Nansat Documentation

Anton Korosov, Morten W. Hansen, Aleksander Vines, Artem Moiseev

Aug 10, 2018
Contents:

1 Installation 3
  1.1 Quickstart .............................................................................................................. 3
  1.2 Requirements .......................................................................................................... 3
  1.3 Installing Requirements ........................................................................................ 4
  1.4 Installing Nansat .................................................................................................... 5
  1.5 Use a self-provisioned Virtual Machine ................................................................. 5

2 Nansat tutorials 7
  2.1 Nansat: First Steps ............................................................................................... 7

3 Packages and modules 13
  3.1 nansat .................................................................................................................... 13
  3.2 nansat.mappers package ...................................................................................... 65

4 Please acknowledge Nansat 81

5 Differentiating between land and water 83

6 Digital Elevation Models (DEMs) 85
  6.1 Global Multi-resolution Terrain Elevation Data 2010 (GMTED2010) ................. 85
  6.2 Global 30 Arc-Second Elevation (GTOPO30) ..................................................... 85

7 Nansat conventions 87
  7.1 Git branching and merging ................................................................................... 87
  7.2 How to report and handle new issues (bugs, improvements, new features, etc.) .... 88
  7.3 General conventions .............................................................................................. 89
  7.4 Naming conventions ............................................................................................. 90
  7.5 Style checking ....................................................................................................... 91
  7.6 Tests ....................................................................................................................... 91

8 About Nansat mappers 93
  8.1 Concept .................................................................................................................. 93
  8.2 Workflow ............................................................................................................... 93
  8.3 Technical details .................................................................................................... 94
  8.4 Where to put new mappers? .................................................................................. 94
  8.5 Required metadata added in the mappers ............................................................ 95
  8.6 Adding mapper tests ............................................................................................. 95
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Releasing Nansat</td>
<td>97</td>
</tr>
<tr>
<td>9.1</td>
<td>General release procedure</td>
<td>97</td>
</tr>
<tr>
<td>9.2</td>
<td>Releasing on PiPy</td>
<td>97</td>
</tr>
<tr>
<td>9.3</td>
<td>Releasing on Anaconda</td>
<td>98</td>
</tr>
<tr>
<td>10</td>
<td>Documenting Nansat</td>
<td>99</td>
</tr>
<tr>
<td>11</td>
<td>Indices and tables</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>Python Module Index</td>
<td>103</td>
</tr>
</tbody>
</table>
Nansat is a scientist friendly Python toolbox for processing 2D satellite earth observation data.

The main **goal** of Nansat is to facilitate:

- easy development and testing of scientific algorithms,
- easy analysis of geospatial data, and
- efficient operational processing.

You can also find a detailed description of Nansat in our paper published in *Journal of Open Research Software* in 2016.

... and you can join the mailing list.
1.1 Quickstart

The fastest way to install nansat:

- Install Miniconda on your platform of choice.

```
conda create -n nansat -c conda-forge nansat
source activate nansat
```

Nansat is now installed. For more details and other methods of installing Nansat, see below.

1.2 Requirements

Nansat requires the following packages:

- Python 2.7 or higher
- Numpy >=1.11.3
- GDAL >=2.2.3
- Pillow >=4.0.0
- netCDF4 >=1.3.1
- py-thesaurus-interface

The following packages are optional:

- Scipy 0.18.1
- Some mappers will not work without scipy. E.g. sentinel1_I1
- Matplotlib >=2.1.1
- matplotlib is required for Nansat methods digitize_points() and crop_interactive()
1.3 Installing Requirements

You have three main options on how to install the requirements. These are described in the following three sections.

1.3.1 Install dependencies from Anaconda

This is the recommended approach for installing dependencies.

- Download Miniconda for your platform of choice.
- Install Miniconda
- When you install Miniconda on Windows, you will get a new app called “Anaconda Prompt”. Run this to access the conda installation.
- On Linux/OS X use a regular terminal and make sure PATH is set to contain the installation directory as explained by the installer.
- Run the following three commands:
  - `conda create -n nansat Python=3.6`
  - Or use Python version 3.5 or 2.7 if you need those versions.
  - `source activate nansat`
  - On Windows you would omit ‘source’ and just run ‘activate nansat’
  - `conda install --yes -c conda-forge pythesint numpy scipy=0.18.1 matplotlib basemap netcdf4 gdal pillow urllib3`

1.3.2 Install Pre-built Binaries

One can find pre-built binaries available for different platforms. We do not have an overview over all the possible repositories where you can find binaries. But if you e.g. are onUbuntu, the following procedure can be used to install dependencies with `apt` and `pip`.

```bash
sudo apt install virtualenv libgdal1-dev python-dev python-gdal python-numpy python-scipy python-matplotlib python-mpltoolkits.basemap python-requests
cd virtualenv --no-site-packages nansat_env
source ~/nansat_env/bin/activate
export PYTHONPATH=/usr/lib/python2.7/dist-packages/
pip install pythesint pillow netcdf4 urllib3
```
1.3.3 Compile and Build Yourself

If you have the technical expertise to build all dependencies, and need to do it yourself, feel free to do so. If you need some aid, we would recommend you to look at how the corresponding conda-forge feedstocks have been built.

1.4 Installing Nansat

1.4.1 Install Nansat from source

If you want to install Nansat from source, you first need to install all requirements. Then proceed with one of the following methods

Install from git repository

- git clone the master (most stable) or develop (cutting edge) branch, and install:
  ```
  git clone https://github.com/nansencenter/nansat.git
cHECKOUT master (or develop, or a specific tag or branch)
python setup.py install
  ```

Nansat will then be added to your site-packages and can be used like any regular Python package.

Install with pip

- Run the following command:
  ```
pip install nansat
  ```

Nansat will then be added to your site-packages and can be used like any regular Python package.

1.4.2 Special install for Nansat Developers

If you are working directly on the Nansat source, you need to install Nansat in the following way.

Git clone the develop branch (or another branch you are working on), and do:

```
python setup.py build_ext --inplace
```

The pixel functions C module is then compiled but no code is copied to site-packages and no linking is performed. Make sure to follow the Nansat conventions if you want to contribute to Nansat.

In addition to the regular dependencies, developers also need to install nose and mock. This can easily be done with

```
pip install nose mock
```

1.5 Use a self-provisioned Virtual Machine

Another option to install Nansat in a controlled environment is to use a virtual machine. Configuration for Vagrant and Ansible that brings up and provision a VirtualBox machine is provided in Nansat repository. To start the machine
you need to install Vagrant and VirtualBox on your computer; clone or download the nansat source code; and start the machine:

```bash
# download nansat source code
git clone https://github.com/nansencenter/nansat.git

cd nansat

# start virtual machine
vagrant up
```

That’s it! The virtual machine will be started and all software will be installed automatically. To start using Nansat you need to log in to the virtual machine and start Python from the conda environment:

```bash
vagrant ssh
source activate py3nansat
python
```
The package nansat-lectures contains several Jupyter notebooks with examples of how to use Nansat. Unfortunately, we have not been able to keep them fully updated. The most recently updated notebooks should, however, work.

### 2.1 Nansat: First Steps

#### 2.1.1 Overview

The NANSAT package contains several classes:

- Nansat - open and read satellite data
- Domain - define grid for the region of interest
- Figure - create raster images (PNG, TIF)
- NSR - define spatial reference (SR)

#### 2.1.2 Copy sample data

In [1]:
```
import os
import shutil
import nansat
idir = os.path.join(os.path.dirname(nansat.__file__), 'tests', 'data/
```

#### 2.1.3 Open file with Nansat

In [2]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```
from nansat import Nansat
n = Nansat(idir+'gcps.tif')

2.1.4 Read information ABOUT the data (METADATA)

In [3]: print n
2.1.5 Read the actual DATA

In [4]: b1 = n[1]

2.1.6 Check what kind of data we have

In [5]: %whos
   
   Variable Type Data/Info
   ------------------------
   Nansat type <class 'nansat.nansat.Nansat'>
   b1 ndarray 200x200: 40000 elems, type 'uint8', 40000 bytes
   idir str /home/docs/checkouts/readthedocs.org/user_builds/nansat/conda/latest/lib/python2.7/site-packages/nansat/tests/data/gcps.tif
   n Nansat -----------------------<...>0.72) ( 29.68, 70.35)

2.1. Nansat: First Steps
2.1.7 Find where the image is taken

In [6]: n.write_map('map.png', pltshow=True)

/home/docs/checkouts/readthedocs.org/user_builds/nansat/conda/latest/lib/python2.7/site-packages/nansat/domain.py:837: NansatFutureWarning: Domain.write_map() will be disabled in Nansat 1.1. Use nansat.write_domain_map().
  warnings.warn(self.WRITE_MAP_WARNING, NansatFutureWarning)
3.1 nansat

3.1.1 nansat package

Submodules

nansat.domain module

class nansat.domain.Domain (srs=None, ext=None, ds=None, **kwargs)
   Bases: object

   Container for geographical reference of a raster

   A Domain object describes all attributes of geographical reference of a raster:
   
   • width and height (number of pixels)
   • pixel size (e.g. in decimal degrees or in meters)
   • relation between pixel/line coordinates and geographical coordinates (e.g. a linear relation)
   • type of data projection (e.g. geographical or stereographic)

   Parameters
   
   • srs (PROJ4 or EPSG or WKT or NSR or osr.SpatialReference()) – Input parameter for nansat.NSR()
   • ext (string) – some gdalwarp options + additional options [http://www.gdal.org/gdalwarp.html] Specifies extent, resolution / size Available options: ((‘-te’ or ‘-lle’) and
   (‘-tr’ or ‘-ts’)) (e.g. ‘-lle -10 30 55 60 -ts 1000 1000’ or ‘-te 100 2000 300 10000 -tr 300
   200’) -tr resolutionx resolutiony -ts sizex sizey -te xmin ymin xmax ymax -lle lonmin latmin
   lonmax latmax
   • ds (GDAL dataset) –
Examples

```python
>>> d = Domain(srs, ext)  # size, extent and spatial reference is given by strings
>>> d = Domain(ds=GDALDataset)  # size, extent copied from input GDAL dataset
>>> d = Domain(srs, ds=GDALDataset)  # spatial reference is given by srs, but size and extent is determined from input GDAL dataset
```

Notes

The core of Domain is a GDAL Dataset. It has no bands, but only georeference information: rasterXsize, rasterYsize, GeoTransform and Projection or GCPs, etc. which fully describe dimention and spatial reference of the grid.

There are three ways to store geo-reference in a GDAL dataset:

- Using GeoTransfrom to define linear relationship between raster pixel/line and geographical X/Y coordinates
- Using GCPs (set of Ground Control Points) to define non-linear relationship between pixel/line and X/Y
- Using Geolocation Array - full grids of X/Y coordinates for each pixel of a raster

The relation between X/Y coordinates of the raster and latitude/longitude coordinates is defined by projection type and projection parameters. These pieces of information are therefore stored in Domain:

- Type and parameters of projection + * GeoTransform, or * GCPs, or * GeolocationArrays

Domain has methods for basic operations with georeference information:

- creating georeference from input options;
- fetching corner, border or full grids of X/Y coordinates;
- making map of the georeferenced grid in a PNG or KML file;
- and some more...

The main attribute of Domain is a VRT object self.vrt. Nansat inherits from Domain and adds bands to self.vrt.

Raises

- `NansatProjectionError`: occurs when `Projection()` is empty – despite it is required for creating extentDic.
- `OptionError`: occurs when the arguments are not proper.

See also:


KML_BASE = '"<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2"
...
xmlns:kml="http://www.opengis.net/kml/2.2"
xmlns:atom="http://www.w3.org/2005/Atom">
{content}
</kml>"

OUTPUT_SEPARATOR = '----------------------------------------

azimuth_y (reductionFactor=1)

Calculate the angle of each pixel position vector with respect to the Y-axis (azimuth).

In general, azimuth is the angle from a reference vector (e.g., the direction to North) to the chosen position vector. The azimuth increases clockwise from direction to North. [http://en.wikipedia.org/wiki/Azimuth](http://en.wikipedia.org/wiki/Azimuth)
Parameters `reductionFactor` *(integer)* – factor by which the size of the output array is reduced

Returns `azimuth` – Values of azimuth in degrees in range 0 - 360

Return type `numpy array`

`contains` *(anotherDomain)*
Checks if this Domain fully covers another Domain

Returns `contains` – True if this Domain fully covers another Domain, False otherwise

Return type `bool`

`classmethod from_lonlat` *(lon, lat, add_gcps=True)*
Create Domain object from input longitudes, latitudes arrays

Parameters

- `lon` *(numpy.ndarray)* – longitudes
- `lat` *(numpy.ndarray)* – latitudes
- `add_gcps` *(bool)* – Add GCPs from lon/lat arrays.

Returns `d`

Return type `Domain`

**Examples**

```python
>>> lon, lat = np.meshgrid(range(10), range(10))
>>> d1 = Domain.from_lonlat(lon, lat)
>>> d2 = Domain.from_lonlat(lon, lat, add_gcps=False)  # add only geolocation

get_border *(nPoints=10, fix_lon=True, **kwargs)*
Generate two vectors with values of lat/lon for the border of domain

Parameters

- `nPoints` *(int, optional)* – Number of points on each border
- `fix_lon` *(bool)* – Convert longitudes to positive numbers when Domain crosses date-line?

Returns `lonVec, latVec` – vectors with lon/lat values for each point at the border

Return type `lists`

`get_border_geojson` *(*args, **kwargs)*
Create border of the Polygon in GeoJson format

Returns

Return type `str`, the Polygon border in GeoJson format

`get_border_geometry` *(*args, **kwargs)*
Get OGR Geometry of the border Polygon

Returns

Return type `OGR Geometry`, type Polygon
get_border_postgis(**kwargs)
Get PostGIS formatted string of the border Polygon

    Returns str
    Return type ‘PolygonFromText(PolygonWKT)’

get_border_wkt(*args, **kwargs)
Creates string with WKT representation of the border polygon

    Returns WKTPolygon – string with WKT representation of the border polygon
    Return type string

get_corners()
Get coordinates of corners of the Domain

    Returns lonVec, latVec – vectors with lon/lat values for each corner
    Return type lists

get_geolocation_grids(stepSize=1, dstSRS=<sphinx.ext.autodoc.importer._MockObject object>)
Get longitude and latitude grids representing the full data grid

If GEOLOCATION is not present in the self.vrt.dataset then grids are generated by converting pixel/line
of each pixel into lat/lon If GEOLOCATION is present in the self.vrt.dataset then grids are read from the
geolocation bands.

Parameters stepSize (int) – Reduction factor if output is desired on a reduced grid size

    Returns
    • longitude (numpy array) – grid with longitudes
    • latitude (numpy array) – grid with latitudes

get_min_max_lon_lat()
Get minimum and maximum of longitude and latitude geolocation grids

    Returns min_lon, max_lon, min_lat, max_lat, – min/max lon/lat values for the Domain
    Return type float

get_pixelsize_meters()
Returns the pixelsize (deltaX, deltaY) of the domain

For projected domains, the exact result which is constant over the domain is returned. For geographic
(lon-lat) projections, or domains with no geotransform, the haversine formula is used to calculate the pixel
size in the center of the domain.

    Returns delta_x, delta_y – pixel size in X and Y directions given in meters
    Return type float

intersects(anotherDomain)
Checks if this Domain intersects another Domain

    Returns intersects – True if Domains intersects, False otherwise
    Return type bool

logger = None

name = None

overlaps(anotherDomain)
Checks if this Domain overlaps another Domain
Returns overlaps – True if Domains overlaps, False otherwise

Return type bool

reproject_gcps(srs_string="")

Reproject all GCPs to a new spatial reference system

Necessary before warping an image if the given GCPs are in a coordinate system which has a singularity in (or near) the destination area (e.g. poles for lonlat GCPs)

Parameters srs_string (string) – SRS given as Proj4 string. If empty ‘+proj=stere’ is used

Notes

Reprojects all GCPs to new SRS and updates GCPProjection

shape()

Return Numpy-like shape of Domain object (ySize, xSize)

Returns shape – Numpy-like shape of Domain object (ySize, xSize)

Return type tuple of two INT

transform_points(colVector, rowVector, DstToSrc=0, dstSRS=<sphinx.ext.autodoc.importer._MockObject object>)

Transform given lists of X,Y coordinates into lon/lat or inverse

Parameters

• colVector (lists) – X and Y coordinates in pixel/line or lon/lat coordinate system

• DstToSrc (0 or 1) – 0 - forward transform (pix/line => lon/lat) 1 - inverse transformation

• dstSRS (NSR) – destination spatial reference

Returns X, Y – X and Y coordinates in lon/lat or pixel/line coordinate system

Return type lists

vr = None

write_kml(xmlFileName=None, kmlFileName=None)

Write KML file with domains

Convert XML-file with domains into KML-file for GoogleEarth or write KML-file with the current Domain

Parameters

• xmlFileName (string, optional) – Name of the XML-file to convert. If only this value is given - kmlFileName=xmlFileName+’.kml’

• kmlFileName (string, optional) – Name of the KML-file to generate from the current Domain

write_kml_image(kmlFileName, kmlFigureName=None)

Create KML file for already projected image

Write Domain Image into KML-file for GoogleEarth

Parameters

• kmlFileName (str) – Name of the KML-file to generate from the current Domain

• kmlFigureName (str) – Name of the projected image stored in .png format
Examples

```
>>> n.undo(100)  # cancel previous reprojection
>>> lons, lats = n.get_corners()  # Get corners of the image and the pixel resolution
>>> srsString = '+proj=latlong +datum=WGS84 +ellps=WGS84 +no_defs'
>>> extentString = '-lle %f %f %f %f -ts 3000 3000' % (min(lons), min(lats), max(lons), max(lats))
>>> d = Domain(srs=srsString, ext=extentString)  # Create Domain with stereographic projection, corner coordinates and resolution 1000m
>>> n.reproject(d)
>>> n.write_figure(filename=figureName, bands=[3], clim=[0,0.15], cmapName='gray', transparency=0)
>>> n.write_kml_image(kmlFileName=oPath + filename + '.kml', kmlFigureName=figureName)  # 6.
```

```
nansat.exceptions module

exception nansat.exceptions.NansatGDALError
    Bases: exceptions.Exception
    Error from GDAL

exception nansat.exceptions.NansatGeolocationError
    Bases: exceptions.Exception
    Exception if geolocation is wrong (e.g., all lat/lon values are 0)

exception nansat.exceptions.NansatMissingProjectionError
    Bases: exceptions.Exception
    Exception raised if no (sub-) dataset has projection

exception nansat.exceptions.NansatProjectionError
    Bases: exceptions.Exception
    Cannot get the projection

exception nansat.exceptions.NansatReadError
    Bases: exceptions.Exception
    Exception if a file cannot be read with Nansat

exception nansat.exceptions.WrongMapperError
    Bases: exceptions.Exception
    Error for handling data that does not fit a given mapper

nansat.exporter module

class nansat.exporter.Exporter
    Bases: object
    Abstract class for export functions

    DEFAULT_INSTITUTE = 'NERSC'
    DEFAULT_SOURCE = 'satellite remote sensing'
    UNWANTED_METADATA = ['dataType', 'SourceFilename', 'SourceBand', '_Unsigned', 'FillValue']
```
export (filename=", bands=None, rm_metadata=None, add_geolocation=True, driver='netCDF', options=None, hardcopy=False)

Export Nansat object into netCDF or GTiff file

Parameters

- **filename (str)** – output file name
- **bands (list (default=None))** – Specify band numbers to export. If None, all bands are exported.
- **rm_metadata (list)** – metadata names for removal before export. e.g. ['name', 'colorormap', 'source', 'sourceBands']
- **add_geolocation (bool)** – add geolocation array datasets to exported file?
- **driver (str)** – Name of GDAL driver (format)
- **options (str or list)** – GDAL export options in format of: ‘OPT=VAL’, or ['OPT1=VAL1', 'OP2=VAL2'] See also http://www.gdal.org/frmt_netcdf.html
- **hardcopy (bool)** – Evaluate all bands just before export?

Notes

If number of bands is more than one, serial numbers are added at the end of each band name. It is possible to fix it by changing line 4605 in GDAL/frmts/netcdf/netcdfdataset.cpp: ‘if( nBands > 1 ) sprintf(szBandName,"%s%d",tmpMetadata,iBand);’ -> ‘if( nBands > 1 ) sprintf(szBandName,"%s",tmpMetadata);’

CreateCopy fails in case the band name has special characters, e.g. the slash in ‘HH/VV’.

Metadata strings with special characters are escaped with XML/HTML encoding.

Examples

# export all the bands into a netDCF 3 file >>> n.export(netcdfile)
# export all bands into a GeoTiff >>> n.export(driver='GTiff')

export2thredds (filename, bands, metadata=None, mask_name=None, rm_metadata=None, time=None, created=None)

Export data into a netCDF formatted for THREDDS server

Parameters

- **filename (str)** – output file name
- **bands (dict)** –
  
  {'band_name': {'type': '>i1', 'scale': 0.1, 'offset': 1000, 'metaKey1': 'meta value 1', 'metaKey2': 'meta value 2'}}

  dictionary sets parameters for band creation 'type' - string representation of data type in the output band 'scale' - sets scale_factor and applies scaling 'offset' - sets 'scale_offset and applies offsetting other entries (e.g. 'units': 'K') set other metadata

3.1. nansat

19
• **metadata** (*dict*) – Global metadata to add

• **mask_name** (*str*) – If data include a mask band: give the mask name. Non-masked value is 64. If None: no mask is added

• **rm_metadata** (*list*) – Unwanted metadata names which will be removed

• **time** (*list with datetime objects*) – Acquisition time of original data. That value will be in time dim

• **created** (*datetime*) – Date of creation. Will be in metadata ‘created’

**Note:** Nansat object (self) has to be projected (with valid GeoTransform and valid Spatial reference information) but not with GCPs

### Examples

