WV_073___-000005___-201401231355',
 'H-000-MSG3__-MSG3________-WV_073___-000004___-201401231355',
 'H-000-MSG3__-MSG3________-WV_073___-000007___-201401231355',
 'H-000-MSG3__-MSG3________-WV_073___-000006___-201401231355',
 'H-000-MSG3__-MSG3________-WV_073___-000008___-201401231355']
```

2.3.2 time from filename

Also, extracting the time signature in a valid granule filename is useful,

```
>>> t = fnp.time_from_filename("H-000-MSG3__-MSG3________-WV_073___-000006___-201401231355")
>>> print t,
--> 2014-01-23 13:55:00.000000
```

If the filename does not match the format string and/or allowed subsets, then this operation raises a ValueError exception. It is therefore recommended to validate the filename pattern first using the validate_filename method, see below.

2.3.3 subset from filename

If the filename parser has a subset definition, then it can extract valid subset names from the filename,

```
>>> print fnp.subset_from_filename('H-000-MSG3__-MSG3________-WV_073___-000003___-201401231355')
--> ('WV_073', '3'))
```

If the filename does not match the format string and/or allowed subsets, then this operation raises a ValueError exception. It is therefore recommended to validate the filename pattern first using the validate_filename method, see below.

2.3.4 validate filename

Filename strings can be validated against the parser format settings. Here's and example that return True,

```
>>> print fnp.validate_filename("H-000-MSG3__-MSG3________-WV_073___-000006___-201401231355")
--> True
```

And the followin check,

```
>>> print fnp.validate_filename("H-000-MSG3__-MSG3________-WV_073___-00000X___-201401231355")
--> False
```

returns False due to an invalid 'X' in the sub-subset placeholder.

2.4 OrbitalLayer

The granule acquisition objects fetch orbital information such as transit time, swath area and intersection with the area of interest through a standardised interface, 'pygranule.orbital_layer.OrbitalLayer'

The OrbitalLayer has then been implemented to access information from the pyorbital library: https://github.com/mraspaud/pyorbital The PyOrbitalLayer is the default orbital interface used by pygranule's OrbitalGranuleFilter.

2.4.1 PyOrbitalLayer

The following code fragment demonstrates how to instanciate a PyOrbitalLayer object for the Icelandic air traffic zone and the meteorological satellite, NOAA-19.

```
>>> from pygranule.pyorbital_layer import PyOrbitalLayer
>>> REYKJAVIK_ATC =((0.0,73.0),(0.0,61.0),(-30.0,61.0),(-39,63.5),
>>>                 (-55+4/6.0,63.5),(-57+45/60.0,65),(-76,76),(-75,78),
>>>                 (-60,82),(0,90),(30,82),(0,82))
>>> orb = PyOrbitalLayer(REYKJAVIK_ATC, "NOAA 19")
```

To evaluate the next transit time over the area of interest, 'AOI' (the time at which the satellite reaches the highest elevation relative to the center of the AOI), do

