# naiad Documentation

***Release 0.1***

**Jeff Piolle**

December 16, 2016

Contents


**1 Installation** **3**
- 1.1 Prerequisites . . . 3
- 1.2 Installing the ElasticSearch cluster . . . 3
- 1.3 Test . . . 4

**2 Quickstart** **5**
- 2.1 Basic stuff . . . 5
- 2.2 Some more commands . . . 9

**3 naiad-tile command in details** **11**

**4 Customization** **13**
- 4.1 Granule and tile properties . . . 13
- 4.2 Creating custom granule and tile files . . . 13

**5 Examples** **17**
- 5.1 Examples of tiling for various products . . . 17

**6 Indices and tables** **21**

Contents:

CHAPTER 1

# Installation

## 1.1 Prerequisites

- ElasticSearch 2.x
- elasticsearch python package
- cerbere python package for data tiling only

## 1.2 Installing the ElasticSearch cluster

To setup the ElasticSearch cluster, you will need to install ElasticSearch on each node of the ES_NODES list.

This guide describes the installation of ElasticSearch from source, but you can use your operating system package manager to install it automatically.

### 1.2.1 Requirements

- SSH access to each node in ES_NODES
- Java JDK installed on each node

### 1.2.2 Instructions

On one of the nodes in ES_NODES (we will call this node NODE1):

1. Install elasticsearch

   Refer to Elasticsearch documentation for installation and set-up.

---

**Important:** Elasticsearch version must be >= 2.0.

---

2. Install the elasticsearch-head plugin

```
cd /opt/elasticsearch-2.3
bin/plugin -install mobz/elasticsearch-head
```

3. Install naiad

Get naiad source code:

```
git clone https://git.cersat.fr/cersat/naiad.git
```

Go into the source code repository and run:

```
sudo pip install .
```

4. Other useful python packages for contribs tilers

https://git.cersat.fr/cerbere/cerform

The following steps are for ElasticSearch installed on a cluster:

5. Edit configuration file config/elasticsearch.yml

```
cluster.name: felyx_cluster
network.host: <IP>
discovery.zen.minimum_master_nodes: 2
discovery.zen.ping.multicast.enabled: false
discovery.zen.ping.unicast.hosts: ["<IP1>", "<IP2>", ..., "<IPx>"]
```

If you plan to use a cluster, but only use one node, remove/comment the discovery.* properties.

- Replace <IP> by the actual IP address of the current node
- Replace <IP1>, <IP2>, ..., <IPx> by the IP addresses of all the nodes contained in ES_NODES.

5. Start ElasticSearch

```
/opt/elasticsearch-2.3/bin/elasticsearch
```

The first ElasticSearch node should now be up and running, all that's left is to do the same for the other nodes.

On each remaining node:

```
scp -r NODE1:/opt/elasticsearch-2.3 /opt
```

It should allow you to skip step 1., 2. and 3.

Then repeat steps 4. (you should just change the "network.host" setting) and 5.

## 1.3 Test

Open a webbrowser on http://<node>:9200/_plugin/head/ (replace <node> by the IP address of one the ElasticSearch nodes), you should see the cluster status page, listing all the nodes currently connected.

Check that they are all listed on this page: if a node is not there, check it's configuration and restart ElasticSearch.

CHAPTER 2

# Quickstart

## 2.1 Basic stuff

As a starter we will play with AVHRR L2P METOP product by OSI SAF.

### 2.1.1 Initialize a product index

We create first a quick index without any specific priorities. The index stores the spatial (shape) and temporal (time range) of each product granule.

We use the `naiad-create_index` command line tool:

```
naiad-create-index --elasticsearch http://localhost:9200/ avhrr_sst_metop_a-osisaf-l2p-v1.0
```

### 2.1.2 Tile some granules

We have to extract the global shape and time frame from the granules and the tiles constituting these granules.

We use the `naiad-tile` command line tool:

```
naiad-tile 20100123-EUR-L2P_GHRSST-SSTsubskin-AVHRR_METOP_A-eumetsat_sstmgr_metop02_20100123_000103-v
```

For more information on this, refer to naiad-tile command in details.