```python
# create THREDDS formatted netcdf file with all bands and time variable >>> n.export2thredds(filename)

# export only the first band and add global metadata >>> n.export2thredds(filename, ['L_469'], {'description': 'example'})

# export several bands and modify type, scale and offset >>> bands = {'L_645': {'type': '>i2', 'scale': 0.1, 'offset': 0},
                          'L_555': {'type': '>i2', 'scale': 0.1, 'offset': 0}}

>>> n.export2thredds(filename, bands)
```

### nansat.figure module

**class nansat.figure.Figure** (*nparray, **kwargs*)

Bases: object

Perform operations with graphical files: create, append legend, save.

Figure instance is created in the Nansat.write_figure method. The methods below are applied consequently in order to generate a figure from one or three bands, estimate min/max, apply logarithmic scaling, convert to uint8, append legend, save to a file

```python
CAPTION_LOCATION_X = 0.1
CAPTION_LOCATION_Y = 0.25
CBAR_HEIGHT = 0.15
CBAR_HEIGHTMIN = 5
CBAR_LOCATION_X = 0.1
CBAR_LOCATION_Y = 0.5
CBAR_WIDTH = 0.8
CBTICK_LOC_ADJUST_X = 5
CBTICK_LOC_ADJUST_Y = 3
DEFAULT_EXTENSION = '.png'
```
add_latlon_grids(**kwargs)
Add lat/lon grid lines into the PIL image

Parameters
- of Figure_init() parameters (Any) –
- latGrid (numpy array) – array with values of latitudes
- lonGrid (numpy array) – array with values of longitudes
- lonTicks (int or list) – number of lines to draw or locations of gridlines
- latTicks (int or list) – number of lines to draw or locations of gridlines

add_latlon_labels(**kwargs)
Add lat/lon labels along upper and left side

Parameters
- of Figure_init() parameters (Any) –
- latGrid (numpy array) – array with values of latitudes
- lonGrid (numpy array) – array with values of longitudes
- lonTicks (int or list) – number of lines to draw or locations of gridlines
- latTicks (int or list) – number of lines to draw or locations of gridlines

add_logo(**kwargs)
Insert logo into the PIL image

Parameters
- of Figure_init() parameters (Any) –
- Modifies –
- self.pilImg –
apply_logarithm(**kwargs)
Apply a tone curve to the array

After the normalization of the values from 0 to 1, logarithm is applied. Then the values are converted to the normal scale.

**Parameters**
- **of Figure.__init__() parameters (Any)** –
- **Modifies** –
- **---------** –
- **self.array** (**numpy array**) –

apply_mask(**kwargs)
Apply mask for coloring land, clouds, etc

If mask_array and mask_lut are provided as input parameters. The pixels in self.array which have index equal to mask_lut key in mask_array will have color equal to mask_lut value

apply_mask should be called only after convert_palettesize (i.e. to uint8 data)

**Parameters**
- **of Figure.__init__() parameters (Any)** –
- **Modifies** –
- **---------** –
- **self.array** (**numpy array**) –

array = None
caption = ''
clim_from_histogram(**kwargs)
Estimate min and max pixel values from histogram

if ratio=1.0, simply the minimum and maximum values are returned. if 0 < ratio < 1.0, get the histogram of the pixel values. Then get rid of (1.0-ratio)/2 from the both sides and return the minimum and maximum values.

**Parameters of Figure.__init__() parameters (Any)** –

Returns clim – minimum and maximum values for each band

**Return type** numpy array 2D ((3x2) or (1x2))

clip(**kwargs)
Convert self.array to values between cmin and cmax

if pixel value < cmin, replaced to cmin. if pixel value > cmax, replaced to cmax.

**Parameters**
- **of Figure.__init__() parameters (Any)** –
- **Modifies** –
- **---------** –
- **self.array** (**numpy array**) –
- **self.cmax** (**self.cmin**) –
cmapName = 'jet'
cmax = [1.0]
cmin = [0.0]

convert_pallettesize(**kwargs)
Convert self.array to palette color size in uint8

Parameters

- of Figure.__init__() parameters (Any) –
- Modifies –
- ---------- –
- self.array (numpy array (=>uint8)) –

create_legend(**kwargs)
self.legend is replaced from None to PIL image
PIL image includes colorbar, caption, and titleString.

Parameters

- of Figure.__init__() parameters (Any) –
- Modifies –
- ---------- –
- self.legend (PIL image) –

create_pilImage(**kwargs)
self.create_pilImage is replaced from None to PIL image
If three images are given, create a image with RGB mode. if self.pilImgLegend is not None, it is pasted.
If one image is given, create a image with P(palette) mode. if self.pilImgLegend is not None, self.array is extended before create the pilImag and then paste pilImgLegend onto it.

Parameters

- of Figure.__init__() parameters (Any) –
- Modifies –
- ---------- –
- self.pilImg (PIL image) – PIL image with / without the legend
- self.array (replace to None) –

extensionList = ['png', 'PNG', 'tif', 'TIF', 'bmp', 'BMP', 'jpg', 'JPG', 'jpeg', 'JPEG
fontRatio = 1
fontSize = None
gamma = 2.0
latGrid = None
latTicks = 5
legend = False
logarithm = False
logoFileName = None
logoLocation = [0, 0]
logoSize = None
lonGrid = None
lonTicks = 5
mask_array = None
mask_lut = None
numOfColor = 250
numOfTicks = 5
palette = None
pilImg = None
pilImgLegend = None

process(**kwargs)
    Do all common operations for preparation of a figure for saving
    1. Modify default values of parameters by the provided ones (if any)
    2. Clip to min/max
    3. Apply logarithm if required
    4. Convert data to uint8
    5. Create palette
    6. Apply mask for colouring land, clouds, etc if required
    7. Create legend if required
    8. Create PIL image
    9. Add logo if required

Parameters
    • of Figure.__init__() parameters (Any) –
      • Modifies –
        • -------- –
          • self.d –
          • self.array –
          • self.palette –
          • self.pilImgLegend –
          • self.pilImg –

ratio = 1.0

save(fileName, **kwargs)
    Save self.pilImg to a physical file
    If given extension is JPG, convert the image mode from Palette to RGB
Parameters

- **fileName** *(string)* – name of output file
- **of Figure.__init__() parameters** *(Any)* –
- **Modifies** –
- **--------** –
- **self.pilImg** *(None)* –

```python
subsetArraySize = 100000
titleString = ''
transparency = None
```

**nansat.geolocation module**

class nansat.geolocation.Geolocation(x_vrt, y_vrt, **kwargs)
    Bases: object

Container for GEOLOCATION data

Keeps references to bands with X and Y coordinates, offset and step of pixel and line. All information is stored in dictionary self.data

Instance of Geolocation is used in VRT and usually created in a Mapper.

data = None
classmethod from_dataset(dataset)
    Create geolocation from GDAL dataset :param dataset: input dataset to copy Geolocation metadata from :type dataset: gdal.Dataset
classmethod from_filenames(x_filename, y_filename, **kwargs)

get_geolocation_grids()
    Read values of geolocation grids

```python
x_vrt = None
y_vrt = None
```

**nansat.nansat module**

class nansat.nansat.Nansat(filename=", mapper="", log_level=30, **kwargs)
    Bases: nansat.domain.Domain, nansat.exporter.Exporter

Container for geospatial data. Performs all high-level operations.

```python
n = Nansat(filename) opens the file with satellite or model data for reading, adds scientific metadata to bands, and prepares the data for further handling.
```

Parameters

- **filename** *(str)* – uri of the input file or OPeNDAP data stream
- **mapper** *(str)* – name of the mapper from nansat/mappers dir. E.g. ‘sentinel1_l1’, ‘asar’, ‘hirlam’, ‘meris_l1’, ‘meris_l2’, etc.
• \texttt{log\_level (int)} – Level of logging. See: \url{http://docs.python.org/howto/logging.html}

• \texttt{kwargs (additional arguments for mappers)} –

\section*{Examples}

\begin{verbatim}
>>> n1 = Nansat(filename)
>>> n2 = Nansat(sentinel1_filename, mapper='sentinel1_l1')
>>> array1 = n1[1]
>>> array2 = n2['sigma0_HV']
\end{verbatim}

\section*{Notes}

The instance of Nansat class (the object \texttt{<n>}) contains information about geographical reference of the data (e.g. raster size, pixel resolution, type of projection, etc) and about bands with values of geophysical variables (e.g. water leaving radiance, normalized radar cross section, chlrophyll concentration, etc). The object \texttt{<n>} has methods for high-level operations with data. E.g.: * reading data from file (Nansat._getitem__); * visualization (Nansat.write\_figure); * changing geographical reference (Nansat.reproject); * exporting (Nansat.export) * and much more...

Nansat inherits from Domain (container of geo-reference information) Nansat uses instance of VRT (wrapper around GDAL VRT-files) Nansat uses instance of Figure (collection of methods for visualization)

\begin{verbatim}
ALT\_FILL\_VALUE = -10000.0
FILL\_VALUE = 9.96921e+36
\end{verbatim}

\texttt{add\_band (array, parameters=None, nomem=False)}

Add band from numpy array with metadata.

Create VRT object which contains VRT and RAW binary file and append it to self.vrt.band_vrts

\textbf{Parameters}

• \texttt{array (ndarray)} – new band data. Shape should be equal to shape

• \texttt{parameters (dict)} – band metadata: wkv, name, etc. (or for several bands)

• \texttt{nomem (bool)} – saves the vrt to a tempfile on disk?

\section*{Notes}

Creates VRT object with VRT-file and RAW-file. Adds band to the self.vrt.

\section*{Examples}

\begin{verbatim}
>>> n.add\_band(array, ('name': 'new\_data')) # add new band and metadata, keep...
←in memory
>>> n.add\_band(array, nomem=True) # add new band, keep on disk
\end{verbatim}

\texttt{add\_bands (arrays, parameters=None, nomem=False)}

Add bands from numpy arrays with metadata.

Create VRT object which contains VRT and RAW binary file and append it to self.vrt.band_vrts
Parameters

- **arrays** *(list of ndarrays)* – new band data. Shape should be equal to shape
- **parameters** *(list of dict)* – band metadata: wkv, name, etc. (or for several bands)
- **nomem** *(bool)* – saves the vrt to a tempfile on disk?

Notes

Creates VRT object with VRT-file and RAW-file. Adds band to the self.vrt.

Examples

```
>>> n.add_bands([array1, array2]) # add new bands, keep in memory
```

```python
bands()
```

Make a dictionary with all metadata from all bands

- **Returns** b – key = N, value = dict with all band metadata
- **Return type** dictionary

```python
crop(x_offset, y_offset, x_size, y_size, allow_larger=False)
```

Crop Nansat object

Create superVRT, modify the Source Rectangle (SrcRect) and Destination Rectangle (DstRect) tags in the VRT file for each band in order to take only part of the original image, create new GCPs or new GeoTransform for the cropped object.

- **Parameters**
  - **x_offset** *(int)* – pixel offset of subimage
  - **y_offset** *(int)* – line offset of subimage
  - **x_size** *(int)* – width in pixels of subimage
  - **y_size** *(int)* – height in pixels of subimage
  - **allow_larger** *(bool)* – Allow resulting extent to be larger than the original image?

Notes

self.vrt : super-VRT is created with modified SrcRect and DstRect

- **Returns** extent – x_offset - X offset in the original dataset y_offset - Y offset in the original dataset x_size - width of the new dataset y_size - height of the new dataset
- **Return type** (x_offset, y_offset, x_size, y_size)

Examples

```
>>> extent = n.crop(10, 20, 100, 200)
```
**crop_interactive** *(band=1, maxwidth=1000, **kwargs)*  
Interactively select boundary and crop Nansat object

**Parameters**

- **band** *(int or str)* – id of the band to show for interactive selection of boundaries
- **maxwidth** *(int)* – large input data is downscaled to <maxwidth>
- ****kwargs **(keyword arguments for imshow)**

**Notes**

self.vrt [VRT] superVRT is created with modified SrcRect and DstRect

**Returns**

extent – x_offset - X offset in the original dataset  
y_offset - Y offset in the original dataset  
x_size - width of the new dataset  
y_size - height of the new dataset

**Return type** *(x_offset, y_offset, x_size, y_size)*

**Examples**

```python
>>> extent = n.crop_interactive(band=1) # crop a subimage interactively
```

**crop_lonlat** *(lonlim, latlim)*

Crop Nansat object to fit into given longitude/latitude limit

**Parameters**

- **lonlim** *(list of 2 float)* – min/max of longitude  
- **latlim** *(list of 2 float)* – min/max of latitude

**Notes**

self.vrt [VRT] crops vrt to size that corresponds to lon/lat limits

**Returns**

extent – x_offset - X offset in the original dataset  
y_offset - Y offset in the original dataset  
x_size - width of the new dataset  
y_size - height of the new dataset

**Return type** *(x_offset, y_offset, x_size, y_size)*

**Examples**

```python
>>> extent = n.crop(lonlim=[-10,10], latlim=[-20,20]) # crop for given lon/lat limits
```

digitize_points *(band=1, **kwargs)*

Get coordinates of interactively digitized points

**Parameters**

- **band** *(int or str)* – ID of Nansat band  
- ****kwargs **(keyword arguments for imshow)**

28 Chapter 3. Packages and modules
Returns points – list of 2xN arrays of points to be used in Nansat.get_transect()

Return type: list

extend(left=0, right=0, top=0, bottom=0)
Extend domain from four sides

Parameters

• left (int) – number of pixels to add from left side
• right (int) – number of pixels to add from right side
• top (int) – number of pixels to add from top side
• bottom (int) – number of pixels to add from bottom side

Notes

Changes self.vrt by adding negative offset or setting size to be larger than original size.

filename = None

classmethod from_domain(domain, array=None, parameters=None, log_level=30)
Create Nansat object from input Domain [and array with data]

Parameters

• domain (Domain) – Defines spatial reference system and geographical extent.
• array (numpy NDarray) – Data for the first band. Shape must correspond to shape of <domain>
• parameters (dict) – Metadata for the first band. May contain ‘name’, ‘wkv’ and other keys.
• log_level (int) – Level of logging.

get_GDALRasterBand(band_id=1)
Get a GDALRasterBand of a given Nansat object
If str is given find corresponding band number. If int is given check if band with this number exists. Get a GDALRasterBand from vrt.

Parameters

• band_id (int or str) – if int - a band number of the band to fetch if str
  band_id = {'name': band_id}

Returns

Return type: GDAL RasterBand

Example

```python
>>> b = n.get_GDALRasterBand(1)
>>> b = n.get_GDALRasterBand('sigma0')
```

get_band_number(band_id)
Return absolute band number

Check if given band_id is valid. Return absolute number of the band in the VRT
Parameters **band_id** *(int or str or dict)* – if int: checks if such band exists and returns band_id if str: finds band with corresponding name if dict: finds first band with given metadata

Returns int

Return type absolute band number

**get_metadata** *(key=None, band_id=None, unescape=True)*

Get metadata from self.vrt.dataset

Parameters

- **key** *(str)* – name of the metadata key. If not given all metadata is returned
- **band_id** *(int or str)* – number or name of band to get metadata from. If not given, global metadata is returned
- **unescape** *(bool)* – Replace ‘&quot;’, ‘&amp;’ ‘&lt;’ and ‘&gt;’ with these symbols ” & < > ?

Returns

- * string with metadata if key is given and found
- * dictionary with all metadata if key is not given

Raises ValueError, if key is not found

**get_transect** *(points, bands, lonlat=True, smooth_radius=0, smooth_function=<sphinx.ext.autodoc.importer._MockObject object>, data=None, cornersonly=False)*

Get values from transect from given vector of points

Parameters

- **points** *(2xN list or array, N (number of points) >= 1)* – coordinates [[x1, x2, y2], [y1, y2, y3]]
- **bands** *(list of int or string)* – elements of the list are band number or band Name
- **lonlat** *(bool)* – If the points in lat/lon, then True. If the points in pixel/line, then False.
- **smooth_radius** *(int)* – If smoothRadius is greater than 0, smooth every transect pixel as the median or mean value in a circle with radius equal to the given number.
- **smooth_function** *(func)* – function for averaging values collected within smooth radius
- **data** *(ndarray)* – alternative array with data to take values from

Returns transect

Return type numpy record array

**has_band** *(band)*

Check if self has band with name <band> :param band: name or standard_name of the band to check :type band: str

Returns

Return type True/False if band exists or not

**list_bands** *(do_print=True)*

Show band information of the given Nansat object
Show serial number, longName, name and all parameters for each band in the metadata of the given Nansat object.

**Parameters**

- **do_print** *(boolean)* – print on screen?

**Returns**

- **outString** – formatted string with bands info

**Return type**

String

```python
logger = None
mapper = None
name = None
path = None
```

```python
reproject(dst_domain=None, resample_alg=0, block_size=None, tps=None, skip_gcps=1, addmask=True, **kwargs)
```

Change projection of the object based on the given Domain

Create superVRT from self.vrt with AutoCreateWarpedVRT() using projection from the dst_domain. Modify XML content of the warped vrt using the Domain parameters. Generate warpedVRT and replace self.vrt with warpedVRT. If current object spans from 0 to 360 and dst_domain is west of 0, the object is shifted by 180 westwards.

**Parameters**

- **dst_domain** *(domain)* – destination Domain where projection and resolution are set
- **resample_alg** *(int (GDALResampleAlg))* – 0 : NearestNeighbour 1 : Bilinear 2 : Cubic 3 : CubicSpline 4 : Lanczos
- **block_size** *(int)* – size of blocks for resampling. Large value decrease speed but increase accuracy at the edge
- **tps** *(bool)* – Apply Thin Spline Transformation if source or destination has GCPs Usage of TPS can also be triggered by setting self.vrt.tps=True before calling to reproject. This options has priority over self.vrt.tps
- **skip_gcps** *(int)* – Using TPS can be very slow if the number of GCPs are large. If this parameter is given, only every [skip_gcps] GCP is used, improving calculation time at the cost of accuracy. If not given explicitly, `skip_gcps` is fetched from the metadata of self, or from dst_domain (as set by mapper or user). [defaults to 1 if not specified, i.e. using all GCPs]
- **addmask** *(bool)* – If True, add band ‘swathmask’. 1 - valid data, 0 no-data. This band is used to replace no-data values with np.nan

**Notes**

- **self.vrt** : VRT object with dataset replaced to warpedVRT dataset Integer data is returned by integer. Round off to decimal place. If you do not want to round off, convert the data types to GDT_Float32, GDT_Float64, or GDT_CFloat32.

See also:

- **http()** //www.gdal.org/gdalwarp.html