```
>>> t = print orb.next_transit()
```

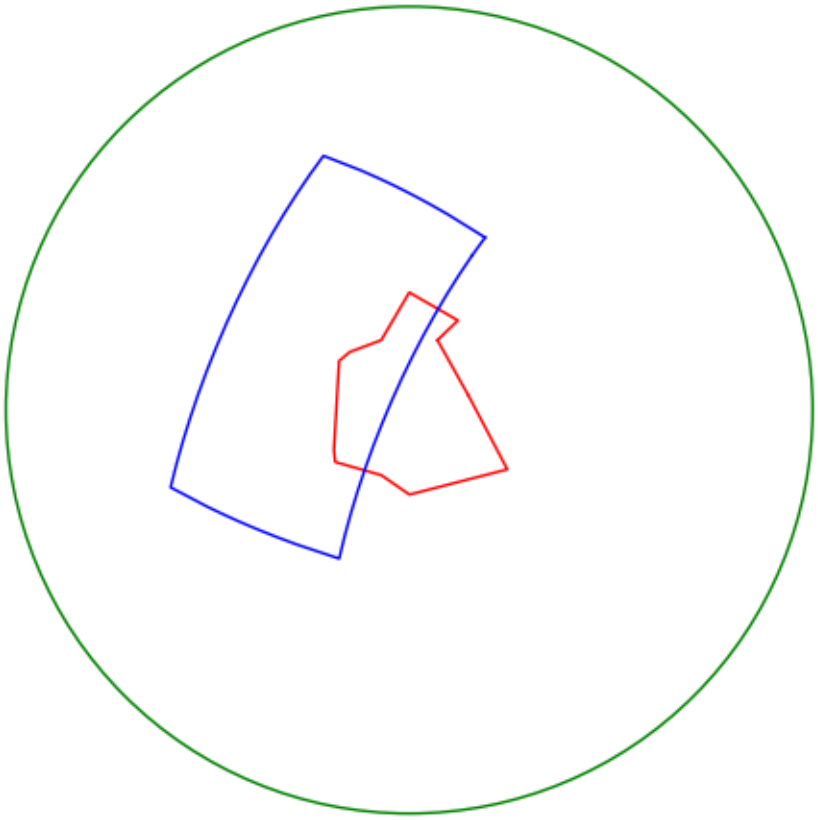
To evaluate the next sampling of the AOI (time and fractional coverage, where the satellite instrument swath samples/observs the AOI, we do,

```
    >>> t, f = orb.next_sampling()
    >>> print "sampling at",t, "will cover", f*100.0, "% of area."
```

—> sampling at 2014-02-04 17:38:37.838085 will cover 38.5896564452 % of area.

We can preview this satellite pass by using an inbuilt function to display the satellite swath and the AOI,

```
>>> orb.show_swath(t-timedelta(minutes=7),period=15.0)
```



2.4.2 Shapely objects

Orbital swaths of arbitrary length and revolutions can be loaded as shapely Polygon geometries. Same is true for the AOI,

```
>>> swa_poly = orb.swath_polygon(t,20)
>>> aoi_poly = orb.aoi_polygon()
```

The python shapely objects allow for a multitude of more complicated cross evaluations between the two areas...

CHAPTER 3

Examples (DRAFT)

3.1 Transferring files (DRAFT)

One very basic application of the GranuleFilter is to observer a source directory, filter out files of interest and transferring.