### 2.1.3 Register the granules

Simply ingest the content of the tile files generated above with `naiad-register` command:

```
naiad-register avhrr_sst_metop_a-osisaf-l2p-v1.0 20100123-EUR-L2P_GHRSST-SSTsubskin-AVHRR_METOP_A-eur
```

### 2.1.4 Get a granule description and display

We can get the description of a previously registered granule and also display its shape on a map using `naiad-granule-info` command:

```
naiad-granule-info ascat_a ascat_20100820_080601_metopa_19899_eps_o_coa_1100_ovw.l2.nc --show
```

### 2.1.5 Do a spatio-temporal search

Naiad is mainly about spatio-temporal search, so let's search for some granules intersecting an area and period of interest. This is done with `naiad-search` command.

Get everything! (be careful!):

```
naiad-search ascat_a --show
```

Add a time frame restriction:

```
naiad-search ascat_a --start 2010-08-19 --end 2010-08-20 --show
```

And now some area, defined as lonmin, latmin, lonmax, latmax:

```
naiad-search ascat_a --start 2010-08-19 --end 2010-12-20 --area='-20,-20,20,20' --show
```

You can request several products at the same time using a coma-separated list:

```
naiad-search ascat_a,avhrr_sst_metop_a-osisaf-l2p-v1.0  --start 2010-08-19 --end 2010-12-20 --area='
```

You can specify some constraints on the granule properties, if they have been registered in the product index. For instance, get all granules with a *file_quality_level* greater than 3 (if such a property was defined).

```
naiad-search amsr2 --granule_constraints=file_quality_level.ge.3
```

Multiple constraints can be combined, separating the different constraints by a `;`:

```
naiad-search amsr2 --granule_constraints='file_quality_level.ge.3;file_quality_level.lt.3'
```

### 2.1.6 Do a cross search (colocation)

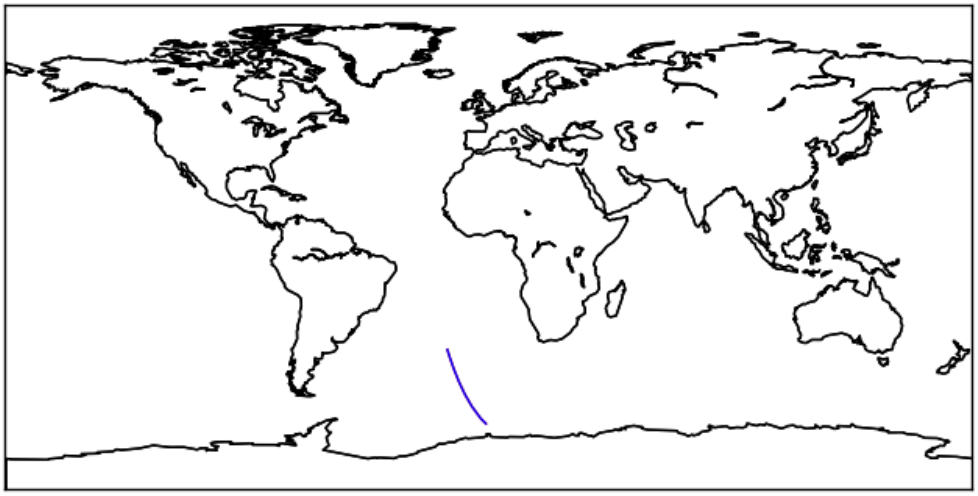
We want to keep only the granules from different products that cross each other within a time window.

One product, given by `--cross` argument is considered as the "reference" sensor, the one with which all other products, given by `--versus` argument must match within the time window given by `--time_window`:

```
naiad-cross-search --cross amsr2 --versus srl_oprssha_2 --time_window=720
```