```python
resize(factor=None, width=None, height=None, pixelsize=None, resample_alg=-1)
```

Proportional resize of the dataset.
The dataset is resized as \( (x_{\text{size}} \times \text{factor}, y_{\text{size}} \times \text{factor}) \). If desired width, height or pixelsize is specified, the scaling factor is calculated accordingly. If GCPs are given in a dataset, they are also rewritten.

**Parameters**

- **factor** (float, optional, default=1) – Scaling factor for width and height > 1 means increasing domain size < 1 means decreasing domain size
- **width** (int, optional) – Desired new width in pixels
- **height** (int, optional) – Desired new height in pixels
- **pixelsize** (float, optional) – Desired new pixelsize in meters (approximate). A factor is calculated from ratio of the current pixelsize to the desired pixelsize.
- **resample_alg** (int (GDALResampleAlg), optional) –
  -1 [Average (default),] 0 : NearestNeighbour 1 : Bilinear, 2 : Cubic, 3 : CubicSpline, 4 : Lancoz

**Notes**

- **self.vrt.dataset** [VRT dataset of VRT object] raster size are modified to downscaled size. If GCPs are given in the dataset, they are also overwritten.

  - **set_metadata** (key=", value=", band_id=None)
  - Set metadata to self.vrt.dataset

  - **Parameters**
  - key (string or dictionary with strings) – name of the metadata, or dictionary with metadata names, values
  - value (string) – value of metadata
  - band_id (int or str) – number or name of band Without : global metadata is set

**Notes**

- **self.vrt.dataset** : sets metadata in GDAL current dataset

**time_coverage_end**

**time_coverage_start**

**undo** (steps=1)

- Undo reproject, resize, add_band or crop of Nansat object

  - Restore the self.vrt from self.vrt.vrt

  - **Parameters** steps (int) – How many steps back to undo

**Notes**

- Modifies self.vrt

**vrt = None**
**watermask** *(mod44path=None, dst_domain=None, **kwargs)*
Create numpy array with watermask (water=1, land=0)

250 meters resolution watermask from MODIS 44W Product: [http://www.glcf.umd.edu/data/watermask/](http://www.glcf.umd.edu/data/watermask/)

Watermask is stored as tiles in TIF(LZW) format and a VRT file. All files are stored in one directory. A tarball with compressed TIF and VRT files should be additionally downloaded from the Nansat documentation page: [http://nansat.readthedocs.io/en/latest/source/features.html#differentiating-between-land-and-water](http://nansat.readthedocs.io/en/latest/source/features.html#differentiating-between-land-and-water)

**The method:** Gets the directory either from input parameter or from environment variable MOD44WPATH. Open Nansat object from the VRT file. Reprojects the watermask onto the current object using reproject() or reproject_on_jcps(). Returns the reprojected Nansat object.

**Parameters**

- **mod44path (string)** – path with MOD44W Products and a VRT file
- **dst_domain (Domain)** – destination domain other than self
- **tps (Bool)** – Use Thin Spline Transformation in reprojection of watermask? See also Nansat.reproject()
- **skip_gcps (int)** – Factor to reduce the number of GCPs by and increase speed. See also Nansat.reproject()

**Returns** watermask

**Return type** Nansat object with water mask in current projection

See also:

- [http://www.glcf.umd.edu/data/watermask/](http://www.glcf.umd.edu/data/watermask/)

**write_figure**(filename=", bands=1, clim=None, addDate=False, array_modfunc=None, **kwargs)
Save a raster band to a figure in graphical format.

Get numpy array from the band(s) and band information specified either by given band number or band id.
– If three bands are given, merge them and create PIL image. – If one band is given, create indexed image
Create Figure object and:
- Adjust the array brightness and contrast using the given min/max or histogram.
- Apply logarithmic scaling of color tone.
- Generate and append legend. Save the PIL output image in PNG or any other graphical format. If the filename extension is ‘tif’, the figure file is converted to GeoTiff

**Parameters**

- **bands** (integer or string or list (elements are integer or string)), default = 1 the size of the list has to be 1 or 3. if the size is 3, RGB image is created based on the three bands. Then the first element is Red, the second is Green, and the third is Blue.
- **clim** (list with two elements or ‘hist’ to specify range of colormap) – None (default): min/max values are fetched from WKV, fallback-’hist’ [min, max]: min and max are numbers, or [[min, min, min], [max, max, max]]: three bands used ‘hist’: a histogram is used to calculate min and max values

---

3.1. nansat
• **addDate** *(boolean)* – False (default): no date will be added to the caption True: the first time of the object will be added to the caption

• **array_modfunc** *(None)* – None (default): figure created using array in provided band function: figure created using array modified by provided function

• **kwargs** *(parameters for Figure())* –

**Notes**

if filename is specified, creates image file

**Returns**

**Return type** Figure object

**Example**

```python
>>> n.write_figure('test.jpg')  # write indexed image
>>> n.write_figure('test_rgb_hist.jpg', clim='hist', bands=[1, 2, 3])  # RGB image
>>> n.write_figure('r09_log3_leg.jpg', logarithm=True, legend=True, gamma=3, titleString='Title', fontSize=30, numOfTicks=15)  # add legend
>>> n.write_figure(filename='transparent.png', bands=[3], mask_array=wmArray, mask_lut={0: [0,0,0]}, clim=[0,0.15], cmapName='gray', transparency=[0,0,0])  # write transparent image
```

**See also:**
Figure()

http() //www.scipy.org/Cookbook/Matplotlib/Show_colormaps

**write_geotiffimage** *(filename, band_id=1)*

Writes an 8-bit GeoTiff image for a given band.

The colormap is fetched from the metadata item ‘colormap’. Fallback colormap is ‘gray’.

Color limits are fetched from the metadata item ‘minmax’. If ‘minmax’ is not specified, min and max of the raster data is used.

The method can be replaced by using nansat.write_figure(). However, write_figure uses PIL, which does not allow Tiff compression. This gives much larger files.

**Parameters**

• **filename**(str)

• **band_id**(int or str)

**nansat.node module**

**class** nansat.node.Node *(tag, value=None, **attributes)*

**Bases:** object

Rapidly assemble XML using minimal coding.
Everything is a Node, and each Node can either have a value or subnodes. Subnodes can be appended to Nodes using `+=`, and a group of Nodes can be strung together using `+`.

Create a node containing a value by saying `Node('tag', 'value')`. You can also give attributes to the node in the constructor: `Node('tag', 'value', attr1 = 'attr1', attr2 = 'attr2')` or without a value: `Node('tag', attr1 = 'attr1', attr2 = 'attr2')`.

To produce xml from a finished Node n, say `n.xml()` (for nicely formatted output) or `n.rawxml()`.

You can read and modify the attributes of an xml Node using `getAttribute()`, `setAttribute()`, or `delAttribute()`.

You can find the value of the first subnode with tag == `tag` by saying `n['tag']`. If there are multiple instances of `n['tag']`, this will only find the first one, so you should use `node()` or `nodeList()` to narrow your search down to a Node that only has one instance of `n['tag']`.

You can replace the value of the first subnode with tag == `tag` by saying `n['tag'] = newValue`. The same issues exist as noted in the above paragraph.

You can find the first node with tag == `tag` by saying `node('tag')`. If there are multiple nodes with the same tag at the same level, use `nodeList('tag')`.

The Node class is also designed to create a kind of ‘domain specific language’ by subclassing Node to create Node types specific to your problem domain.

This implementation uses xml.dom.minidom which is available in the standard Python 2.4 library. However, it can be retargeted to use other XML libraries without much effort.

```python
static create(dom)  
    Create a Node representation, given either a string representation of an XML doc, or a dom.

delAttribute(name)  
    Remove XML attribute with this name.

delNode(tag, options=None)  
    Recursively find nodes containing subnodes with this tag and remove subnodes
    
    options [dictionary] if there are several tags, specify a node by their attributes.

doc = <xml.dom.minidom.Document instance>

dom()  
    Lazily create a minidom from the information stored in this Node object.

getAttribute(name)  
    Read XML attribute of this node.

getAttributeList()  
    get attributes and valuse from the node and return their lists

insert(contents)  
    return Node of the node with inserted <contents>

node(tag, elemNum=0)  
    Recursively find the first subnode with this tag.
    
    elemNum [int] if there are several same tag, specify which element to take.

nodeList(tag)  
    Produce a list of subnodes with the same tag. Note: It only makes sense to do this for the immediate children of a node. If you went another level down, the results would be ambiguous, so the user must choose the node to iterate over.
```

3.1. nansat
rawxml()
replaceAttribute(name, value)
    replace XML attribute of this node.
replaceNode(tag, elemNum=0, newNode=None)
    Find the first subnode with this tag and replace with given node.
    tag [str] node tag
    elemNum [int] number of subnode among other subnodes with similar tag
replaceTag(oldTag, newTag)
    Replace tag name
setAttribute(name, item)
    Modify XML attribute of this node.
tagList()
    Produce a list of all tags of the immediate children
xml(separator=' ')  

nansat.nsr module

nansat.pointbrowser module

class nansat.pointbrowser.PointBrowser(data, fmt='x-k', force_interactive=True, **kwargs)
    Click on raster images shown by plt.imshow and get the X-Y coordinates.

Parameters

- data (ndarray) – image to imshow
- transect (bool) – if True, get transects / points if False, get only points
- force_interactive (bool) – force PointBrowser to interactive mode? (True for regular use, False for tests)
- **kwargs (dict) – optional parameters for imshow

Note: self.fig : pyplot Figure self.data : ndarray with data self.ax : axes self.points : plot with points self.line : plot with points self.coordinates: container for recorded coordinates

ax = None
coordinates = None
data = None
fig = None
fmt = None
get_points()
    Enables the onclick events and returns the points.
    The format of the returned array: array([[x1,...,xn],[y1,...,yn]]),array([[xn+1,...],[yn+1,...]])... Each ‘array’ element is a numpy.ndarray and represents one transect, where x1,y1 is the first point in the first transect, and xn,yn the last point in the first transect. The inner x/y-arrays are also numpy.ndarrays
lines = None
**onclick** *(event)*

Process mouse onclick event Append coordinates of the click to self.coordinates, add point and 2D line to self.points If click is outside, nothing is done If click with ‘z’ pressed, nothing is done If click with ‘anykey’, new line is started

**Parameters**

- **event** *(matplotlib.mouse_event)*

  points = None
text_ax = None

---

**nansat.tools module**

**nansat.tools.add_logger** *(logName='', logLevel=None)*

Creates and returns logger with default formatting for Nansat

**Parameters**

- **logName** *(string, optional)* – Name of the logger

  Returns

  Return type  *logging.logger*

  See also: [http://docs.python.org/howto/logging.html](http://docs.python.org/howto/logging.html)

**nansat.tools.get_random_color** *(c0=None, minDist=100, low=0, high=255)*

Create random color which is far enough from the input color

**Parameters**

- **c0** *(str)* – hexademical representation of the color (e.g. ‘#ff0000’ for red)
- **minDist** *(int)* – minimal distance to input color

  Returns **c0** – hexademical representation of the new random color

  Return type  *str*

**nansat.tools.haversine** *(lon1, lat1, lon2, lat2)*

Calculate the great circle distance between two points on the spherical earth (specified in decimal degrees)

**nansat.tools.initial_bearing** *(lon1, lat1, lon2, lat2)*

Initial bearing when traversing from point1 (lon1, lat1) to point2 (lon2, lat2)

See [http://www.movable-type.co.uk/scripts/latlong.html](http://www.movable-type.co.uk/scripts/latlong.html)

**Parameters**

- **lat1** *(lon1,)* – longitude and latitude of start point
- **lat2** *(lon2,)* – longitude and latitude of end point

  Returns **initial_bearing** – The initial bearing (azimuth direction) when heading out from the start point towards the end point along a great circle.

  Return type  *float*

**nansat.tools.parse_time** *(time_string)*

Parse time string accounting for possible wrong formatting

**Parameters**

- **time_string** *(string with date and time)*

  Returns **time_value**

  Return type  *datetime object*

---

3.1. nansat
nansat.tools.register_colormaps()
Create custom colormaps and register them

nansat.tools.remove_keys(dict, keys)

nansat.tools.test_openable(fname)

nansat.tools.write_domain_map(border, out_filename, lon_vec=None, lat_vec=None, lon_border=10.0, lat_border=10.0, figure_size=(6,6), dpi=50, projection='cyl', resolution='c', continents_color='coral', meridians=10, parallels=10, p_color='r', p_line='k', p_alpha=0.5, padding=0.0, mer_labels=[False, False, False, False], par_labels=[False, False, False, False], pltshow=False, labels=None)

Create an image with a map of the domain

Uses Basemap to create a World Map Adds a semitransparent patch with outline of the Domain Writes to an image file

Parameters

- **out_filename** (string) – name of the output file name
- **lon_vec** ([floats] or [[floats]]) – longitudes of patches to display
- **lat_vec** ([floats] or [[floats]]) – latitudes of patches to display
- **lon_border** (float) – 10, horizontal border around patch (degrees of longitude)
- **lat_border** (float) – 10, vertical border around patch (degrees of latitude)
- **figure_size** (tuple of two integers) – (6, 6), size of the generated figure in inches
- **dpi** (int) – 50, resolution of the output figure (size 6,6 and dpi 50 produces 300 x 300 figure)
- **projection** (string, one of Basemap projections) – ‘cyl’, projection of the map
- **resolution** (string, resolution of the map) – ‘c’, crude ‘l’, low ‘i’, intermediate ‘h’, high ‘f’, full
- **continents_color** (string or any matplotlib color representation) – ‘coral’, color of continents
- **meridians** (int) – 10, number of meridians to draw
- **parallels** (int) – 10, number of parallels to draw
- **p_color** (string or any matplotlib color representation) – ‘r’, color of the Domain patch
- **p_line** (string or any matplotlib color representation) – ‘k’, color of the Domain outline
- **p_alpha** (float 0 - 1) – 0.5, transparency of Domain patch
- **padding** (float) – 0., width of white padding around the map
- **mer_labels** (list of 4 booleans) – where to put meridian labels, see also Basemap.drawmeridians()
- **par_labels** (list of 4 booleans) – where to put parallel labels, see also Basemap.drawparallels()
• **labels** (*list of str*) – labels to print on top of patches

**nansat.vrt module**

```
class nansat.vrt.VRT(x_size=1, y_size=1, metadata=None, nomem=False, **kwargs)
    Bases: object

Wrapper around GDAL VRT-file

The GDAL VRT-file is an XML-file. It contains all metadata, geo-reference information and information
ABOUT each band including band metadata, reference to the bands in the source file. VRT-class performs
all operation on VRT-files: create, copy, modify, read, write, add band, add GeoTransform, set Projection, etc. It
uses either GDAL methods for these operations (e.g. Create, AddBand, SetMetadata, AutoCreateWarpedVRT,
etc.) or reads/writes the XML-file directly (e.g. remove_geotransform, get_warped_vrt, etc).

The core of the VRT object is GDAL dataset `<self.dataset>` generated by the GDAL VRT-Driver. The respective
VRT-file is located in `/vismem` and has a random name.

GDAL data model doesn’t have place for geolocaion arrays therefore VRT-object has instance of Geolocation
(self.geolocation) an object to keep information about Geolocation metadata: reference to file with source data,
pixel and line step and offset, etc.

Domain has an instance of VRT-class `<self.vrt>`. It keeps only geo-reference information.

All Mappers inherit from VRT. When Nansat opens a file it loops through list of mappers, selects the one
appropriate for the input file, and creates an instance of Mapper.

Nansat has one instances of Mapper-class (>=VRT-class): self.vrt. It holds VRT-file in original projection
(derived from the input file). After most of the operations with Nansat object (e.g. reproject, crop, resize,
add_band) self.vrt is replaced with a new VRT object which has reference to the previous VRT object inside
(self.vrt.vrt).

Parameters

• **x_size** (*int*) – width of self.dataset
• **y_size** (*int*) – arguments for VRT()
• **metadata** (*dict*) – dictionary with metadata keys (str) and values (str)
• **nomem** (*bool*) – don’t create VRT in VSI memory?

Notes

adds self.logger, self.driver, self.filename, self.band_vrts, self.tps, self.vrt adds self.dataset - GDAL Dataset
without bands and with size=(x_zie, y_size) adds metadata to self.dataset writes VRT file content to self.filename

```
it’s contents is also copied into sub-VRT of the copy. Other attributes of self, such as tps flag and band_vrts are also copied.

classmethod `copy_dataset` *(gdal_dataset, **kwargs)*
Create VRT with bands and georeference as a full copy of input GDAL Dataset

**Parameters**
- `gdal_dataset` *(GDAL.Dataset)* – input dataset
- `**kwargs` *(dict)* – arguments for VRT()

**Returns** `vrt`
**Return type** `VRT`

`copyproj` *(filename)*
Copy geolocation data from given VRT to a figure file
Useful for adding geolocation information to figure files produced e.g. by Figure class, which contain no geolocation. Analogue to utility gdalcopyproj.py.

**Parameters** `filename` *(string)* – Name of file to which the geolocation data shall be written

`create_band` *(src, dst=None)*
Add band to self.dataset:
Get parameters of the source band(s) from input Generate source XML for the VRT, add options of creating Call GDALDataset.AddBand Set source and options Add metadata

**Parameters**
- `src` *(dict with parameters of sources:)* – SourceFilename SourceBand ScaleRatio ScaleOffset NODATA LUT SourceType DataType ImageOffset (RawVRT) PixelOffset (RawVRT) LineOffset (RawVRT) ByteOrder (RawVRT) xSize ySize
- `dst` *(dict with parameters of the created band)* – name dataType wkv suffix AnyOtherMetadata PixelFunctionType: - band will be a pixel function defined by the corresponding name/value. In this case src may be list of dicts with parameters for each source.

– in case the dst band has a different datatype than the source band it is important to add a SourceTransferType parameter in dst

**Returns** `name`
**Return type** `string`, name of the added band

**Examples**