```
>>> from pygranule import get_granule_filters()
>>>
>>> Filters = get_granule_filters()
>>> for Filter in Filters:
>>>     Filter().transfer()
```

Done.

This method of executing a file transfer is the same, no matter in what form or subsetting the data appears.

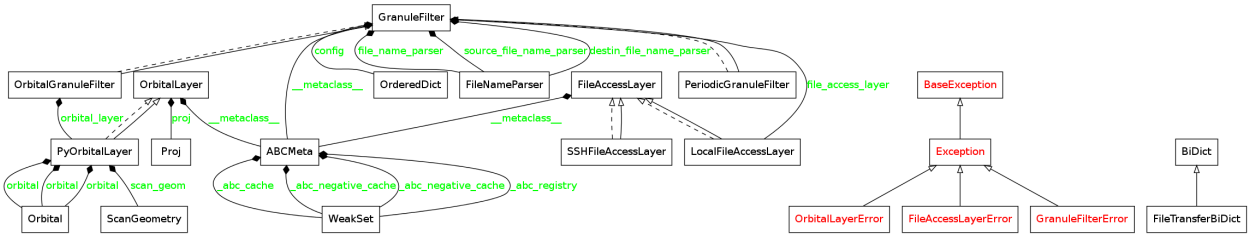
CHAPTER 4

Programming

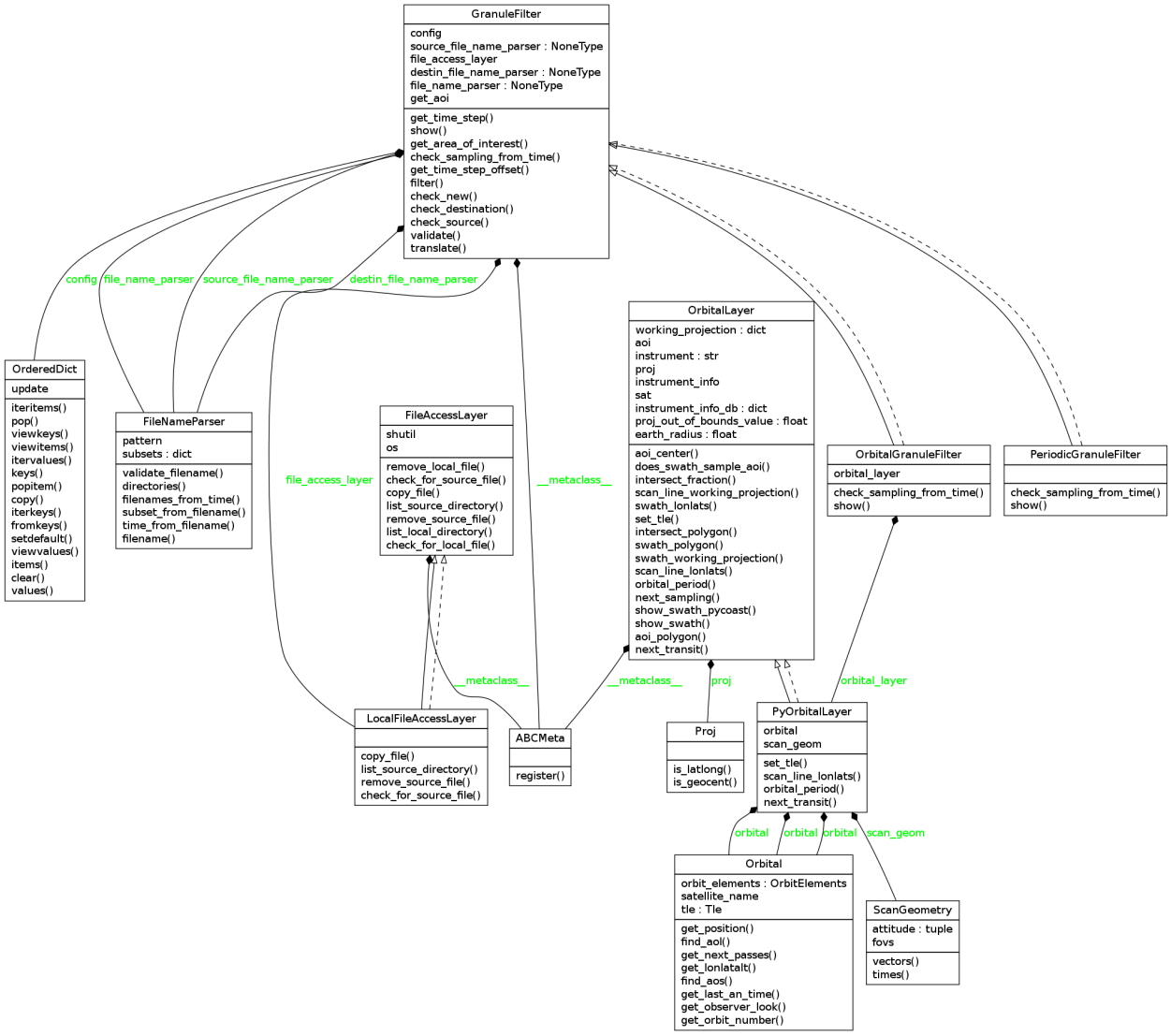
4.1 `pygranule` architecture (DRAFT)

The main high level module in pygranule is the GranuleFilter. The GranuleFilter holds the granule acquisition configuration, and aggregates more low level objects that assist it in evaluating satelltie granules. Notably, a FileNameParser, OrbitalLayer and a FileAccessLayer objects are aggregated and operated by the filter.

4.1.1 architecture overview



4.1.2 GranuleFilter in detail



4.2 `pygranule` API (DRAFT)

4.2.1 FileNameParser

4.2.2 GranuleFilter

4.2.3 OrbitalGranuleFilter

4.2.4 OrbitalLayer

CHAPTER 5

Indices and tables

- genindex
- modindex
- search