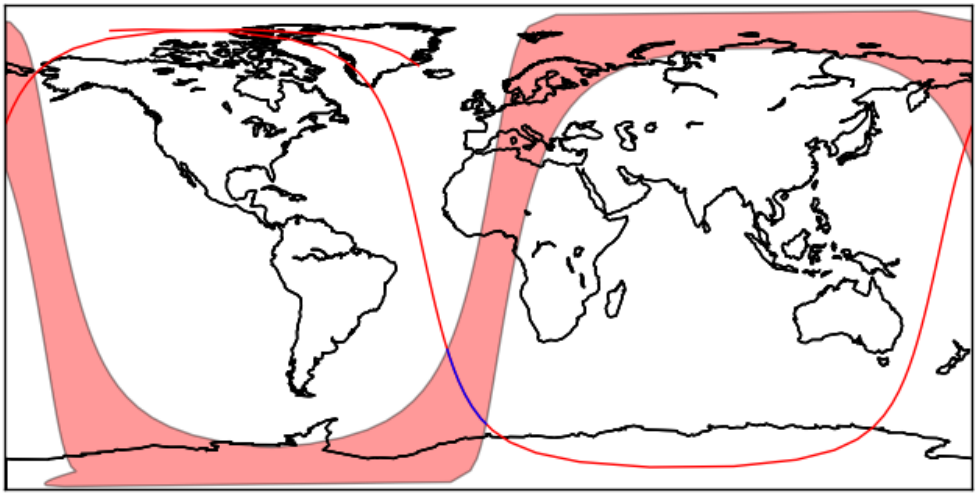
You can display the crossovers one by one with `--show` option:

```
naiad-cross-search --cross amsr2 --versus srl_oprssha_2 --time_window=720 --show
```



Use `--full_footprint` option in addition to display the full footprint of the colocated granules(usefull for checking).

```
naiad-cross-search --cross amsr2 --versus srl_oprssha_2 --time_window=720 --full_footprint --show
```



### 2.1.7 Use filters in search

If some properties were indexed together with the granules or the tiles, they can be used as search filters, using either `--granule_constraints` to filter on granule properties. For instance let's assume we gave indexed the granules of *slstr* product with a property *version* specifying the algorithm version used for a specific granule. We can query the granules having *version* number equal to 2.3:

```
naiad-search slstr  --granule_constraints='version eq 2.3'
```

Note the available comparison operators: `eq`, `lt`, `le`, `gt`, `ge`.

Filters can be combined (logical AND) using `;`:

```
naiad-search slstr  --granule_constraints='version eq 2.3';'cloud_fraction le 0.5'
```

It can be used also in a cross-over search, but the expressed constraint (for now) only applies to the reference dataset, defined by `--cross argument`. In the following example, we request the cross-overs between *slstr* and *metop_a* datasets where the *version* property for *slstr* granules equals 2.3:

```
naiad-cross-search --cross slstr --versus metop_a --time_window=15--start 20160520T000000 --end 2016
```

## 2.2 Some more commands

### 2.2.1 Delete an index

This will delete the index and all registered granules and tiles. Use with caution.

```
curl -XDELETE http://localhost:9200/avhrr_sst_metop_a-osisa-l2p-v1.0/
```

CHAPTER 3

# naiad-tile command in details

The `naiad-tile` commandline tool allows to extract the metadata from a granule file at global level (properties or attributes valid for the whole file content) and at a finer division level (subsets of the granule refered to as **tiles**).

These properties are later registered into an index (implemented by ElasticSearch). These properties include:

- time frame of the data
- spatial boundaries (shape) of the observed area
- any custom property that one may want to extract from the data, as a future search/selection criterium.

The most basic behaviour is to call `naiad-tile` with the two mandatory arguments:

```
naiad-tile <file> <mapper>
```

where:

- `file` is the full path to the granule file
- `mapper` is the name of the mapper class in **cerbere** package to read this type of granule.

for instance:

```
naiad-tile 20100123-EUR-L2P_GHRSST-SSTsubskin-AVHRR_METOP_A-eumetsat_sstmgr_metop02_20100123_000103-v
```

This will result in a single tile matching the full area covered by the granule. You see that by using the `--show` option:

```
naiad-tile --show 20100123-EUR-L2P_GHRSST-SSTsubskin-AVHRR_METOP_A-eumetsat_sstmgr_metop02_20100123_(
```