```python
vrt.create_band({'SourceFilename': filename, 'SourceBand': 1})
vrt.create_band({'SourceFilename': filename, 'SourceBand': 2,
    'ScaleRatio': 0.0001},
    {'name': 'LAT', 'wkv': 'latitude'})
vrt.create_band({'SourceFilename': filename, 'SourceBand': 2})
```
create_bands(metadata_dict)
Generic function called from the mappers to create bands in the VRT dataset from an input dictionary of metadata

Parameters metadata_dict (list of dict with params of input bands and generated bands.) –

Each dict has: 'src': dictionary with parameters of the sources: 'dst': dictionary with parameters of the generated bands

Notes

Adds bands to the self.dataset based on info in metaDict

See also:
VRT.create_band()

create_geolocation_bands()
Create bands from Geolocation
dataset = None
delete_band(band_num)
Delete a band from the given VRT

Parameters band_num (int) – band number
delete_bands(band_nums)
Delete bands

Parameters bandNums (list) – elements are int
driver = None
export(filename)
Export VRT file as XML into given <filename>

filename = u''
fix_band_metadata(rm_metadata)
Add NETCDF_VARNAME and remove <rm_metadata> in metadata for each band

fix_global_metadata(rm_metadata)
Remove unwanted global metadata and escape special characters
classmethod from_array(array, **kwargs)
Create VRT from numpy array with dataset with one band but without georeference.

Parameters

- array (numpy.ndarray) – array with data
- **kwargs (dict) – arguments for VRT()

Returns vrt
Return type `VRT`

classmethod `from_dataset_params`(*x_size*, *y_size*, *geo_transform*, *projection*, *gcps*, *gcp_projection*, **kwargs)

Create VRT from GDAL Dataset parameters

Create VRT with dataset without bands but with size/georeference corresponding to input parameters.

Parameters

- *x_size* (int) – X-size of dataset
- *y_size* (int) – Y-size of dataset
- *geo_transform* (tuple with 6 floats) – information on affine transformation
- *projection* (str) – WKT representation of spatial reference system
- *gcps* (tuple or list with GDAL GCP objects) – GDAL Ground Control Points
- *gcp_projection* (str) – WKT representation of GCPs spatial reference system
- **kwargs*(dict) – arguments for VRT()

Returns vrt

Return type `VRT`

classmethod `from_gdal_dataset`(*gdal_dataset*, **kwargs)

Create VRT from GDAL Dataset with the same size/georeference but without bands.

Parameters

- *gdal_dataset* (gdal.Dataset) – input GDAL dataset
- **kwargs*(dict) – arguments for VRT()

Returns vrt

Return type `VRT`

classmethod `from_lonlat`(*lon*, *lat*, *add_gcps=True*, **kwargs)

Create VRT from longitude, latitude arrays

Create VRT with dataset without bands but with GEOLOCATION metadata and Geolocation object. Geolocation contains 2 2D arrays with lon/lat values given at regular pixel/line steps. GCPs can be created from lon/lat arrays and added to the dataset

Parameters

- *lon* (numpy.ndarray) – array with longitudes
- *lat* (numpy.ndarray) – array with latitudes
- *add_gcps* (bool) – Create GCPs from lon/lat arrays and add to dataset
- **kwargs*(dict) – arguments for VRT()

Returns vrt

Return type `VRT`

geolocation = None
get_projection()  
Get projection (spatial reference system) of the dataset  
Uses dataset.GetProjection() or dataset.GetGCPProjection() or dataset.GetMetadata('GEOLOCATION')['SRS']  

Returns

• projection (projection WKT)
• source (str ['gcps' or 'dataset' or 'geolocation'])

Raises NansatProjectionError: occurs when the projection is empty.

get_resized_vrt(x_size, y_size, resample_alg=1)
Resize VRT  
Create Warped VRT with modified RasterXSize, RasterYSize, GeoTransform. The returned VRT object has a copy of its original source VRT in its own vrt object (e.g. warpedVRT.vrt = originalVRT.copy()).

Parameters

• y_size(x_size,) – new size of the VRT object
• resample_alg(GDALResampleAlg) – see also gdal.AutoCreateWarpedVRT

Returns warped_vrt

Return type Resized VRT object

get_shifted_vrt(shift_degree)
Roll data in bands westwards or eastwards  
Create shift_vrt which references self. Modify georeference of shift_vrt to account for the roll. Add as many bands as in self but for each band create two complex sources: for western and eastern parts. Keep self in shift_vrt.vrt

Parameters shift_degree(float) – rolling angle, how far east/west to roll

Returns shift_vrt

Return type VRT object with rolled bands

get_sub_vrt(steps=1)
Returns sub-VRT from given depth  
Iteratively copy self.vrt into self until self.vrt is None or steps == 0

Parameters steps(int) – How many sub VRTs to restore

Returns

• self (if no deeper VRTs found)
• self.vrt (if deeper VRTs are found)

Notes

self self.vrt

get_subsampled_vrt(new_raster_x_size, new_raster_y_size, resample_alg)
Create VRT and replace step in the source

get_super_vrt()  
Create a new VRT object with a reference to the current object (self)
Create a new VRT (super_vrt) with exactly the same structure (number of bands, raster size, metadata) as the current object (self). Create a copy of the current object and add it as an attribute of the new object (super_vrt.vrt). Bands in the new object will refer to the same bands in the current object. Recursively copy all vrt attributes of the current object (self.vrt.vrt.vrt...) into the new object (super_vrt.vrt.vrt.vrt.vrt...).

Returns super_vrt – a new VRT object with copy of self in super_vrt.vrt

Return type VRT

get_warped_vrt dst_srs, x_size, y_size, geo_transform, resample_alg=0, dst_gcps=[], skip_gcps=1, block_size=None, working_data_type=None, resize_only=False)

Create warped (reprojected) VRT object

1. Create a simple warped dataset using GDAL.AutoCreateWarpedVRT.
2. Create VRT from the warped dataset.
3. Modify the warped VRT according to the input options (size, geotransform, GCPs, etc)
4. Keep the original VRT in the attribute vrt

For reprojecting the function tries to use geolocation by default, if geolocation is not present it tries to use GCPs, if GCPs are not present it uses GeoTransform.

If destination image has GCPs (provided in <dst_gcps>): fake GCPs for referencing line/pixel of SRC image and X/Y of DST image are created and added to the SRC image. After warping the dst_gcps are added to the warped VRT.

Parameters

• dst_srs (string) – WKT of the destination projection
• x_size, y_size – dimension of the destination raster
• geo_transform (tuple with 6 floats) – destination GDALGeotransform
• resample_alg (int (GDALResampleAlg)) – 0: NearestNeighbour, 1: Bilinear, 2: Cubic, 3: CubicSpline, 4: Lanczos
• dst_gcps (list with GDAL GCPs) – GCPs of the destination image
• skip_gcps (int) – Using TPS can be very slow if the number of GCPs are large. If this parameter is given, only every [skip_gcp] GCP is used, improving calculation time at the cost of accuracy.
• block_size (int) – BlockSize to use for resampling. Larger blocksize reduces speed but increases accuracy.
• working_data_type (str) – ‘Float32’, ‘Int16’, etc.
• resize_only (bool) – Create warped_vrt which will be used for resizing only?

Returns warped_vrt

Return type VRT object with warped dataset and with vrt

hardcopy_bands()

Make ‘hardcopy’ of bands: evaluate array from band and put into original band

leave_few_bands(bands=None)

Leave only given bands in VRT

logger = None

prepare_export_gtiff()

Prepare dataset for export using GTiff driver
**prepare_export_netcdf()**

Prepare dataset for export using netCDF driver

**static read_vsi(filename)**

Read text from input `<filename:str>` using VSI and return `<content:str>`.

**reproject_gcps(dst_srs)**

Reproject all GCPs to a new spatial reference system

Necessary before warping an image if the given GCPs are in a coordinate system which has a singularity in (or near) the destination area (e.g. poles for lonlat GCPs)

**Parameters**

- **dst_srs** (`proj4`, `WKT`, `NSR`, `EPSG`) – Destination SRS given as any NSR input parameter

**Notes**

Reprojects all GCPs to new SRS and updates GCPProjection

**set_offset_size(axis, offset, size)**

Set offset and size in VRT dataset and band attributes

**Parameters**

- **axis** (`str`) – name of axis (‘x’ or ‘y’)
- **offset** (`int`) – value of offset to put into VRT
- **size** (`int`) – value of size to put into VRT

**Notes**

Changes VRT file, sets new offset and size

**shift_cropped_gcps(x_offset, x_size, y_offset, y_size)**

Modify GCPs to fit the size/offset of cropped image

**shift_cropped_geo_transform(x_offset, x_size, y_offset, y_size)**

Modify GeoTransform to fit the size/offset of the cropped image

**split_complex_bands()**

Recursively find complex bands and relace by real and imag components

**tps = None**

**static transform_coordinates(src_srs, src_points, dst_srs)**

Transform coordinates of points from one spatial reference to another

**Parameters**

- **src_srs** (`nansat.NSR`) – Source spatial reference system
- **src_points** (tuple of two or three N-D arrays) – Coordinates of points in the source spatial reference system. A tuple with (X, Y) or (X, Y, Z) coordinates arrays. Each coordinate Each array can be a list, 1D, 2D, N-D array.
- **dst_srs** (`nansat.NSR`) – Destination spatial reference

**Returns**

- **dst_points** – Coordinates of points in the destination spatial reference system. A tuple with (X, Y) or (X, Y, Z) coordinates arrays. Each coordinate Each array can be 1D, 2D, N-D array. Shape of output arrays correspond to shape of inputs.
**Return type** tuple of two or three N-D arrays

**transform_points** *(col_vector, row_vector, dst2src=0, dst_srs=<sphinx.ext.autodoc.importer._MockObject object>, dst_ds=None, options=None)*

Transform input pixel/line coordinates into lon/lat (or opposite)

**Parameters**
- **row_vector** *(col_vector,)* – X and Y coordinates with any coordinate system
- **dst2src** *(0 or 1)* – 1 for inverse transformation, 0 for forward transformation.
- **dst_srs** *(NSR)* – destination SRS.
- **dst_ds** *(GDAL Dataset)* – destination dataset. The default is None. It means transform ownPixLin <–> ownXY.
- **option** *(string)* – if ‘METHOD=GEOLOC_ARRAY’, specify here.

**Returns** lon_vector, lat_vector – X and Y coordinates in degree of lat/lon

**Return type** numpy arrays

vrt = None

**write_xml** *(vsi_file_content=None)*

Write XML content into a VRT dataset

**Parameters**
- **vsi_file_content** *(string, optional)* – XML Content of the VSI file to write

**Notes**

- **self.dataset** If XML content was written, self.dataset is re-opened

xml

Read XML content of the VRT-file using VSI

**Returns** string

**Return type** XML Content which is read from the VSI file

**nansat.warnings module**

**exception** nansat.warnings.NansatFutureWarning

Bases: exceptions.Warning

**Module contents**

nansat.NSR

Used by autodoc_mock_imports.

**class nansat.Domain** *(srs=None, ext=None, ds=None, **kwarg)_

Bases: object

Container for geographical reference of a raster

A Domain object describes all attributes of geographical reference of a raster:
- width and height (number of pixels)
• pixel size (e.g. in decimal degrees or in meters)
• relation between pixel/line coordinates and geographical coordinates (e.g. a linear relation)
• type of data projection (e.g. geographical or stereographic)

Parameters

- **srs** ([PROJ4 or EPSG or WKT or NSR or osr.SpatialReference()]) – Input parameter for nansat.NSR()
- **ext** (string) – some gdalwarp options + additional options [http://www.gdal.org/gdalwarp.html] Specifies extent, resolution / size Available options: ('-te' or '-lle') and ('-tr' or '-ts') (e.g. '-lle -10 30 55 60 -ts 1000 1000' or '-te 100 2000 300 10000 -tr 300 200') -tr resolutionx resolutiony -ts sizex sizey -te xmin ymin xmax ymax -lle lonmin lonmax latmin latmax
- **ds** (GDAL dataset) –

Examples

```python
>>> d = Domain(srs, ext) #size, extent and spatial reference is given by strings
>>> d = Domain(ds=GDALDataset) #size, extent copied from input GDAL dataset
>>> d = Domain(srs, ds=GDALDataset) # spatial reference is given by srs, but size and extent is determined from input GDAL dataset
```

Notes

The core of Domain is a GDAL Dataset. It has no bands, but only georeference information: rasterXsize, rasterYsize, GeoTransform and Projection or GCPs, etc. which fully describe dimensions and spatial reference of the grid.

There are three ways to store geo-reference in a GDAL dataset:

- Using GeoTransform to define linear relationship between raster pixel/line and geographical X/Y coordinates
- Using GCPs (set of Ground Control Points) to define non-linear relationship between pixel/line and X/Y
- Using Geolocation Array - full grids of X/Y coordinates for each pixel of a raster

The relation between X/Y coordinates of the raster and latitude/longitude coordinates is defined by projection type and projection parameters. These pieces of information are therefore stored in Domain:

- Type and parameters of projection + * GeoTransform, or * GCPs, or * GeolocationArrays

Domain has methods for basic operations with georeference information:

- creating georeference from input options;
- fetching corner, border or full grids of X/Y coordinates;
- making map of the georeferenced grid in a PNG or KML file;
- and some more...

The main attribute of Domain is a VRT object self.vrt. Nansat inherits from Domain and adds bands to self.vrt
Nansat Documentation

Raises

- NansatProjectionError : occurs when Projection() is empty – despite it is required for creating extentDic.
- OptionError : occurs when the arguments are not proper.

See also:


KML_BASE = '<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2"
... xmlns:kml="http://www.opengis.net/kml/2.2"
xmlns:atom="http://www.w3.org/2005/Atom">
{content}
</kml>

OUTPUT_SEPARATOR = '----------------------------------------

azimuth_y (reductionFactor=1)

Calculate the angle of each pixel position vector with respect to the Y-axis (azimuth).

In general, azimuth is the angle from a reference vector (e.g., the direction to North) to the chosen position vector. The azimuth increases clockwise from direction to North. http://en.wikipedia.org/wiki/Azimuth

Parameters

reductionFactor (integer) – factor by which the size of the output array is reduced

Returns azimuth – Values of azimuth in degrees in range 0 - 360

Return type numpy array

contains (anotherDomain)

Checks if this Domain fully covers another Domain

Returns contains – True if this Domain fully covers another Domain, False otherwise

Return type bool

classmethod from_lonlat (lon, lat, add_gcps=True)

Create Domain object from input longitudes, latitudes arrays

Parameters

- lon (numpy.ndarray) – longitudes
- lat (numpy.ndarray) – latitudes
- add_gcps (bool) – Add GCPs from lon/lat arrays.

Returns d

Return type Domain

Examples

```python
>>> lon, lat = np.meshgrid(range(10), range(10))
>>> d1 = Domain.from_lonlat(lon, lat)
>>> d2 = Domain.from_lonlat(lon, lat, add_gcps=False) # add only geolocation
```

get_border (nPoints=10, fix_lon=True, **kwargs)

Generate two vectors with values of lat/lon for the border of domain

Parameters

- nPoints (int, optional) – Number of points on each border
• **fix_lon** (*bool*) – Convert longitudes to positive numbers when Domain crosses dateline?

**Returns**  lonVec, latVec – vectors with lon/lat values for each point at the border

**Return type**  lists

get_border_geojson (*args, **kwargs*)
Create border of the Polygon in GeoJson format

**Returns**

**Return type**  str, the Polygon border in GeoJson format

get_border_geometry (*args, **kwargs*)
Get OGR Geometry of the border Polygon

**Returns**

**Return type**  OGR Geometry, type Polygon

get_border_postgis (**kwargs**)
Get PostGIS formatted string of the border Polygon

**Returns**  str

**Return type**  ‘PolygonFromText(PolygonWKT)’

get_border_wkt (*args, **kwargs*)
Creates string with WKT representation of the border polygon

**Returns**  WKTPolygon – string with WKT representation of the border polygon

**Return type**  string

get_corners()
Get coordinates of corners of the Domain

**Returns**  lonVec, latVec – vectors with lon/lat values for each corner

**Return type**  lists

get_geolocation_grids (*stepSize=1, dstSRS=<sphinx.ext.autodoc.importer._MockObject object>)*
Get longitude and latitude grids representing the full data grid

If GEOLOCATION is not present in the self.vrt.dataset then grids are generated by converting pixel/line of each pixel into lat/lon If GEOLOCATION is present in the self.vrt.dataset then grids are read from the geolocation bands.

**Parameters**  *stepSize* (*int*) – Reduction factor if output is desired on a reduced grid size

**Returns**

• **longitude** (*numpy array*) – grid with longitudes

• **latitude** (*numpy array*) – grid with latitudes

get_min_max_lon_lat()
Get minimum and maximum of longitude and latitude geolocation grids

**Returns**  min_lon, max_lon, min_lat, max_lat, – min/max lon/lat values for the Domain

**Return type**  float
**get_pixelsize_meters()**

Returns the pixelsize (deltaX, deltaY) of the domain

For projected domains, the exact result which is constant over the domain is returned. For geographic (lon-lat) projections, or domains with no geotransform, the haversine formula is used to calculate the pixel size in the center of the domain.

**Returns** `delta_x, delta_y` – pixel size in X and Y directions given in meters

**Return type** `float`

**intersects(anotherDomain)**

Checks if this Domain intersects another Domain

**Returns** `intersects` – True if Domains intersects, False otherwise

**Return type** `bool`

**logger = None**

**name = None**

**overlaps(anotherDomain)**

Checks if this Domain overlaps another Domain

**Returns** `overlaps` – True if Domains overlaps, False otherwise

**Return type** `bool`

**reproject_gcps(srs_string=”)**

Reproject all GCPs to a new spatial reference system

Necessary before warping an image if the given GCPs are in a coordinate system which has a singularity in (or near) the destination area (e.g. poles for lonlat GCPs)

**Parameters**

- `srs_string (string)` – SRS given as Proj4 string. If empty ‘+proj=stere’ is used

**Notes**

Reprojects all GCPs to new SRS and updates GCPProjection

**shape()**

Return Numpy-like shape of Domain object (ySize, xSize)

**Returns** `shape` – Numpy-like shape of Domain object (ySize, xSize)

**Return type** `tuple of two INT`

**transform_points(colVector, rowVector, DstToSrc=0, dstSRS=<sphinx.ext.autodoc.importer._MockObject object>)**

Transform given lists of X,Y coordinates into lon/lat or inverse

**Parameters**

- `colVector (lists)` – X and Y coordinates in pixel/line or lon/lat coordinate system
- `DstToSrc (0 or 1)` – 0 - forward transform (pix/line => lon/lat) 1 - inverse transformation
- `dstSRS (NSR)` – destination spatial reference

**Returns** `X, Y` – X and Y coordinates in lon/lat or pixel/line coordinate system
Return type  lists

vrt = None

write_kml (xmlFileName=None, kmlFileName=None)
Write KML file with domains
Convert XML-file with domains into KML-file for GoogleEarth or write KML-file with the current Domain

Parameters

• xmlFileName (string, optional) – Name of the XML-file to convert. If only this value is given - kmlFileName=xmlFileName+'.kml'
• kmlFileName (string, optional) – Name of the KML-file to generate from the current Domain

write_kml_image (kmlFileName, kmlFigureName=None)
Create KML file for already projected image
Write Domain Image into KML-file for GoogleEarth

Parameters

• kmlFileName (str) – Name of the KML-file to generate from the current Domain
• kmlFigureName (str) – Name of the projected image stored in .png format

Examples

```python
>>> n.undo(100) # cancel previous reprojection
>>> lons, lats = n.get_corners() # Get corners of the image and the pixel resolution
>>> srsString = '+proj=latlong +datum=WGS84 +ellps=WGS84 +no_defs
>>> extentString = '-lle %f %f %f %f -ts 3000 3000' % (min(lons), min(lats), max(lons), max(lats))
>>> d = Domain(srs=srsString, ext=extentString) # Create Domain with stereographic projection, corner coordinates and resolution 1000m
>>> n.reproject(d)
>>> n.write_figure(filename=figureName, bands=[3], clim=[0,0.15], cmapName='gray', transparency=0)
>>> n.write_kml_image(kmlFileName=oPath + filename + '.kml', kmlFigureName=figureName) # 6.
```

class nansat.Nansat (filename='', mapper='', log_level=30, **kwargs)

Bases: nansat.domain.Domain, nansat.exporter.Exporter

Container for geospatial data. Performs all high-level operations.

n = Nansat(filename) opens the file with satellite or model data for reading, adds scientific metadata to bands, and prepares the data for further handling.

Parameters

• filename (str) – uri of the input file or OPeNDAP datastream
• mapper (str) – name of the mapper from nansat/mappers dir. E.g. ‘sentinel1_l1’, ‘asar’, ‘hirlam’, ‘meris_l1’, ‘meris_l2’, etc.
• log_level (int) – Level of logging. See: http://docs.python.org/howto/logging.html
• kwargs (additional arguments for mappers) –
Examples

```python
>>> n1 = Nansat(filename)
>>> n2 = Nansat(sentinel1_filename, mapper='sentinel1_l1')
>>> array1 = n1[1]
>>> array2 = n2['sigma0_HV']
```

Notes

The instance of Nansat class (the object `n`) contains information about geographical reference of the data (e.g. raster size, pixel resolution, type of projection, etc) and about bands with values of geophysical variables (e.g. water leaving radiance, normalized radar cross section, chlorophyll concentration, etc). The object `n` has methods for high-level operations with data. E.g.: * reading data from file (Nansat.__getitem__); * visualization (Nansat.write_figure); * changing geographical reference (Nansat.reproject); * exporting (Nansat.export) * and much more...

Nansat inherits from Domain (container of geo-reference information) Nansat uses instance of VRT (wrapper around GDAL VRT-files) Nansat uses instance of Figure (collection of methods for visualization)

```
ALT_FILL_VALUE = -10000.0
FILL_VALUE = 9.96921e+36
```

```python
add_band (array, parameters=None, nomem=False)
```

Add band from numpy array with metadata.

Create VRT object which contains VRT and RAW binary file and append it to self.vrt.band_vrts

Parameters

- `array` *(ndarray)* – new band data. Shape should be equal to shape
- `parameters` *(dict)* – band metadata: wkv, name, etc. (or for several bands)
- `nomem` *(bool)* – saves the vrt to a tempfile on disk?

Notes

Creates VRT object with VRT-file and RAW-file. Adds band to the self.vrt.

Examples

```python
>>> n.add_band(array, {'name': 'new_data'})
→ keep in memory
>>> n.add_band(array, nomem=True)  # add new band, keep on disk
```

```python
add_bands (arrays, parameters=None, nomem=False)
```

Add bands from numpy arrays with metadata.

Create VRT object which contains VRT and RAW binary file and append it to self.vrt.band_vrts

Parameters

- `arrays` *(list of ndarrays)* – new band data. Shape should be equal to shape
- `parameters` *(list of dict)* – band metadata: wkv, name, etc. (or for several bands)
• **nomem** (*bool*) – saves the vrt to a tempfile on disk?

### Notes

Creates VRT object with VRT-file and RAW-file. Adds band to the self.vrt.

### Examples

```python
>>> n.add_bands([array1, array2]) # add new bands, keep in memory
```

**bands ()**

Make a dictionary with all metadata from all bands

- **Returns** `b` – key = N, value = dict with all band metadata
- **Return type** dictionary

**crop** (*x_offset*, *y_offset*, *x_size*, *y_size*, *allow_larger=False*)

Crop Nansat object

Create superVRT, modify the Source Rectangle (SrcRect) and Destination Rectangle (DstRect) tags in the VRT file for each band in order to take only part of the original image, create new GCPs or new GeoTransform for the cropped object.

**Parameters**

- **x_offset** (*int*) – pixel offset of subimage
- **y_offset** (*int*) – line offset of subimage
- **x_size** (*int*) – width in pixels of subimage
- **y_size** (*int*) – height in pixels of subimage
- **allow_larger** (*bool*) – Allow resulting extent to be larger than the original image?

**Notes**

self.vrt : super-VRT is created with modified SrcRect and DstRect

- **Returns** `extent` – x_offset - X offset in the original dataset y_offset - Y offset in the original dataset x_size - width of the new dataset y_size - height of the new dataset
- **Return type** (x_offset, y_offset, x_size, y_size)

### Examples

```python
>>> extent = n.crop(10, 20, 100, 200)
```

**crop_interactive** (*band=1, maxwidth=1000, **kwargs*)

Interactively select boundary and crop Nansat object

**Parameters**

- **band** (*int or str*) – id of the band to show for interactive selection of boundaries
maxwidth (int) – large input data is downscaled to <maxwidth>

**kwargs (keyword arguments for imshow) –

Notes

self.vrt [VRT] superVRT is created with modified SrcRect and DstRect

Returns extent – x_offset - X offset in the original dataset y_offset - Y offset in the original
dataset x_size - width of the new dataset y_size - height of the new dataset

Return type (x_offset, y_offset, x_size, y_size)

Examples

```python
>>> extent = n.crop_interactive(band=1) # crop a subimage interactively
```

crop_lonlat (lonlim, latlim)
Crop Nansat object to fit into given longitude/latitude limit

Parameters

- **lonlim** (list of 2 float) – min/max of longitude
- **latlim** (list of 2 float) – min/max of latitude

Notes

self.vrt [VRT] crops vrt to size that corresponds to lon/lat limits

Returns extent – x_offset - X offset in the original dataset y_offset - Y offset in the original
dataset x_size - width of the new dataset y_size - height of the new dataset

Return type (x_offset, y_offset, x_size, y_size)

Examples

```python
>>> extent = n.crop(lonlim=[-10,10], latlim=[-20,20]) # crop for given lon/lat limits
```

digitize_points (band=1, **kwargs)
Get coordinates of interactively digitized points

Parameters

- **band** (int or str) – ID of Nansat band
- **kwargs** (keyword arguments for imshow) –

Returns points – list of 2xN arrays of points to be used in Nansat.get_transect()

Return type list

extend (left=0, right=0, top=0, bottom=0)
Extend domain from four sides
Parameters

- **left** (*int*) – number of pixels to add from left side
- **right** (*int*) – number of pixels to add from right side
- **top** (*int*) – number of pixels to add from top side
- **bottom** (*int*) – number of pixels to add from bottom side

**Notes**

Changes self.vrt by adding negative offset or setting size to be larger than original size.

```python
filename = None
classmethod from_domain(domain, array=None, parameters=None, log_level=30)
Create Nansat object from input Domain [and array with data]

Parameters

- **domain** (*Domain*) – Defines spatial reference system and geographical extent.
- **array** (*numpy NDarray*) – Data for the first band. Shape must correspond to shape of <domain>
- **parameters** (*dict*) – Metadata for the first band. May contain ‘name’, ‘wkv’ and other keys.
- **log_level** (*int*) – Level of logging.

get_GDALRasterBand(band_id=1)
Get a GDALRasterBand of a given Nansat object
If str is given find corresponding band number If int is given check if band with this number exists. Get a GDALRasterBand from vrt.

Parameters **band_id** (*int or str*) – if int - a band number of the band to fetch if str

```

```python
>>> b = n.get_GDALRasterBand(1)
>>> b = n.get_GDALRasterBand('sigma0')
```

get_band_number(band_id)
Return absolute band number
Check if given band_id is valid Return absolute number of the band in the VRT

Parameters **band_id** (*int or str or dict*) – if int : checks if such band exists and returns band_id if str : finds band with corresponding name if dict : finds first band with given metadata

Returns **int**

Return type absolute band number
**get_metadata** *(key=None, band_id=None, unescape=True)*

Get metadata from self.vrt.dataset

**Parameters**

- **key** *(str)* – name of the metadata key. If not given all metadata is returned
- **band_id** *(int or str)* – number or name of band to get metadata from. If not given, global metadata is returned
- **unescape** *(bool)* – Replace ‘&quot;’, ‘&amp;’, ‘&lt;’ and ‘&gt;’ with these symbols ” & < > ?

**Returns**

- *string with metadata if key is given and found*
- *dictionary with all metadata if key is not given*

**Raises** ValueError, if key is not found

**get_transect** *(points, bands, lonlat=True, smooth_radius=0, smooth_function=<sphinx.ext.autodoc.importer._MockObject object>, data=None, cornersonly=False)*

Get values from transect from given vector of points

**Parameters**

- **points** *(2xN list or array, N (number of points) >= 1)* – co-ordinates [[x1, x2, y2], [y1, y2, y3]]
- **bands** *(list of int or string)* – elements of the list are band number or band Name
- **lonlat** *(bool)* – If the points in lat/lon, then True. If the points in pixel/line, then False.
- **smooth_radius** *(int)* – If smootRadius is greater than 0, smooth every transect pixel as the median or mean value in a circule with radius equal to the given number.
- **smooth_function** *(func)* – function for averaging values collected within smooth radius
- **data** *(ndarray)* – alternative array with data to take values from

**Returns** transect

**Return type** numpy record array

**has_band** *(band)*

Check if self has band with name <band> :param band: name or standard_name of the band to check

**Return type** True/False if band exists or not

**list_bands** *(do_print=True)*

Show band information of the given Nansat object

Show serial number, longName, name and all parameters for each band in the metadata of the given Nansat object.

**Parameters** do_print *(boolean)* – print on screen?

**Returns** outString – formatted string with bands info

**Return type** String
logger = None
mapper = None
name = None
path = None

reproject

Change projection of the object based on the given Domain

Change projection of the object based on the given Domain

Create superVRT from self.vrt with AutoCreateWarpedVRT() using projection from the dst_domain. Modify XML content of the warped vrt using the Domain parameters. Generate warpedVRT and replace self.vrt with warpedVRT. If current object spans from 0 to 360 and dst_domain is west of 0, the object is shifted by 180 westwards.

Parameters

• dst_domain (domain) – destination Domain where projection and resolution are set

• resample_alg (int (GDALResampleAlg)) – 0 : NearestNeighbour 1 : Bilinear 2 : Cubic, 3 : CubicSpline 4 : Lanczos

• block_size (int) – size of blocks for resampling. Large value decrease speed but increase accuracy at the edge

• tps (bool) – Apply Thin Spline Transformation if source or destination has GCPs Usage of TPS can also be triggered by setting self.vrt.tps=True before calling to reproject. This options has priority over self.vrt.tps

• skip_gcps (int) – Using TPS can be very slow if the number of GCPs are large. If this parameter is given, only every [skip_gcp] GCP is used, improving calculation time at the cost of accuracy. If not given explicitly, ‘skip_gcps’ is fetched from the metadata of self, or from dst_domain (as set by mapper or user). [defaults to 1 if not specified, i.e. using all GCPs]

• addmask (bool) – If True, add band ‘swathmask’. 1 - valid data, 0 no-data. This band is used to replace no-data values with np.nan

Notes

self.vrt : VRT object with dataset replaced to warpedVRT dataset Integer data is returned by integer. Round off to decimal place. If you do not want to round off, convert the data types to GDT_Float32, GDT_Float64, or GDT_CFloat32.

See also:

http() //www.gdal.org/gdalwarp.html

resize

Proportional resize of the dataset.

The dataset is resized as (x_size*factor, y_size*factor) If desired width, height or pixelsize is specified, the scaling factor is calculated accordingly. If GCPs are given in a dataset, they are also rewritten.

Parameters

• factor (float, optional, default=1) – Scaling factor for width and height > 1 means increasing domain size < 1 means decreasing domain size
• `width(int, optional)` – Desired new width in pixels
• `height(int, optional)` – Desired new height in pixels
• `pixelsize(float, optional)` – Desired new pixelsize in meters (approximate). A factor is calculated from ratio of the current pixelsize to the desired pixelsize.
• `resample_alg(int (GDALResampleAlg), optional)` –
  -1 [Average (default),] 0 : NearestNeighbour 1 : Bilinear, 2 : Cubic, 3 : CubicSpline, 4 : Lanczos

Notes

`self.vrt.dataset` [VRT dataset of VRT object] raster size are modified to downscaled size. If GCPs are given in the dataset, they are also overwritten.

`set_metadata(key=", value=", band_id=None)`
Set metadata to `self.vrt.dataset`

Parameters

• `key (string or dictionary with strings)` – name of the metadata, or dictionary with metadata names, values
• `value (string)` – value of metadata
• `band_id (int or str)` – number or name of band Without : global metadata is set

Notes

`self.vrt.dataset` : sets metadata in GDAL current dataset

`time_coverage_end`
`time_coverage_start`
`undo(steps=1)`
Undo reproject, resize, add_band or crop of Nansat object
Restore the `self.vrt` from `self.vrt.vrt`

Parameters `steps (int)` – How many steps back to undo

Notes

Modifies `self.vrt`

`vrt = None`

`watermask(mod44path=None, dst_domain=None, **kwargs)`
Create numpy array with watermask (water=1, land=0)

250 meters resolution watermask from MODIS 44W Product: [http://www.glc.f.umd.edu/data/watermask/](http://www.glc.f.umd.edu/data/watermask/)
Watermask is stored as tiles in TIF(LZW) format and a VRT file. All files are stored in one directory. A tarball with compressed TIF and VRT files should be additionally downloaded from the Nansat documentation page: http://nansat.readthedocs.io/en/latest/source/features.html#differentiating-between-land-and-water

**The method**: Gets the directory either from input parameter or from environment variable MOD44WPATH. Opens Nansat object from the VRT file. Reprojects the watermask onto the current object using `reproject()` or `reproject_on_jcps()`. Returns the reprojected Nansat object.

**Parameters**

- `mod44path` (**string**) – path with MOD44W Products and a VRT file
- `dst_domain` (**Domain**) – destination domain other than self
- `tps` (**bool**) – Use Thin Spline Transformation in reprojection of watermask? See also Nansat.reproject()
- `skip_gcps` (**int**) – Factor to reduce the number of GCPs by and increase speed See also Nansat.reproject()

**Returns** watermask

**Return type** Nansat object with water mask in current projection.

See also:

- http() //www.glcf.umd.edu/data/watermask/

**write_figure** (filename=", bands=1, clim=None, addDate=False, array_modfunc=None, **kwargs)

Save a raster band to a figure in graphical format.

Get numpy array from the band(s) and band information specified either by given band number or band id.
- If three bands are given, merge them and create PIL image. – If one band is given, create indexed image.

Create Figure object and: Adjust the array brightness and contrast using the given min/max or histogram. Apply logarithmic scaling of color tone. Generate and append legend. Save the PIL output image in PNG or any other graphical format. If the filename extension is ‘tif’, the figure file is converted to GeoTiff.

**Parameters**

- `bands` (**integer or string or list (elements are integer or string),**) – default = 1 the size of the list has to be 1 or 3. if the size is 3, RGB image is created based on the three bands. Then the first element is Red, the second is Green, and the third is Blue.
- `clim` (**list with two elements or ‘hist’ to specify range of colormap**) – None (default) : min/max values are fetched from WKV, fallback: ‘hist’ [min, max] : min and max are numbers, or [[min, min, min], [max, max, max]]: three bands used ‘hist’ : a histogram is used to calculate min and max values
- `addDate` (**boolean**) – False (default) : no date will be added to the caption. True : the first time of the object will be added to the caption
array_modfunc (None) – None (default): figure created using array in provided band function: figure created using array modified by provided function

**kwargs (parameters for Figure()) –

Notes

if filename is specified, creates image file

Returns

Return type Figure object

Example

```python
>>> n.write_figure('test.jpg')  # write indexed image
>>> n.write_figure('test_rgb_hist.jpg', clim='hist', bands=[1, 2, 3])  # RGB image
>>> n.write_figure('r09_log3_leg.jpg', logarithm=True, legend=True,
    gamma=3, titleString='Title', fontSize=30,
    numOfTicks=15)  # add legend
>>> n.write_figure(filename='transparent.png', bands=[3],
    mask_array=wmArray,
    mask_lut={0: [0, 0, 0]},
    clim=[0, 0.15], cmapName='gray',
    transparency=[0, 0, 0])  # write transparent image
```

See also:

Figure()

http() //www.scipy.org/Cookbook/Matplotlib/Show_colormaps

write_geotiffimage (filename, band_id=1)

Writes an 8-bit GeoTiff image for a given band.

The colormap is fetched from the metadata item ‘colormap’. Fallback colormap is ‘gray’.

Color limits are fetched from the metadata item ‘minmax’. If ‘minmax’ is not specified, min and max of the raster data is used.

The method can be replaced by using nansat.write_figure(). However, write_figure uses PIL, which does not allow Tiff compression. This gives much larger files.

Parameters

* filename (str) –
* band_id (int or str) –

class nansat.Figure (nparray, **kwargs)

Bases: object

Perform operation with graphical files: create, append legend, save.

Figure instance is created in the Nansat.write_figure method. The methods below are applied consequently in order to generate a figure from one or three bands, estimate min/max, apply logarithmic scaling, convert to uint8, append legend, save to a file

CAPTION_LOCATION_X = 0.1
CAPTION_LOCATION_Y = 0.25
CBAR_HEIGHT = 0.15
CBAR_HEIGHTMIN = 5
CBAR_LOCATION_X = 0.1
CBAR_LOCATION_Y = 0.5
CBAR_WIDTH = 0.8
CBTICK_LOC_ADJUST_X = 5
CBTICK_LOC_ADJUST_Y = 3
DEFAULT_EXTENSION = '.png'
LEGEND_HEIGHT = 0.1
TITLE_LOCATION_X = 0.1
TITLE_LOCATION_Y = 0.05

add_latlon_grids(**kwargs)
Add lat/lon grid lines into the PIL image
Compute step of the grid Make matrices with binarized lat/lon Find edge (make line) Convert to mask Add mask to PIL

Parameters
- of Figure__init__() parameters (Any) –
  - latGrid (numpy array) – array with values of latitudes
  - lonGrid (numpy array) – array with values of longitudes
  - lonTicks (int or list) – number of lines to draw or locations of gridlines
  - latTicks (int or list) – number of lines to draw or locations of gridlines
- Modifies –
- ---------
- self.pilImg –

add_latlon_labels(**kwargs)
Add lat/lon labels along upper and left side
Compute step of labels Get lat/lon for these labels from latGrid, lonGrid Print labels to PIL in white

Parameters
- of Figure__init__() parameters (Any) –
  - latGrid (numpy array) – array with values of latitudes
  - lonGrid (numpy array) – array with values of longitudes
  - lonTicks (int or list) – number of lines to draw or locations of gridlines
  - latTicks (int or list) – number of lines to draw or locations of gridlines
- Modifies –
- ---------
- self.pilImg –
add_logo(**kwargs)
Insert logo into the PIL image
Read logo from file as PIL. Resize to the given size. Pan using the given location. Paste into pilImg.

Parameters

- of Figure__init__() parameters (Any)
- Modifies
- self.pilImg

apply_logarithm(**kwargs)
Apply a tone curve to the array
After the normalization of the values from 0 to 1, logarithm is applied. Then the values are converted to
the normal scale.

Parameters

- of Figure__init__() parameters (Any)
- Modifies
- self.array (numpy array)

apply_mask(**kwargs)
Apply mask for coloring land, clouds, etc
If mask_array and mask_lut are provided as input parameters, The pixels in self.array which have index
equal to mask_lut key in mask_array will have color equal to mask_lut value
apply_mask should be called only after convert palettesize (i.e. to uint8 data).

Parameters

- of Figure__init__() parameters (Any)
- self.array (numpy array)

array = None
caption = '
clim_from_histogram(**kwargs)
Estimate min and max pixel values from histogram
if ratio=1.0, simply the minimum and maximum values are returned. if 0 < ratio < 1.0, get the histogram of
the pixel values. Then get rid of (1.0-ratio)/2 from the both sides and return the minimum and maximum
values.

Parameters of Figure.__init__() parameters (Any)

Returns clim – minimum and maximum pixel values for each band

Return type numpy 2D ((3x2) or (1x2))
**clip(**\***kwargs\**)**

Convert self.array to values between cmin and cmax
if pixel value < cmin, replaced to cmin. if pixel value > cmax, replaced to cmax.

**Parameters**

- of `Figure.__init__() parameters` (Any) –
  - Modifies –
  - ---------
  - `self.array` (numpy array) –
  - `self.cmax` (self.cmin,) –