This is not exactly what we want, because tiles are meant for fine spatial and temporal subsetting. A good compromise is usually to have a tile size of about 500km along and accross track. This can be set by using the `--xstep` and `--ystep` (along-track) options where the value is given in pixels. Adjust this value so that the tile size is about 500km.

```
naiad-tile --show --xstep=540 --ystep=540 20100123-EUR-L2P_GHRSST-SSTsubskin-AVHRR_METOP_A-eumetsat_s
```

CHAPTER 4

# Customization

Naiad can be customized to better suit your need in various ways:

- adding custom properties to the indexed granules and tiles
- generating your self the granule and tile files

## 4.1 Granule and tile properties

By default, Naiad only index the spatial and temporal boundaries of the granules and tiles registered into the system (done by `naiad-tile` command).

However it is possible to index more metadata for both granules and tiles in order to later perform search on additional criteria rather than just a geographic area and time frame. For instance, you would want only granules or granule subsets over an area that have a mean solar zenital angle greater than 90 degrees (nighttime data).

This can be done by implementing a python class inheriting from `Tiler` class, where you will override two methods:

- `process_granule_properties()` : add additional properties to granule metadata
- `process_tiles_properties()` : add custom properties to the spatial/temporal properties (or metadata) of a granule's tiles

In the following example, we design a specific tiler class for GHRSST products: we want to store the mean solar zenital angle for each granule's tile to later select only night time granule subsets. To do so, we create a new class inheriting from :class:Tiler class where we implement the two forementioned methods.

To apply this class in the tiling process, there is no need to write a new command line tool replacing `naiad-tile`. Just call `naiad-tile` with `--tile-class <your class>` option, which in above case would give:

```
naiad-tile 20100123-EUR-L2P_GHRSST-SSTsubskin-AVHRR_METOP_A-eumetsat_sstmgr_metop02_20100123_000103-
```

## 4.2 Creating custom granule and tile files

The easiest way to produce the granule and tile metadata to be indexed into Naiad is using the `naiad-tile` command line tool. However it gives you little flexibility to produce your own metadata content.

We showed above a first method to customize your metadata, by heritage of the `Tiler` class.

This is not the only way : you can also edit or create these metadata yourself by writing your own tile files (that you will then register with `naiad-register` command) or even writing json documents that you will directly index in

elasticsearch (this is for very advanced users as you have to know how to work with elasticsearch and be fully aware of how naiad stores the granule and tile metadata into elasticsearch indices.

Writing your own tile files is very easy. A tile file will generally contain a line containing the metadata of your granule file and as many lines as you have tiles breaking down this granule into smaller pieces.

Here is an example:

```
S3A_SL_1_RBT____20160523T083711_20160523T084011_20160523T102935_0179_004_249_4499_MAR_O_NR_001.SEN3;2
71.912800 -97.172300,-72.321800 -97.839300,-72.731400 -98.546800,-73.142900 -99.314000,-73.543800 -10
584000,-75.900000 -105.630000,-76.276500 -106.738000,-76.649400 -107.898000,-77.016100 -109.133000,-7
063100 -118.113000,-79.369900 -119.935000,-79.673200 -121.864000,-79.957400 -123.895000,-80.231300 -1
.193000,-85.041100 -104.256000,-85.903700 -71.995200,-85.232200 -38.290600,-85.180700 -38.721700,-84.
-49.621600,-82.229600 -50.719800,-81.791300 -51.744700,-81.348900 -52.660900,-80.904600 -53.463300,-8
0 -57.077100,-77.760100 -57.530500,-77.311300 -57.957800,-76.858500 -58.348800,-76.404900 -58.712900,
200 -60.485900,-73.214200 -60.713700,-72.757600 -60.941500,-72.300700 -61.157900,-72.476400 -70.10270
S3A_SL_1_RBT____20160523T083711_20160523T084011_20160523T102935_0179_004_249_4499_MAR_O_NR_001.SEN3;2
71.912800 -97.172300,-72.321800 -97.839300,-72.731400 -98.546800,-73.142900 -99.314000,-73.543800 -10
584000,-75.900000 -105.630000,-76.276500 -106.738000,-76.649400 -107.898000,-77.016100 -109.133000,-7
063100 -118.113000,-79.369900 -119.935000,-79.673200 -121.864000,-79.957400 -123.895000,-80.231300 -1
.193000,-85.041100 -104.256000,-85.903700 -71.995200,-85.232200 -38.290600,-85.180700 -38.721700,-84.
-49.621600,-82.229600 -50.719800,-81.791300 -51.744700,-81.348900 -52.660900,-80.904600 -53.463300,-8
0 -57.077100,-77.760100 -57.530500,-77.311300 -57.957800,-76.858500 -58.348800,-76.404900 -58.712900,
200 -60.485900,-73.214200 -60.713700,-72.757600 -60.941500,-72.300700 -61.157900,-72.476400 -70.10270
...
```