```python
cmapName = 'jet'
cmax = [1.0]
cmin = [0.0]
```

**convert_pallettesize(**\***kwargs\***)**

Convert self.array to palette color size in uint8

**Parameters**

- of `Figure.__init__() parameters` (Any) –
  - Modifies –
  - ---------
  - `self.array` (numpy array (=>uint8)) –

**create_legend(**\***kwargs\***)**

self.legend is replaced from None to PIL image
PIL image includes colorbar, caption, and titleString.

**Parameters**

- of `Figure.__init__() parameters` (Any) –
  - Modifies –
  - ---------
  - `self.legend` (PIL image) –

**create_pilImage(**\***kwargs\***)**

self.create_pilImage is replaced from None to PIL image

If three images are given, create a image with RGB mode. if self.pilImgLegend is not None, it is pasted.

If one image is given, create a image with P(palette) mode. if self.pilImgLegend is not None, self.array is extended before create the pilImg and then paste pilImgLegend onto it.

**Parameters**

- of `Figure.__init__() parameters` (Any) –
  - Modifies –
  - ---------
  - `self.pilImg` (PIL image) – PIL image with / without the legend
• **self.array**(replace to None) –

    extensionList = ['png', 'PNG', 'tif', 'TIF', 'bmp', 'BMP', 'jpg', 'JPG', 'jpeg', 'JPEG'

    fontRatio = 1
    fontSize = None
    gamma = 2.0
    latGrid = None
    latTicks = 5
    legend = False
    logarithm = False
    logoFileName = None
    logoLocation = [0, 0]
    logoSize = None
    lonGrid = None
    lonTicks = 5
    mask_array = None
    mask_lut = None
    numOfColor = 250
    numOfTicks = 5
    palette = None
    pilImg = None
    pilImgLegend = None

    process(**kwargs)

    Do all common operations for preparation of a figure for saving
      1. Modify default values of parameters by the provided ones (if any)
      2. Clip to min/max
      3. Apply logarithm if required
      4. Convert data to uint8
      5. Create palette
      6. Apply mask for colouring land, clouds, etc if required
      7. Create legend if required
      8. Create PIL image
      9. Add logo if required

    Parameters

    • of Figure.__init__() parameters (Any) –
      • Modifies –
      • -------- –

---

Chapter 3. Packages and modules
self.d –
self.array –
self.palette –
self.pilImgLegend –
self.pilImg –

ratio = 1.0
save(fileName, **kwargs)
Save self.pilImg to a physical file
If given extension is JPG, convert the image mode from Palette to RGB

Parameters

• fileName (string) – name of output file
• of Figure.__init__() parameters (Any) –
• Modifies –
• --------
• self.pilImg (None) –

subsetArraySize = 100000
titleString = ''
transparency = None

3.2 nansat.mappers package

3.2.1 Submodules

3.2.2 nansat.mappers.envisat module

class nansat.mappers.envisat.Envisat
  Bases: object

Methods/data shared between Envisat mappers

This class is needed to read awkward N1 format of ENVISAT Mostly it support reading of variables from ADS (additional data sets) which are TIE_POINTS_ADS (MERIS) or GEOLOCATION_GRID_ADS (ASAR)

add_geolocation_from_ads (gdalDataset, zoomSize=500, step=1)
Add geolocation domain metadata to the dataset

Get VRTs with zoomed arrays of lon and lat Create geolocation object and add to the metadata

Parameters

• gdalDataset (GDAL Dataset) – input dataset
• zoomSize (int, optional, 500) – size, to which the ADS array will be zoomed using scipy array of this size will be stored in memory
• step (int) – step of pixel and line in GeolocationArrays. lat/lon grids are generated at that step
• Modifies
• --------
• Geolocation Array metadata (Ads)

allADSParams = {
    'ASA_': {
        'width': 11,
        'list': {
            'last_line_longs': {
                'dataType': <sphinx.ext.autodoc.importer._MockObject ... object at 0x7f197fa48190>,
                'units': '(10)^-6 deg',
                'offset': 13
            }
        }
    }
}

create_VRT_from_ADS (adsName, zoomSize=500)

Create VRT with a band from Envisat ADS metadata

Read offsets of the <adsName> ADS. Read 2D matrix of binary values from ADS from file. Zoom array with ADS data to <zoomSize>. Zooming is needed to create smooth matrices. Array is zoomed to small size because it is stored in memory. Later the VRT with zoomed array is VRT.get_resized_vrt() in order to match the size of the Nansat object.

Create VRT from the ADS array.

**Parameters**

- adsName (str) – name of variable from ADS to read. should match allADSParams
- zoomSize (int, optional, 500) – size, to which original matrix from ADSR is zoomed using scipy.zoom

**Returns**

adsVrt

**Return type**

VRT, vrt with a band created from ADS array

get_ads_vrts (gdalDataset, adsNames, zoomSize=500, step=1, **kwargs)

Create list with VRTs with zoomed and resized ADS arrays

For given names of variables (which should match self.allADSParams): Get VRT with zoomed ADS array Get resized VRT

**Parameters**

- gdalDataset (GDAL Dataset) – input dataset
- adsNames (list with strings) – names of variables from self.allADSParams['list']
- zoomSize (int, 500) – size to which the ADS array will be zoomed by scipy.zoom
- step (int, 1) – step, at which data will be given

**Returns**

adsVRTs – list with resized VRT with zoomed arrays

**Return type**

list with VRT

get_array_from_ADS (adsName)

Create VRT with a band from Envisat ADS metadata

Read offsets of the <adsName> ADS. Read 2D matrix of binary values from ADS from file. Read last line ADS (in case of ASAR). Zoom array with ADS data to <zoomSize>. Zooming is needed to create smooth matrices. Array is zoomed to small size because it is stored in memory. Later the VRT with zoomed array is VRT.get_resized_vrt() in order to match the size of the Nansat object. Create VRT from the ADS array.

**Parameters**

- adsName (str) – name of variable from ADS to read. should match allADSParams

**Returns**

adsVrt – vrt with a band created from ADS array

**Return type**

VRT
lonlatNames = {'ASA_': ['first_line_longs', 'first_line_lats'], 'MER_': ['longitude', 'latitude']}

read_binary_line(offset, fmtString, length)
Read line with binary data at given offset

Parameters
- offset (int, start of reading) -
- fmtString (str, data type format) -
- length (number of values to read) -

Returns binaryValues – values which are read from the file. the number of elements is length
Return type list

read_offset_from_header(gadsDSName)
Read offset of ADS from text header.

Find a location of gadsDSName. Adjust the location with textOffset and read the text at the location.
Convert the text to integer and set it into offsetDict.

Returns offsetDict – offset of DS, size of DS, number of records, size of record
Return type dictionary

read_scaling_gads(indices)
Read Scaling Factor GADS to get scalings of MERIS L1/L2

Parameters
- filename (string) -
- indices (list) -

Returns
Return type list

setup_ads_parameters(filename, gdalMetadata)
Select set of params and read offset of ADS

structFmt = 
{<sphinx.ext.autodoc.importer._MockObject object at 0x7f197fa481d0>: '>i', ...
<sphinx.ext.autodoc.importer._MockObject object at 0x7f197fa48a90>: '>f', ...
<sphinx.ext.autodoc.importer._MockObject object at 0x7f197fa48bd0>: '>H'}

3.2.3 nansat.mappers.globcolour module

class nansat.mappers.globcolour.Globcolour
Mapper for GLOBCOLOR L3M products

make_rrsw_meta_entry(nlwMetaEntry)
Make metaEntry for calculation of Rrsw

varname2wkv = {'BBP_mean': 'volume_backscattering_coefficient_of_radiative_flux_in_sea_water_due_to_suspended_particles', ...

3.2.4 nansat.mappers.hdf4_mapper module

class nansat.mappers.hdf4_mapper.HDF4Mapper(x_size=1, y_size=1, metadata=None, nomem=False, **kwargs)
Bases: nansat.vrt.VRT
**find_metadata** (*iMetadata, iKey, default=“")

Find metadata which has similar key

**Parameters**

- **iMetadata** (*dict*) – input metadata, usually gdalMetadata
- **iKey** (*str*) – key to search for
- **default** (*str*) – default value

### 3.2.5 nansat.mappers.mapper_aapp_l1b module

class nansat.mappers.mapper_aapp_l1b.Mapper (*filename, gdalDataset, gdalMetadata, **kwargs*)

**Bases:** nansat.vrt.VRT

VRT with mapping of WKV for AVHRR L1B output from AAPP

### 3.2.6 nansat.mappers.mapper_aapp_l1c module

class nansat.mappers.mapper_aapp_l1c.Mapper (*filename, gdalDataset, gdalMetadata, **kwargs*)

**Bases:** nansat.vrt.VRT

VRT with mapping of WKV for AVHRR L1C output from AAPP

### 3.2.7 nansat.mappers.mapper_amsr2_l1r module

class nansat.mappers.mapper_amsr2_l1r.Mapper (*filename, gdalDataset, gdalMetadata, **kwargs*)

**Bases:** nansat.vrt.VRT

Mapper for AMSR-2 L1 data

### 3.2.8 nansat.mappers.mapper_amsr2_l3 module

class nansat.mappers.mapper_amsr2_l3.Mapper (*filename, gdalDataset, gdalMetadata, **kwargs*)

**Bases:** nansat.vrt.VRT

Mapper for Level-3 AMSR2 data from https://gcom-w1.jaxa.jp

freqs = [6, 7, 10, 18, 23, 36, 89]

### 3.2.9 nansat.mappers.mapper_amsre_uham_leadfraction module

class nansat.mappers.mapper_amsre_uham_leadfraction.Mapper (*filename, gdalDataset, gdalMetadata, **kwargs*)

**Bases:** nansat.vrt.VRT
3.2.10 nansat.mappers.mapper_arome module

```python
class nansat.mappers.mapper_arome.Mapper(*args, **kwargs)
    Bases: nansat.mappers.mapper_netcdf_cf.Mapper
```

3.2.11 nansat.mappers.mapper_asar module

```python
class nansat.mappers.mapper_asar.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)
    Bases: nansat.vrt.VRT, nansat.mappers.envisat.Envisat
    VRT with mapping of WKV for ASAR Level 1
    See also:
    http://envisat.esa.int/handbooks/asar/CNTR6-6-9.htm#eph.asar.asardf.asarrec.ASAR_Geo_Grid_ADSR
```

3.2.12 nansat.mappers.mapper_ascat_nasa module

```python
class nansat.mappers.mapper_ascat_nasa.Mapper(filename, gdalDataset, gdalMetadata, latlonGrid=None, mask='.', **kwargs)
    Bases: nansat.vrt.VRT
    Create VRT with mapping of WKV
```

3.2.13 nansat.mappers.mapper_aster_l1a module

```python
class nansat.mappers.mapper_aster_l1a.Mapper(filename, gdalDataset, gdalMetadata, GCP_COUNT=10, bandNames=["VNIR_Band1", "VNIR_Band2", "VNIR_Band3N"], bandWaves=[560, 660, 820], **kwargs)
    Bases: nansat.vrt.VRT
    Mapper for ASTER L1A VNIR data
```

3.2.14 nansat.mappers.mapper_aster_l1b module

```python
class nansat.mappers.mapper_aster_l1b.Mapper(filename, gdalDataset, gdalMetadata, emrange='VNIR', **kwargs)
    Bases: nansat.mappers.hdf4_mapper.HDF4Mapper
    VRT with mapping of WKV for MODIS Level 1 (QKM, HKM, 1KM)
```

3.2.15 nansat.mappers.mapper_case2reg module

```python
class nansat.mappers.mapper_case2reg.Mapper(filename, gdalDataset, gdalMetadata, wavelengths=[None, 413, 443, 490, 510, 560, 620, 665, 681, 709, 753, None, 778, 864], **kwargs)
    Bases: nansat.mappers.mapper_generic.Mapper
    Mapping for the BEAM/Visat output of Case2Regional algorithm
```

3.2. nansat.mappers package
3.2.16 nansat.mappers.mapper_cmems module

```python
class nansat.mappers.mapper_cmems.Mapper(*args, **kwargs)
    Bases: nansat.mappers.mapper_netcdf_cf.Mapper
    time_coverage()
nansat.mappers.mapper_cmems.get_gcmd_keywords_mapping()
```

3.2.17 nansat.mappers.mapper_csks module

```python
class nansat.mappers.mapper_csks.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)
    Bases: nansat.vrt.VRT
    VRT with mapping of WKV for Cosmo-Skymed
```

3.2.18 nansat.mappers.mapper_ecmwf_metno module

```python
class nansat.mappers.mapper_ecmwf_metno.Mapper(*args, **kwargs)
    Bases: nansat.mappers.mapper_netcdf_cf.Mapper
```

3.2.19 nansat.mappers.mapper_emodnet module

```python
class nansat.mappers.mapper_emodnet.Mapper(inputFileName, gdalDataset, gdalMetadata, logLevel=30, **kwargs)
    Bases: nansat.vrt.VRT
```

3.2.20 nansat.mappers.mapper_generic module

```python
class nansat.mappers.mapper_generic.Mapper(inputFileName, gdalDataset, gdalMetadata, logLevel=30, rmMetadata=['NETCDF_VARNAME', '_Unsigned', 'ScaleRatio', 'ScaleOffset', 'dods_variable'], **kwargs)
    Bases: nansat.vrt.VRT
```

```python
add_gcps_from_metadata(geoMetadata)
    Get GCPs from strings in metadata and insert in dataset
```

```python
add_gcps_from_variables(filename)
    Get GCPs from GCPPixel, GCPLine, GCPX, GCPY, GCPZ variables
```

```python
repare_projection(projection)
    Replace odd symbols in projection string ‘|’ => ‘;’, ‘&’ => ‘’
```

3.2.21 nansat.mappers.mapper_globcolour_l3b module

```python
class nansat.mappers.mapper_globcolour_l3b.Mapper(filename, gdalDataset, gdalMetadata, latlonGrid=None, mask='', **kwargs)
    Bases: nansat.vrt.VRT, nansat.mappers.globcolour.Globcolour
    Create VRT with mapping of WKV for MERIS Level 2 (FR or RR)
```
3.2.22 nansat.mappers.mapper_globcolour_l3m module

class nansat.mappers.mapper_globcolour_l3m.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)

Bases: nansat.vrt.VRT, nansat.mappers.globcolour.Globcolour

Mapper for GLOBCOLOR L3M products

3.2.23 nansat.mappers.mapper_goci_l1 module

class nansat.mappers.mapper_goci_l1.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)

Bases: nansat.vrt.VRT

VRT with mapping of WKV for MODIS Level 1 (QKM, HKM, 1KM)

3.2.24 nansat.mappers.mapper_hirlam module

class nansat.mappers.mapper_hirlam.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)

Bases: nansat.vrt.VRT

VRT with mapping of WKV for HIRLAM

3.2.25 nansat.mappers.mapper_hirlam_wind_netcdf module

class nansat.mappers.mapper_hirlam_wind_netcdf.Mapper(filename, gdalDataset, gdalMetadata, logLevel=30, **kwargs)

Bases: nansat.vrt.VRT

3.2.26 nansat.mappers.mapper_kmss module

class nansat.mappers.mapper_kmss.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)

Bases: nansat.vrt.VRT

VRT with mapping of WKV for KMSS TOA tiff data

3.2.27 nansat.mappers.mapper_landsat module

class nansat.mappers.mapper_landsat.Mapper(filename, gdalDataset, gdalMetadata, resolution='low', **kwargs)

Bases: nansat.vrt.VRT

Mapper for LANDSAT5,6,7,8 .tar.gz or tif files

3.2.28 nansat.mappers.mapper_meris_l1 module

class nansat.mappers.mapper_meris_l1.Mapper(filename, gdalDataset, gdalMetadata, geolocation=False, zoomSize=500, step=1, **kwargs)

Bases: nansat.vrt.VRT, nansat.mappers.envisat.Envisat
VRT with mapping of WKV for MERIS Level 1 (FR or RR)

3.2.29 nansat.mappers.mapper_meris_l2 module

```python
class nansat.mappers.mapper_meris_l2.Mapper(filename, gdalDataset, gdalMetadata, geolocation=False, zoomSize=500, step=1, **kwargs)
```

Bases: nansat.vrt.VRT, nansat.mappers.envisat.Envisat

Create VRT with mapping of WKV for MERIS Level 2 (FR or RR)

3.2.30 nansat.mappers.mapper_metno_hires_seaice module

```python
class nansat.mappers.mapper_metno_hires_seaice.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)
```

Bases: nansat.vrt.VRT

Create VRT with mapping of WKV for Met.no seaice

3.2.31 nansat.mappers.mapper_metno_local_hires_seaice module

3.2.32 nansat.mappers.mapper_mod44w module

```python
class nansat.mappers.mapper_mod44w.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)
```

Bases: nansat.vrt.VRT

VRT with mapping of WKV for MOD44W produc (MODIS watermask at 250 m)

3.2.33 nansat.mappers.mapper_modis_l1 module

```python
class nansat.mappers.mapper_modis_l1.Mapper(filename, gdalDataset, gdalMetadata, GCP_COUNT=30, **kwargs)
```

Bases: nansat.mappers.hdf4_mapper.HDF4Mapper

VRT with mapping of WKV for MODIS Level 1 (QKM, HKM, 1KM)

3.2.34 nansat.mappers.mapper_ncep module

```python
class nansat.mappers.mapper_ncep.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)
```

Bases: nansat.vrt.VRT

VRT with mapping of WKV for NCEP GFS

3.2.35 nansat.mappers.mapper_ncep_wind module

```python
class nansat.mappers.mapper_ncep_wind.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)
```

Bases: nansat.vrt.VRT

VRT with mapping of WKV for NCEP GFS
3.2.36 nansat.mappers.mapper_ncep_wind_online module

class nansat.mappers.mapper_ncep_wind_online.Mapper(filename, gdalDataset, gdalMetadata, outFolder=u'/home/docs/ncep_gfs_downloads', **kwargs)

Bases: nansat.vrt.VRT, object
VRT with mapping of WKV for NCEP GFS

3.2.37 nansat.mappers.mapper_netcdf_cf module

Nansat NetCDF-CF mapper
Check CF-compliance of your files here: http://cfconventions.org/compliance-checker.html

class nansat.mappers.mapper_netcdf_cf.Mapper(filename, gdal_dataset, gdal_metadata, *args, **kwargs)

Bases: nansat.vrt.VRT
times()
    Get times from time variable
    NOTE: This cannot be done with gdal because the time variable is a vector

3.2.38 nansat.mappers.mapper_nora10_local_vpv module

class nansat.mappers.mapper_nora10_local_vpv.Mapper(filename, gdalDataset, gdalMetadata, logLevel=30, **kwargs)

Bases: nansat.vrt.VRT

3.2.39 nansat.mappers.mapper_obpg_l2 module

class nansat.mappers.mapper_obpg_l2.Mapper(filename, gdalDataset, gdalMetadata, GCP_COUNT=10, **kwargs)

Bases: nansat.mappers.obpg.OBPGL2BaseClass
Mapper for SeaWIFS/MODIS/MERIS/VIIRS L2 data from OBPG
TODO: * Test on SeaWIFS * Test on MODIS Terra

3.2.40 nansat.mappers.mapper_obpg_l2_nc module

class nansat.mappers.mapper_obpg_l2_nc.Mapper(filename, gdalDataset, gdalMetadata, GCP_COUNT=10, **kwargs)

Bases: nansat.mappers.obpg.OBPGL2BaseClass
Mapper for SeaWIFS/MODIS/MERIS/VIIRS L2 data from OBPG in NC4 format

3.2.41 nansat.mappers.mapper_obpg_l3 module

class nansat.mappers.mapper_obpg_l3.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)

Bases: nansat.vrt.VRT
Mapper for Level-3 Standard Mapped Image from http://oceancolor.gsfc.nasa.gov

```python
param2wkv = {u'CDOM Index': u'volume_absorption_coefficient_of_radiative_flux_in_sea_water', ...}
```

### 3.2.42 nansat.mappers.mapper_ocean_productivity module

```python
class nansat.mappers.mapper_ocean_productivity.Mapper(filename, gdalDataset, gdalMetadata, **kwargs)
```

Bases: `nansat.vrt.VRT`

Mapper for Ocean Productivity website http://www.science.oregonstate.edu/ocean.productivity/

```python
bandNames = {'instantaneous_downwelling_photosynthetic_photon_radiance_in_sea_water': 'ipar', 'mass_concentration_of_chlorophyll_a_in_sea_water': 'algal_1', 'particle_backscatter_at_443_nm': 'bbp_443', 'sea_surface_temperature': 'SST'}
```

```python
param2wkv = {'bbp': 'particle_backscatter_at_443_nm', 'chl': 'mass_concentration_of_chlorophyll_a_in_sea_water', ...}
```

### 3.2.43 nansat.mappers.mapper_opendap_arome module

```python
class nansat.mappers.mapper_opendap_arome.Mapper(*args, **kwargs)
```

Bases: `nansat.mappers.opendap.Opendap, nansat.mappers.mapper_arome.Mapper`

```python
baseURLs = ['http://thredds.met.no/thredds/catalog/arome25/catalog.html']
```

### 3.2.44 nansat.mappers.mapper_opendap_globcurrent module

```python
class nansat.mappers.mapper_opendap_globcurrent.Mapper(filename, gdalDataset, gdalMetadata, date=None, ds=None, bands=None, cachedir=None, **kwargs)
```

Bases: `nansat.mappers.opendap.Opendap`

VRT with mapping of WKV for NCEP GFS

```python
baseURLs = ['http://www.ifremer.fr/opendap/cerdap1/globcurrent/v2.0/']
```

```python
convert_dstime_datetimes(dsTime)
```

Convert time variable to np.datetime64

```python
srcDSProjection
```

Used by autodoc_mock_imports.

```python
timeCalendarStart = '1950-01-01'
```

```python
timeVarName = 'time'
```

```python
xName = 'lon'
```

```python
yName = 'lat'
```
3.2.45 nansat.mappers.mapper_opendap_globcurrent_thredds module

```
class nansat.mappers.mapper_opendap_globcurrent_thredds.Mapper(filename, gdal.Dataset, gdal.Metadata, date=None, ds=None, bands=None, cachedir=None, **kwargs)
```

Bases: nansat.mappers.opendap.Opendap

VRT with mapping of WKV for NCEP GFS

```
baseURLs = ['http://tds0.ifremer.fr/thredds/dodsC/CLS-L4']
```

```
convert_dstime_datetimes(dsTime)
    Convert time variable to np.datetime64
```

```
srcDSProjection
    Used by autodoc_mock_imports.
```

```
timeCalendarStart = '1950-01-01'
timeVarName = 'time'
xName = 'lon'
yName = 'lat'
```

3.2.46 nansat.mappers.mapper_opendap_occci module

```
class nansat.mappers.mapper_opendap_occci.Mapper(filename, gdalDataset, gdalMetadata, date=None, ds=None, bands=None, cachedir=None, **kwargs)
```

Bases: nansat.mappers.opendap.Opendap

VRT with mapping of WKV for NCEP GFS

```
```

```
convert_dstime_datetimes(dsTime)
    Convert time variable to np.datetime64
```

```
srcDSProjection
    Used by autodoc_mock_imports.
```

```
timeCalendarStart = '1970-01-01'
timeVarName = 'time'
xName = 'lon'
yName = 'lat'
```
3.2.47 nansat.mappers.mapper_opendap_osisaf module

```python
class nansat.mappers.mapper_opendap_osisaf.Mapper(filename, gdalDataset, gdalMetadata, date=None, ds=None, bands=None, cachedir=None, **kwargs):
    Bases: nansat.mappers.opendap.Opendap
    VRT with mapping of WKV for NCEP GFS
    baseURLs = ['http://thredds.met.no/thredds/dodsC/cryoclim/met.no/osisaf-nh', 'http://thredds.met.no/thredds/dodsC/cryoclim/met.no/osisaf-nh']
    convert_dstime_datetimes(dsTime)
        Convert time variable to np.datetime64
    srcDSProjection
        Used by autodoc_mock_imports.
    t0 = datetime.datetime(1978, 1, 1, 0, 0)
    timeVarName = 'time'
    xName = 'xc'
    yName = 'yc'
```

3.2.48 nansat.mappers.mapper_opendap_siwtacsst module

```python
class nansat.mappers.mapper_opendap_siwtacsst.Mapper(filename, gdalDataset, gdalMetadata, date=None, ds=None, bands=None, cachedir=None, **kwargs):
    Bases: nansat.mappers.opendap.Opendap
    VRT with mapping of WKV for NCEP GFS
    convert_dstime_datetimes(dsTime)
        Convert time variable to np.datetime64
    srcDSProjection
        Used by autodoc_mock_imports.
    t0 = datetime.datetime(1981, 1, 1, 0, 0)
    timeVarName = 'time'
    xName = 'lon'
    yName = 'lat'
```

3.2.49 nansat.mappers.mapper_opendap_sstcci module

```python
class nansat.mappers.mapper_opendap_sstcci.Mapper(filename, gdalDataset, gdalMetadata, date=None, ds=None, bands=None, cachedir=None, **kwargs):
    Bases: nansat.mappers.opendap.Opendap
    VRT with mapping of WKV for NCEP GFS
```

---

Chapter 3. Packages and modules
baseURLs = ['http://dap.ceda.ac.uk/data/neodc/esacci/sst/data/lt/Analysis/L4/v01.1/']

def convert_dstime_datetimes(dsTime):
    Convert time variable to np.datetime64

def srcDSProjection:
    Used by autodoc_mock_imports.

timeCalendarStart = '1981-01-01'
timeVarName = 'time'
xName = 'lon'
yName = 'lat'

3.2.50 nansat.mappers.mapper_pathfinder52 module

class nansat.mappers.mapper_pathfinder52.Mapper(filename, gdalDataset, gdalMetadata, minQual=4, **kwargs)

    Bases: nansat.vrt.VRT
    Mapper PATHFINDER (local files)
    TODO: * remote files

3.2.51 nansat.mappers.mapper_radarsat2 module

class nansat.mappers.mapper_radarsat2.Mapper(inputFileName, gdalDataset, gdalMetadata, xmlonly=False, **kwargs)

    Bases: nansat.vrt.VRT
    Create VRT with mapping of WKV for Radarsat2
    init_from_xml(productXml)
        Fast init from metada in XML only

3.2.52 nansat.mappers.mapper_sentinel1_l1 module

class nansat.mappers.mapper_sentinel1_l1.Mapper(filename, gdalDataset, gdalMetadata, fast=False, fixgcp=True, **kwargs)

    Bases: nansat.vrt.VRT
    Create VRT with mapping of Sentinel-1 (A and B) stripmap mode (S1A_SM)

    Parameters
    
    • filename (str) – name of input Sentinel-1 L1 file
    • gdalDataset (None) –
    • gdalMetadata (None) –
    • fast (bool) – Flag that triggers faster reading of metadata from Sentinel-1 file. If True, no bands are added to the dataset and georeference is not corrected. If False, all bands are added and GCPs are corrected if necessary (see Mapper.correct_geolocation_data for details).

    Note: Creates self.dataset and populates it with S1 bands (when fast=False).
correct_geolocation_data()
Correct lon/lat values in geolocation data for points high above ground (incorrect)

Each GCP in Sentinel-1 L1 image (both in the GeoTIF files and Annotation LUT) have five coordinates: X, Y, Z (height), Pixel and Line. On some scenes that cover Greenland (and probably other lands) some GCPs have height above zero even over ocean. This is incorrect, because the radar signal comes actually from the surface and not from a point above the ground as stipulated in such GCPs. Correction of GCPs in this function is equivalent to reverse DEM correction of SAR data.

Notes

Updates ‘pixel’ and ‘height’ in self.annotation_data

static create_gcps(x, y, z, p, l)
Create GCPs from geolocation data

Parameters

• y, z, p, l(x) –
• arrays with value of X, Y, Z, Pixel and Line coordinates. \( (N-D) \) –
• and Y are typically lon, lat, Z = height. \( (X) \) –

Returns gcps

Return type list with GDAL GCPs

read_annotation(.annotation_files)
Read lon, lat, etc from annotation XML

Parameters: ——— _ annotation_files : list
strings with names of annotation files

data [dict]

geolocation data from the XML as 2D np.arrays. Keys: pixel, line, longitude, latitude, height, incidenceAngle, elevationAngle: 2D arrays shape : tuple (shape of geolocation data arrays)
x_size, y_size : int pol : list

read_calibration(xml, vectorListName, variable_names, pol)
Read calibration data from calibration or noise XML files :param xml: String with XML from calibration or noise files :type xml: str :param vectorListName: tag of the element that contains lists with LUT values :type vectorListName: str :param variable_names: names of LUT variable to read :type variable_names: list of str :param pol: HH, HV, etc :type pol: str

Returns data – Calibration or noise data. Keys: The same as variable_names + ‘pixel’, ‘line’

Return type dict

read_manifest_data(input_file)
Read information (time_coverage_start, etc) manifest XML

Parameters: ——— _ input_file : str
name of manifest file

data [dict]
manifest data. Keys: time_coverage_start  time_coverage_end  platform_familyName  platform_number

vrts_from_arrays (data, variable_names, pol="", resize=True, resample_alg=2)
Convert input dict with arrays into dict with VRTs
Parameters: _______ data : dict

2D arrays with data from LUT

variable_names [list of str] variable names that should be converted to VRTs
pol [str] HH, HV, etc
resize [bool] Shall VRT be zoomed to full size?
resample_alg [int] Index of resampling algorithm. See VRT.get_resized_vrt()

vrts : dict with (resized) VRTs

3.2.53 nansat.mappers.mapper_sentinel1_l2 module

class nansat.mappers.mapper_sentinel1_l2.Mapper (filename, gdalDataset, gdalMetadata, product_type='RVL', GCP_COUNT=10, **kwargs)

Bases: nansat.vrt.VRT

Create VRT with mapping of Sentinel-1A stripmap mode (S1A_SM)

3.2.54 nansat.mappers.mapper_topography module

class nansat.mappers.mapper_topography.Mapper (filename, gdal_dataset, *args, **kwargs)

Bases: nansat.vrt.VRT


filename [string] The vrt filename, e.g., gtopo30.vrt
gdal_dataset [osgeo.gdal.Dataset] The GDAL dataset returned by gdal.Open(filename)

Either the name of a GTOPO30 DEM file or GMTED2010 tif file, or <path>/<dem>.vrt. The latter is an aggregation of the DEM-files available from the given DEM. The GTOPO30 vrt does not contain the Antarctic, because this is in polarstereographic projection.

You can create your own gtopo30.vrt file with gdal, e.g.:

```
gdalbuildvrt <dem>.vrt [E,W]*.DEM
```

Remember to update this mapper by adding allowed filenames to the list of accepted filenames (accepted_names) if you create or apply new DEM datasets.

3.2.55 nansat.mappers.mapper_viirs_l1 module

class nansat.mappers.mapper_viirs_l1.Mapper (filename, gdalDataset, gdalMetadata, GCP_COUNT0=5, GCP_COUNT1=20, pixelStep=1, lineStep=1, **kwargs)

Bases: nansat.vrt.VRT

3.2. nansat.mappers package 79
VRT with mapping of WKV for VIIRS Level 1B

### 3.2.56 nansat.mappers.obpg module

```python
class nansat.mappers.obpg.OBPGL2BaseClass(x_size=1, y_size=1, metadata=None, nomem=False, **kwargs)
Bases: nansat.vrt.VRT
Base Class for Mappers for SeaWIFS/MODIS/MERIS/VIIRS L2 data from OBPG
titles = ['HMODISA Level-2 Data', 'MODISA Level-2 Data', 'HMODIST Level-2 Data', 'MERIS Level-2 Data', 'GOCI Level-2 Data', 'VIIRSN Level-2 Data', 'SeaWiFS Level-2 Data']
```

### 3.2.57 nansat.mappers.opendap module

```python
class nansat.mappers.opendap.Opendap(x_size=1, y_size=1, metadata=None, nomem=False, **kwargs)
Bases: nansat.vrt.VRT
Methods for all OpenDAP mappers
P2S = {'D': 86400, 'H': 3600, 'M': 2592000, 'Y': 31536000}
create_vrt(filename, gdalDataset, gdalMetadata, date, ds, bands, cachedir)
Create VRT
get_dataset(ds)
Open Dataset
get_dataset_time()
Load data from time variable
get_geospatial_variable_names()
Get names of variables with both spatial dimentions
get_geotransform()
Get first two values of X,Y variables and create geoTranform
get_layer_datetime(date, datetimes)
Get datetime of the matching layer and layer number
get_metaitem(url, varName, layerNo)
Set metadata for creating band VRT
get_shape()
Get srcRasterXSize and srcRasterYSize from OpenDAP
get_time_coverage_resolution()
Try to feecth time_coverage_resolution and convert to seconds
test_mapper(filename)
Tests if filename fits mapper. May raise WrongMapperError
```

### 3.2.58 Module contents
Please acknowledge Nansat

We appreciate acknowledgments of Nansat. If you use Nansat in scientific publications, please add a reference to the following paper:

Differentiating between land and water

To add simple land- or water-masks to your figures, you can use the watermask() method in the main Nansat class. Download the prepared MODIS 250M water-mask product from our server and add the path to the directory with this data to an environment variable named MOD44WPATH (e.g. MOD44WPATH=/Data/sat/auxdata/mod44w).
Digital Elevation Models (DEMs)

6.1 Global Multi-resolution Terrain Elevation Data 2010 (GMTED2010)

The GMTED2010 datasets are provided by the U.S. Geological Survey. We have prepared a GDAL vrt file that can be used together with mapper_topography.py to open the 30 arcseconds Digital Elevation Model (DEM) with Nansat. To use it, the vrt file must be downloaded from ftp://ftp.nersc.no/nansat/dem/gmted2010_30.vrt and stored in the same folder as the tif files of mean elevation available at https://edcintl.cr.usgs.gov/downloads/sciweb1/shared/topo/downloads/GMTED/Global_tiles_GMTED/300darcsec/mea/.

In case you want to use a different DEM, the procedure is as follows:

1. Download the relevant GDAL readable files to a local folder
2. Generate a vrt file using gdalbuildvrt, e.g.:

   ```
   gdalbuildvrt gmted2010_30.vrt *.tif
   ```

3. Update mapper_topography.py to accept the new kind of file(s)

6.2 Global 30 Arc-Second Elevation (GTOPO30)

We have also created a vrt-file for the GTOPO30 dataset. This is available as ftp://ftp.nersc.no/nansat/dem/gtopo30.vrt. The vrt-file should be placed in the same folder as the .DEM files available at https://dds.cr.usgs.gov/srtm/version2_1/SRTM30/.
7.1 Git branching and merging

We adopt the following system for branching and merging:

1. **master** branch: numbered releases of the code. Never edited. Merged from **develop** and **hot fix** branches (see notes on workflow below). Long living.


4. **hotfix<NNN>_<short-heading>** - branches that are specific to a hotfix issue. Hotfixes are bugfixes on master that can not wait until next release.

**Note:**

1. Never edit code in the master or develop branch. Always make a new branch for your edits.
2. A new branch should be very specific to only one problem. It should be short living.
3. Commit often.
4. Branch often.
5. Branch only from master or from develop.
6. Create pull requests for your branches and **always** assign a reviewer to merge, delete the branch, and close the issue (this is easy in github)
7.2 How to report and handle new issues (bugs, improvements, new features, etc.)

If you discover a bug in Nansat or you would like to suggest improvements to Nansat, the following procedure should be followed:

1. Check that no one else has reported the same issue at https://github.com/nansencenter/nansat/issues
2. If not, add a new issue at https://github.com/nansencenter/nansat/issues
3. If Nansat repository is not accessible for writing, fork Nansat and clone your own fork locally
4. Create an issue branch on your local system named `issue<NNN>_<short-heading>` where NNN is the issue number from GitHub. This will be the main (short living) working area. The issue branch should originate from `develop`.
5. Add tests to reproduce the bug or test the new functionality
6. Write the necessary code
7. Push new issue branch to your own fork at GitHub
8. Create a pull request of the issue branch on your fork at GitHub. A member of our team will review the code, merge, delete the branch, and close the issue (this is easy in Github)

If a bug is relatively quick and easy to handle, we call it a hotfix. A hotfix is branched from master (by team members), and the following workflow applies (workflow for issue branches based on develop is similar):

1. Branch from master into the hotfix specific branch (hotfix<NNN>_<short-heading>)
1. Update the tests
2. Fix the bug
3. Increment micro version in `setup.py`
4. Commit to the hotfix branch
2. If needed rebase the hotfix branch on top of master:
1. Checkout the hotfix branch
2. Use `git rebase master`
3. Fix conflicts if any
4. Test the code
5. Push your hotfix branch to GitHub. Note: You have to use ‘git push -f’ in order to rewrite the history on github. Rebase will not cause problems if only one person works with the hotfix branch.
3. Go to https://github.com/nansencenter/nansat and add a pull request for the newly pushed hotfix branch and assign a reviewer
4. Let the reviewer do the following:
1. Wait for tests on Travis CI to pass
2. Check the code
3. Request changes or merge the pull request into master using ‘Rebase and Merge’ button in the online tool.
4. Delete the branch
5. Merge master into develop (`git checkout develop; git merge master; test the code; push to GitHub; check Travis CI status`)
6. Close the related issue

**Note:** If you work on a project using Nansat, this project should use the master version of Nansat. Two situations may then occur:

1. You discover a **bug**. This can be solved with a **hotfix**.
2. You want to add functionality to Nansat. This is solved by creating an issue in Nansat and a related issue branch. When this is completed and merged into develop, we may have a new release.
3. You want to add a mapper. This can be done by adding a package `nansat_mappers` to your project. When the mapper is completed, create an issue and an issue-branch in Nansat, and finally submit a pull request with suggestion of a reviewer. You can still use the mapper via your `nansat_mappers` package until the new mapper is implemented in a release version of Nansat.

### 7.3 General conventions

- Nansat coding style generally follows PEP-8 (General style guide) and PEP-257 (Docstrings)
- Max line length is set to 100 chars
- Every unit of code must be properly tested (see unit-test) and documented
- All class/function/method/variable names have to be explicit and should contain no more than 3 words
- Single quotes should be used consistently instead of double quotes (except for cases where quotes are required, and for docstrings)
- GNU v3 licence should be inserted in all files. Mappers should have a standard header like this:

```
# Name:       mapper_asar.py  
# Purpose:    Mapper for Envisat/ASAR data  
# Authors:    Asuka Yamakava, Anton Korosov  
# Licence:    This file is part of NANSAT. You can redistribute it or modify  
#             under the terms of GNU General Public License, v.3  
#             http://www.gnu.org/licenses/gpl-3.0.html  
#  
# Additional mapper/format specific links and information
```

- Docstrings should follow the Numpy style
- Available headers are `Parameters`, `Returns`, `Other parameters`, `Modifies`, `Crates`, `Raises`, `See also`, `Notes`, `References` and `Examples`

#### 7.3.1 Example function with complete Docstring

```python
def some_function(start = 0, stop, step = 1):
    """ Return evenly spaced values within a given interval.

    | Values are generated within the half-open interval '[start, stop)'  
    | (in other words, the interval including 'start' but excluding 'stop').  
    | For integer arguments the function is equivalent to the Python built-in  
    | 'range'_ function, but returns a ndarray rather than a list.

    Parameters
```

(continues on next page)
start : number, optional
    Start of interval. The interval includes this value. The default start value is 0.
stop : number
    End of interval. The interval does not include this value.
step : number, optional
    Spacing between values. For any output 'out', this is the distance between two adjacent values, '
    'out[i+1] - out[i]''. The default step size is 1. If 'step' is specified, 'start' must also be given.
dtype : dtype
    The type of the output array. If 'dtype' is not given, infer the data type from the other input arguments.

Returns
-------
out : ndarray
    Array of evenly spaced values.

For floating point arguments, the length of the result is '
    ''ceil((stop - start)/step)''. Because of floating point overflow, this rule may result in the last element of 'out' being greater than 'stop'.

Modifies
-------
self.vrt : VRT
    Dataset RasterXSize and RasterYSize are changed in the current VRT dataset.

See Also
--------
linspace : Evenly spaced numbers with careful handling of endpoints
ogrid: Arrays of evenly spaced numbers in N-dimensions
mgrid: Grid-shaped arrays of evenly spaced numbers in N-dimensions

Examples
--------
>>> np.arange(3)
array([0, 1, 2])

7.4 Naming conventions

- when a variable points to the GDALDataset, GDALDriver, etc. its name must always contain word “dataset”, “driver”, etc. representatively (raw_dataset, src_dataset, example_driver)
- when a variable points to a string with name it should contain ‘name’ (band_name)
- when longitude and latitude are input to (or output from) a function, they should be given in this order: (lon, lat). These variables should always be named ‘lon’ and ‘lat’ (i.e. never ‘long’).
- source and destination are prefixed as ‘src’ and ‘dst’ (src_dataset, dst_raster_xsize)
- band numbers should be called ‘band_number’
- GDAL bands should be called ‘band’ or, e.g., ‘dst_band’ when prefixed (GDAL is actually in-consistent here:
Nansat Documentation

gdal.Dataset.GetRasterBand returns a ‘Band’-object; hence ‘Band’ is the name of the class and the Python datatype
• We use ‘filename’ (as in Python standard library)

7.5 Style checking

In your IDE/editor, it is highly recommended to activate/install a plugin for/script a save hook for doing automatic style checks and/or corrections, eg autopep8, pylint, pyflakes.

7.6 Tests

In general:
• Every function must be accompanied with a test suite
• Tests should be both positive (testing that the function work as intended with valid data) and negative (testing that the function behaves as expected with invalid data e.g. that correct exceptions are thrown)
• If a function has optional arguments, separate tests for all options should be created

7.6.1 Testing core Nansat functionality

• Tests for Nansat, Domain, etc should be added to nansat/tests/test_<module_name>.py file;
• These tests should be added as functions of classes inheriting from unittest.TestCase (e.g. DomainTest);
• Tests sharing similar set-up may inherit from the same class which has a setUp function;
• The core tests are run at Travis CI (continuous integration) which integrates with Coveralls for providing test coverage

7.6.2 Integration testing

Products read by Nansat mappers are tested in modules within the nansat_integration_tests folder in the repository root. These tests should have access to all the kinds of data read by nansat. Since this is a very large amount of data, and since we cannot share every data product openly, these tests are not presently executed at Travis CI. Every developer should add new end-to-end tests and execute them when new mappers or workflows are added. Unavailable test data will lead to fewer tests being executed, i.e. they won’t fail because of missing data. If possible, datasets used in new tests should be made available to the Nansen Center such that we can run the full test suite.

7.6.3 Testing mappers

General tests checking that the mappers don’t violate the functionality of nansat and checks that some specific metadata is added, are collected in the nansat_integration_tests.mapper_tests module.

Also, we aim to create proper unit tests that use mock object for all the mappers. This will help to significantly increase the test coverage.
7.6.4 Testing specific data products or workflows

In typical scientific workflows, a data product is opened with Nansat and some operations are performed, e.g., adding new derived bands and exporting the results to a netcdf, or creating figures. To make sure that new versions of nansat do not harm these workflows with bugs or sudden interface changes, we collect tests for typical workflows in separate modules within the nansat_integration_tests package, e.g. test_sar, test_radarsat2, etc. We encourage users and developers to add such tests to avoid such potential problems.

7.6.5 Doctests

TODO: add information about how to use doctests
8.1 Concept

GDAL can read most satellite EO and NetCDF-CF compliant raster data relevant for Earth sciences. However, GDAL does not attach meaning to the contained information, i.e., it does not specify what kind of data is contained in a given band, e.g., if it is water-leaving-radiance used for monitoring of water quality. Nansat provides a mapping between geophysical variables of known meaning and the raster bands provided in the “Datasets” returned by GDAL.

The modules within the mappers package each contain a class defining the mapping between the bands returned from GDAL and metadata vocabularies provided via the py-thesaurus-interface package. For example, the simplest mapper for Meris level-1 data explicitly states that the first 15 bands are upwelling_spectral_radiance at 15 wavelengths and that the 16th band contains quality flags. The description of these bands require some compulsory metadata attributes to be defined in the mapper. These attributes follow certain given conventions:

- NetCDF-CF
- NASA GCMD keywords
- nersc-vocabularies

By using these conventions, the mappers thus attach unambiguous geophysical meaning to variables following the given standards. This allows the user to open and use a geo-refenced raster dataset with Nansat without depending on detailed apriori knowledge about the origin or type of the data.

8.2 Workflow

When we open a file with Nansat:

```python
#!/usr/bin/env python
n = Nansat(filename)
```

these steps follow:
The Nansat constructor calls gdal.Open(filename) to open the file with GDAL, and returns a GDAL Dataset with a list of available raster bands.

The Nansat constructor loops through available mappers and parses the Dataset to the mapper.

- Each mapper checks if the input Dataset is appropriate for the mapper, i.e., if the format, the metadata and the set of bands in the Dataset corresponds to what is expected in the mapper.
  - If the Dataset is not valid, the mapper silently fails and the next mapper is tested.
  - If the Dataset fits the mapper:
    - the mapper creates a **GDAL VRT file** with georeference and raster bands corresponding to the “well known variables” in nersc-vocabularies and adds respective metadata to each band (standard_name, units, etc).

- The mapper object, which is an instance of the VRT-class, is then returned to the Nansat instance as an attribute named **vrt** (Nansat.vrt).

The VRT has the following properties:

- we can use any available GDAL API functions, e.g., warping or exporting
- it contains georeferencing recognised by GDAL
- we can add PixelFunctions for, e.g., calculation of speed given two vector components of wind or current
- still it contains only Raster Bands with metadata which correspond to any of the NANSAT “Well Known Variables”

The Dataset may, e.g., be subsetted, reprojected, merged, etc., by simply modifying the VRT-file, either automatically by the GDAL high level applications/functions, or with NANSAT-specific Python logic. An important benefit of this approach is that we employ the **lazy processing concept** in GDAL.

**Note:** No processing or file reading/writing is performed before it is needed.

The VRT file defines a set of operations in xml format. When information is needed, data is extracted as numpy arrays for further processing or plotting. As such, we basically use the GDAL Datamodel and do not need to design our own.

### 8.3 Technical details

- The VRT-file is stored in memory using GDAL VSI-approach
- The VRT-class is a wrapper around the VRT-file. It has methods for generating, modifying, copying and other operations with VRT-files. VRT-class uses both GDAL methods and direct writing for modifying the VRT-file.
- Each mapper inherits the VRT-class.

### 8.4 Where to put new mappers?

If you have created a new mapper, you can either submit a pull request for inclusion in the nansat mappers package, or you can make a namespace package to let nansat discover your mapper automatically. This is done the following way:

1. Create a directory called `nansat_mappers` within a directory on your $PYTHONPATH
2. Inside `nansat_mappers`, create the file `__init__.py` with the following lines:
3. Add your mapper module (the filename should start with `mapper_` and end with `.py`) to the `nansat_mappers` folder.

4. Reload your shell and start Python.

5. Nansat should now find your mapper.

Note that user defined mappers have higher priority than standard mappers.

### 8.5 Required metadata added in the mappers

- TODO: add list of required metadata

### 8.6 Adding mapper tests

TODO: add documentation about how to write mapper tests
CHAPTER 9

Releasing Nansat

9.1 General release procedure

In setup.py, Make sure the version is correct, by checking MAJOR, MINOR and MICRO, and that ‘dev’ is removed from VERSION.

Releases should be made from the master branch. Make sure that Nansat works on all operating systems (Windows, Linux and OS X), with all combinations of python (py27, py35, py36) before releasing.

When you are ready to release a new version, create a tag for the release and push it. The following creates a tag for version 0.6.17 and pushes it.