The first line describes a granule, with the following sequence of fields, separated by `;`:

- the granule name, here `S3A_SL_1_RBT____20160523T083711_20160523T084011_20160523T102935_0179_00`
- the granule start time, here `2016-05-23T08:37:11`
- the granule end time, here `2016-05-23T08:40:11`
- the granule spatial shape, as a wkt string, here `POLYGON ((-70.660400 -95.301800,-71.078900 -95.895700,-71.495400 -96.515400,-71.912800 -97.172300,-72.321800 -97.839300,-72.731400 -98.546800,-73.142900 -99.314000,-73.543800 -100.068000,-73.949600 -100.893000,-74.344400 -101.741000,-74.738600 -102.641000,-75.128900 -103.586000,-75.518000 -104.584000,-75.900000 -105.630000,-76.276500 -106.738000,-76.649400 -107.898000,-77.016100 -109.133000,-77.377400 -110.419000,-77.729700 -111.785000,-78.074000 -113.249000,-78.413600 -114.772000,-78.742800 -116.407000,-79.063100 -118.113000,-79.369900 -119.935000,-79.673200 -121.864000,-79.957400 -123.895000,-80.231300 -126.044000,-80.490900 -128.346000,-80.736200 -130.722000,-80.966300 -133.261000,-80.991400 -133.564000,-83.205600 -123.193000,-85.041100 -104.256000,-85.903700 -71.995200,-85.232200 -38.290600,-85.180700 -38.721700,-84.777400 -41.230800,-84.364700 -43.400200,-83.951100 -45.278100,-83.527300 -46.917300,-83.096100 -48.325000,-82.661900 -49.621600,-82.229600 -50.719800,-81.791300 -51.744700,-81.348900 -52.660900,-80.904600 -53.463300,-80.456300 -54.201600,-80.013000 -54.892600,-79.566100 -55.509400,-79.114900 -56.075500,-78.665100 -56.606600,-78.215700 -57.077100,-77.760100 -57.530500,-77.311300 -57.957800,-76.858500 -58.348800,-76.404900 -58.712900,-75.950400 -59.054900,-75.492600 -59.366100,-75.037700 -59.667700,-74.584400 -59.962600,-74.129100 -60.222600,-73.673200 -60.485900,-73.214200 -60.713700,-72.757600 -60.941500,-72.300700 -61.157900,-72.476400 -70.102700,-72.246100 -79.020600,-71.625100 -87.534400,-70.660400 -95.301800 ))`

- the granule observation pattern among **Swath**, **Grid** and **Trajectory**, here `Swath`
- additional metadata properties, as a dictionary of (*property name*, *property value*), here `{"version": "2.3"}`

The tiles follow the same sequence of fields, except that before the additional metadata properties you must have the offsets in the file of the data subset corresponding to the tile:

- min column, max column, min line, max line for **Grid** or **Swath** patterns, here `0;1199;0;1499`
- min indice, max indice along the time axis for a **Trajectory** pattern

You can then register the tile files with the usual *naiad-register*‘ command.

CHAPTER 5

# Examples

## 5.1 Examples of tiling for various products

### 5.1.1 KNMI L2B products for ASCAT-A and ASCAT-B

ASCAT has a double swath. This has to be specified or the resulting shape will not separate both tiles.

```
    naiad-tile --doubleswath=41 --xstep 41 --ystep 41 --orientation=counterclockwise "/home/cerdata/
```

### 5.1.2 IASI L2P from EUMETSAT / OSI SAF

These are short (3 minutes along-swath) granules. We don't want to tile them along the 'row' axis (along-track axis). We don't specify any *ystep* argument. The tiler will automatically adjust the tile to the granule row length.

```
    naiad-tile --xstep 40  --display foo GHRSSTNCFile
```

### 5.1.3 GPM JAXA L2A

```
    naiad-tile --ystep 50 GPMCOR_KUR_1501211635_1808_005104_L2S_DU2_03B.h5 GPMHDF5File
```

### 5.1.4 AATSR L2P GHRSST

```
    naiad-tile --ystep 512  ../test/20100819-ATS_NR_2P-UPA-L2P-ATS_NR__2PNPDE20100819_012520_0000453
```

### 5.1.5 AVHRR L2P from OSI SAF

```
    naiad-tile --xstep 512 --ystep 540 --outpath 20100508-EUR-L2P_GHRSST-SSTsubskin-AVHRR_METOP_A-eu
```

### 5.1.6 PODAAC Rapidscat data

**Several issues exist with the Rapidscat BUFR data:**

- swaths are not complete and miss some sections, creating spatial discontinuities in the file.
- the lat/lon contain a lot of invalid data (nan).

```
naiad-tile --xstep=21 --ystep 20 --view="{'cell':slice(0,41)}" --tiling_method="corners"  --use_
```

### 5.1.7 MODIS L2P GHRSST from PODAAC

The MODIS L2P have some issues with invalid lat/lon.

```
naiad-tile --xstep=452 --ystep=677 --tiling_method="corners" jpl-l2p-modis_a/data/2015/001/20150
```

### 5.1.8 SARAL L2 from EUMETSAT

Saral altimeter data are along-track observations. Use the *Trajectory* feature class, and `--ystep` argument to specify the size of the tiles.

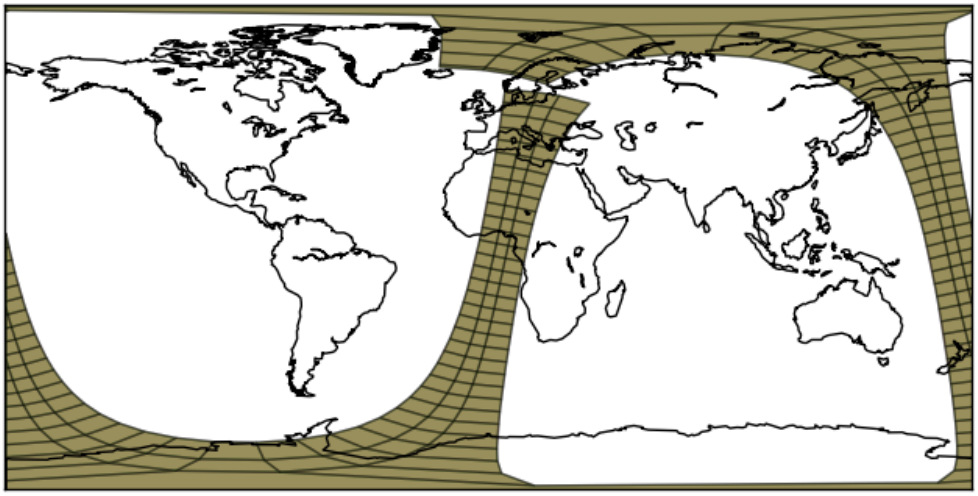
```
naiad-tile ../test/data/saral/SRL_OPRSSHA_2PTS026_0007_20150806_112515_20150806_130348.EUM.nc Cr
```

### 5.1.9 AMSR2 L2P from REMSS

```
naiad-tile --xstep 50 --ystep 50  --feature Swath --orientation=clockwise --show ../test/data/20
```

### 5.1.10 IASI 1C spectra data in NetCDF, EUMETSAT

```
naiad-tile --xstep=40 --ystep=10 /media/sumba/Iomega_HDD/iasi/W_XX-EUMETSAT-Darmstadt,HYPERSPECT+SOUN
```



CHAPTER 6

# Indices and tables

- genindex
- modindex
- search