```
git tag "v0.6.17"
git push origin --tags
```

Go to https://github.com/nansencenter/nansat/releases and click “Draft a new release”.

Select the tag version you just created, create a release title, for consistency, use the format of Nansat-0.6.17

Write some release notes that describes the changes done in this release.

9.2 Releasing on PiPy

First, wait until Nansat passes all tests on Travi-CI, Appveyor and Coverals. Then execute:

```
conda create -n release_nansat -c conda-forge -y python pythesint scipy basemap
   →netcdf4 gdal pillow mock nose urllib3 twine
source activate release_nansat
python setup.py sdist
# Check the dist file that was just created
ls dist
# Should be a file on this format 'nansat-1.0.20.tar.gz'
twine upload dist/nansat-1.0.20.tar.gz
```
Packaging documentation is found at PyPA Packaging and Distributing Projects
To avoid having to enter password when uploading, you can set $HOME/.pypirc as described in the above link.

9.3 Releasing on Anaconda

We are releasing Nansat through the conda-forge channel on Anaconda. First, wait until Nansat passes all tests on Travi-CI, Appveyor and Coverals. Then execute:

```
# install (or update) conda-smithy
conda install -n root -c conda-forge conda-smithy
git clone git@github.com:conda-forge/nansat-feedstock.git
cd nansat-feedstock
conda smithy rerender -c auto
git push
```

Information how to use Conda-Smithy can be found at The tool for managing conda-forge feedstocks
Documenting Nansat

Documentation should follow the conventions.

**Note:** Documentation for classes should be given after the class definition, not within the `__init__`-method.

To build documentation locally, the best is to create a virtual environment with the sphinx environment installed. This is done as follows:

```
cd docs
conda env create -n build_docs --file environment.yml
source activate build_docs
```

Then, the following commands should build the documentation:

```
make clean
sphinx-apidoc -o source/ ../nansat
make html
```

Some documentation remains to be written. This is marked by `TODO` in the rst source files. Find open tasks by:

```
cd docs/source
grep TODO *
```
Indices and tables

- genindex
- modindex
- search
n
nansat, 46
nansat.domain, 13
nansat.exceptions, 18
nansat.exporter, 18
nansat.figure, 20
nansat.geolocation, 25
nansat.mappers, 80
nansat.mappers.envisat, 65
nansat.mappers.globcolour, 67
nansat.mappers.hdf4_mapper, 67
nansat.mappers.mapper_aapp_l1b, 68
nansat.mappers.mapper_aapp_l1c, 68
nansat.mappers.mapper_amsr2_l1r, 68
nansat.mappers.mapper_amsre_uham_leadfraction, 68
nansat.mappers.mapper_arome, 69
nansat.mappers.mapper_asar, 69
nansat.mappers.mapper_ascat_nasa, 69
nansat.mappers.mapper_aster_l1a, 69
nansat.mappers.mapper_aster_l1b, 69
nansat.mappers.mapper_case2reg, 69
nansat.mappers.mapper_cmems, 70
nansat.mappers.mapper_csks, 70
nansat.mappers.mapper_ecmwf_metno, 70
nansat.mappers.mapper_emodnet, 70
nansat.mappers.mapper_generic, 70
nansat.mappers.mapper_globe, 70
nansat.mappers.mapper_globe_13b, 70
nansat.mappers.mapper_globe_13m, 71
nansat.mappers.mapper_goci_l1, 71
nansat.mappers.mapper_hirlam, 71
nansat.mappers.mapper_hirlam_wind_netcdf, 71
nansat.mappers.mapper_kmss, 71
nansat.mappers.mapper_landsat, 71
nansat.mappers.mapper_meris_l1, 71
nansat.mappers.mapper_meris_l2, 72
nansat.mappers.mapper_metno_hires_seaice, 72
nansat.mappers.mapper_mod44w, 72
nansat.mappers.mapper_modis_l1, 72
nansat.mappers.mapper_nce, 72
nansat.mappers.mapper_nce_wind, 72
nansat.mappers.mapper_nce_wind_online, 73
nansat.mappers.mapper_netcdf_cf, 73
nansat.mappers.mapper_nora10_local_vpv, 73
nansat.mappers.mapper_obpg_l1, 73
nansat.mappers.mapper_obpg_l2 nc, 73
nansat.mappers.mapper_obpg_l3, 73
nansat.mappers.mapper_ocean_productivity, 74
nansat.mappers.mapper_opendap_arome, 74
nansat.mappers.mapper_opendap_globcurrent, 74
nansat.mappers.mapper_opendap_globcurrent_thredds, 75
nansat.mappers.mapper_opendap_occ, 75
nansat.mappers.mapper_opendap_osisaf, 76
nansat.mappers.mapper_opendap_siwtacsst, 76
nansat.mappers.mapper_opendap_sstcci, 76
nansat.mappers.mapper_pathfinder52, 77
nansat.mappers.mapper_radsat2, 77
nansat.mappers.mapper_sentinel1_l1, 77
nansat.mappers.mapper_sentinel1_l2, 79
nansat.mappers.mapper_topography, 79
nansat.mappers.mapper_virs_l1, 79
nansat.mappers.obpg, 80
nansat.mappers.opendap, 80
nansat.nansat, 25
nansat.node, 34
nansat.nsr, 36
nansat_pointbrowser, 36
nansat_tools, 37
nansat_vrt, 39
nansat_warnings, 46
Nansat Documentation

filename (nansat.nansat.Nansat attribute), 29
FILENAME (nansat.nansat.Nansat attribute), 52
FILENAME (nansat.nansat.Nansat attribute), 52
find_metadata() (nansat.mappers.hdf4_mapper.HDF4Mapper method), 67
fix_band_metadata() (nansat.vrt.VRT method), 41
fix_global_metadata() (nansat.vrt.VRT method), 41
fmt (nansat.pointbrowser.PointBrowser attribute), 36
fontRatio (nansat.figure.Figure attribute), 23
tFontSize (nansat.figure.Figure attribute), 23
tFontSize (nansat.figure.Figure attribute), 23
tfreqs (nansat.mappers.mapper_amsr2_l3.Mapper attribute), 68
from_array() (nansat.vrt.VRT class method), 41
from_dataset() (nansat.geolocation.Geolocation class method), 25
from_dataset_params() (nansat.geolocation.Geolocation class method), 25
from_domain() (nansat.vrt.VRT class method), 42
from_domain() (nansat.vrt.VRT class method), 42
from_filenames() (nansat.geolocation.Geolocation class method), 25
from_gdal_dataset() (nansat.vrt.VRT class method), 42
from_lonlat() (nansat.domain.Domain method), 15
from_lonlat() (nansat.domain.Domain method), 15
from_lonlat() (nansat.vrt.VRT class method), 42
get_ads_vrts() (nansat.mappers.envisat.Envisat method), 66
get_array_from_ADS() (nansat.mappers.envisat.Envisat method), 66
gem_band_number() (nansat.nansat.Nansat method), 55
gem_band_number() (nansat.nansat.Nansat method), 29
gem_border() (nansat.domain.Domain method), 48
gem_border_geojson() (nansat.domain.Domain method), 49
gem_border_geojson() (nansat.domain.Domain method), 49
gem_border_geometry() (nansat.domain.Domain method), 49
gem_border_geometry() (nansat.domain.Domain method), 49
gem_border_postgis() (nansat.domian.Domain method), 49
gem_border_postgis() (nansat.domain.Domain method), 49
gem_border_wkt() (nansat.domain.Domain method), 49
gem_border_wkt() (nansat.domain.Domain method), 49
gem_corners() (nansat.domain.Domain method), 49
gem_corners() (nansat.domain.Domain method), 16
gem_dataset() (nansat.mappers.opendap.Opendap method), 80
gem_dataset_time() (nansat.mappers.opendap.Opendap method), 80
gem_gcmd_keywords_mapping() (in module nansat.mappers.mapper_cmems), 70
gem_GDALRasterBand() (nansat.Nansat method), 55
gem_GDALRasterBand() (nansat.nansat.Nansat method), 29
gem_geolocation_grids() (nansat.nansat.Nansat method), 49
gem_geolocation_grids() (nansat.domain.Domain method), 16
gem_geolocation_grids() (nansat.geolocation.Geolocation method), 25
gem_geospatial_variable_names() (nansat.mappers.opendap.Opendap method), 80
gem_geotransform() (nansat.mappers.opendap.Opendap method), 80
gem_layer_dateratetime() (nansat.mappers.opendap.Opendap method), 80
gem_metadata() (nansat.Nansat method), 55
gem_metadata() (nansat.nansat.Nansat method), 30
gem_metaitem() (nansat.mappers.opendap.Opendap method), 80
gem_min_max_lon_lat() (nansat.domain.Domain method), 49
gem_min_max_lon_lat() (nansat.domain.Domain method), 16
gem_pixelsize_meters() (nansat.domain.Domain method), 49
gem_pixelsize_meters() (nansat.domain.Domain method), 16
gem_points() (nansat.pointbrowser.PointBrowser method), 36
gem_projection() (nansat.vrt.VRT method), 42
gem_random_color() (in module nansat.tools), 37
gem_resized_vrt() (nansat.vrt.VRT method), 43
gem_shape() (nansat.mappers.opendap.Opendap method), 80
gem_shifted_vrt() (nansat.vrt.VRT method), 43
gem_sub_vrt() (nansat.vrt.VRT method), 43
gem_subsampled_vrt() (nansat.vrt.VRT method), 43
gem_super_vrt() (nansat.vrt.VRT method), 43
gem_time_coverage_resolution() (nansat.mappers.opendap.Opendap method), 80
gem_transect() (nansat.Nansat method), 56
gem_transect() (nansat.nansat.Nansat method), 30
gem warped_vrt() (nansat.vrt.VRT method), 30
gemAttribute() (nansat.node.Node method), 35
gemAttributeList() (nansat.node.Node method), 35
Globcolour (class in nansat.mappers.globcolour), 67

Index

107
H

hardcopy_bands() (nansat.vrt.VRT method), 44
has_band() (nansat.Nansat method), 56
has_band() (nansat.nansat.Nansat method), 30
haversine() (in module nansat.tools), 37
HDF4Mapper (class in nansat.mappers.hdf4_mapper), 67

I

init_from_xml() (nansat.mappers.mapper_radarsat2.Mapper method), 77
initial_bearing() (in module nansat.tools), 37
insert() (nansat.node.Node method), 35
intersects() (nansat.Domain method), 50
intersects() (nansat.domain.Domain method), 16

K

KML_BASE (nansat.Domain attribute), 48
KML_BASE (nansat.domain.Domain attribute), 14

L

latGrid (nansat.Figure attribute), 64
latGrid (nansat.figure.Figure attribute), 23
latTicks (nansat.figure.Figure attribute), 64
latTicks (nansat.figure.Figure attribute), 23
leave_few_bands() (nansat.vrt.VRT method), 44
legend (nansat.Figure attribute), 64
legend (nansat.figure.Figure attribute), 23
LEGEND_HEIGHT (nansat.Figure attribute), 61
LEGEND_HEIGHT (nansat.figure.Figure attribute), 20
lines (nansat.pointbrowser.PointBrowser attribute), 36
list_bands() (nansat.nansat.Nansat method), 56
list_bands() (nansat.nansat.Nansat method), 30
logarithm (nansat.Figure attribute), 64
logarithm (nansat.figure.Figure attribute), 23
logger (nansat.Domain attribute), 50
logger (nansat.domain.Domain attribute), 16
logger (nansat.Nansat attribute), 56
logger (nansat.nansat.Nansat attribute), 31
logger (nansat.vrt.VRT attribute), 44
logoFileName (nansat.Figure attribute), 64
logoFileName (nansat.figure.Figure attribute), 23
logoLocation (nansat.Figure attribute), 64
logoLocation (nansat.figure.Figure attribute), 24
logoSize (nansat.Figure attribute), 64
logoSize (nansat.figure.Figure attribute), 24
lonGrid (nansat.Figure attribute), 64
lonGrid (nansat.figure.Figure attribute), 24
lonlatNames (nansat.mappers.envisat.Envisat attribute), 66
lonTicks (nansat.Figure attribute), 64
lonTicks (nansat.figure.Figure attribute), 24

M

make_rsw_meta_entry()
(nansat.mappers.globcolour.Globcolour method), 67
Mapper (class in nansat.mappers.mapper_aapp_11b), 68
Mapper (class in nansat.mappers.mapper_aapp_11c), 68
Mapper (class in nansat.mappers.mapper_amsr2_1lr), 68
Mapper (class in nansat.mappers.mapper_amsr2_1l3), 68
Mapper (class in nansat.mappers.mapper_amsre_uham_leadfraction), 68
Mapper (class in nansat.mappers.mapper_arome), 69
Mapper (class in nansat.mappers.mapper_asar), 69
Mapper (class in nansat.mappers.mapper_aster_l1a), 69
Mapper (class in nansat.mappers.mapper_aster_l1b), 69
Mapper (class in nansat.mappers.mapper_case2reg), 69
Mapper (class in nansat.mappers.mapper_cmems), 70
Mapper (class in nansat.mappers.mapper_csk), 70
Mapper (class in nansat.mappers.mapper_ecmwf_metno), 70
Mapper (class in nansat.mappers.mapper_emodnet), 70
Mapper (class in nansat.mappers.mapper_generic), 70
Mapper (class in nansat.mappers.mapper_globcolour_l3b), 70
Mapper (class in nansat.mappers.mapper_globcolour_l3m), 71
Mapper (class in nansat.mappers.mapper_goci_11), 71
Mapper (class in nansat.mappers.mapper_hirlaml), 71
Mapper (class in nansat.mappers.mapper_hirlam_wind_netcdf), 71
Mapper (class in nansat.mappers.mapper_kmss), 71
Mapper (class in nansat.mappers.mapper_landsat), 71
Mapper (class in nansat.mappers.mapper_meris_11), 71
Mapper (class in nansat.mappers.mapper_meris_12), 72
Mapper (class in nansat.mappers.mapper_metno_hires_seaice), 72
Mapper (class in nansat.mappers.mapper_mod44w), 72
Mapper (class in nansat.mappers.mapper_modis_11), 72
Mapper (class in nansat.mappers.mapper_ncep), 72
Mapper (class in nansat.mappers.mapper_ncep_wind), 72
Mapper (class in nansat.mappers.mapper_ncep_wind_online), 73
Mapper (class in nansat.mappers.mapper_netcdf_cf), 73
Mapper (class in nansat.mappers.mapper_nora10_local_vpv), 73
Mapper (class in nansat.mappers.mapper_obpg_l2), 73
Mapper (class in nansat.mappers.mapper_obpg_l2_nc), 73
Mapper (class in nansat.mappers.mapper_obpg_l3), 73
Mapper (class in nansat.mappers.mapper_ocean_productivity), 74
Mapper (class in nansat.mappers.mapper_opendap_arome), 74
Mapper (class in nansat.mappers.mapper_opendap_globcurrent), 74
Mapper (class in nansat.mappers.mapper_opendap_globcurrent_thredds), 75
Mapper (class in nansat.mappers.mapper_opendap_occci), 75
Mapper (class in nansat.mappers.mapper_opendap_osisaf), 76
Mapper (class in nansat.mappers.mapper_opendap_siwtacsst), 76
Mapper (class in nansat.mappers.mapper_opendap_sstcci), 76
Mapper (class in nansat.mappers.mapper_pathfinder52), 77
Mapper (class in nansat.mappers.mapper_radarsat2), 77
Mapper (class in nansat.mappers.mapper_sentinel1_l1), 77
Mapper (class in nansat.mappers.mapper_sentinel1_l2), 79
Mapper (class in nansat.mappers.mapper_topography), 79
Mapper (class in nansat.mappers.mapper_viirs_l1), 79
mapper (nansat.Nansat attribute), 57
mask_array (nansat.Figure attribute), 64
mask_lut (nansat.Figure attribute), 24

N
name (nansat.Domain attribute), 50
name (nansat.domain.Domain attribute), 16
name (nansat.Nansat attribute), 57
name (nansat.nansat.Nansat attribute), 31
Nansat (class in nansat), 51
Nansat (class in nansat.nansat), 25
nansat (module), 46
nansat.domain (module), 13
nansat.exceptions (module), 18
nansat.exporter (module), 18
nansat.figure (module), 20
nansat.geolocation (module), 25
nansat.mappers (module), 80
nansat.mappers.envisat (module), 65
nansat.mappers.globcolour (module), 67
nansat.mappers.hdf4_mapper (module), 67
nansat.mappers.mapper_aapp_l1b (module), 68
nansat.mappers.mapper_aapp_l1c (module), 68
nansat.mappers.mapper_amsr2_l1r (module), 68
nansat.mappers.mapper_amsr2_l3 (module), 68
nansat.mappers.mapper_amsrre_uham_leadfraction (module), 68
nansat.mappers.mapper_arome (module), 69
nansat.mappers.mapper_asar (module), 69
nansat.mappers.mapper_ascat_nasa (module), 69
nansat.mappers.mapper_aster_l1a (module), 69
nansat.mappers.mapper_aster_l1b (module), 69
nansat.mappers.mapper_case2reg (module), 69
nansat.mappers.mapper_cmems (module), 70
nansat.mappers.mapper_csk (module), 70
nansat.mappers.mapper_ecmwf_metno (module), 70
nansat.mappers.mapper_emodnet (module), 70
nansat.mappers.mappergeneric (module), 70
nansat.mappers.mapper_globcolour_l3b (module), 70
nansat.mappers.mapper_globcolour_l3m (module), 71
nansat.mappers.mapper_goci_l1 (module), 71
nansat.mappers.mapper_hirlam (module), 71
nansat.mappers.mapper_hirlam_wind_netcdf (module), 71
nansat.mappers.mapper_kmss (module), 71
nansat.mappers.mapper_landsat (module), 71
nansat.mappers.mapper_meris_l1 (module), 71
nansat.mappers.mapper_meris_l2 (module), 72
nansat.mappers.mapper_metno_hires_seaice (module), 72
nansat.mappers.mapper_mod44w (module), 72
nansat.mappers.mapper_modis_l1 (module), 72
nansat.mappers.mapper_ncep (module), 72
nansat.mappers.mapper_ncep_wind (module), 72
nansat.mappers.mapper_ncep_wind_online (module), 73
nansat.mappers.mapper_netcdf Cf (module), 73
nansat.mappers.mapper_nora10_local_vpv (module), 73
nansat.mappers.mapper_obpg_l2 (module), 73
nansat.mappers.mapper_obpg_l2_nc (module), 73
nansat.mappers.mapper_obpg_l3 (module), 73
nansat.mappers.mapper_ocean_productivity (module), 74
nansat.mappers.mapper_opendap_arome (module), 74
nansat.mappers.mapper_opendap_globcurrent (module), 74
nansat.mappers.mapper_opendap_globcurrent_thredds (module), 75
nansat.mappers.mapper_opendap_occci (module), 75
nansat.mappers.mapper_opendap_osisaf (module), 76
nansat.mappers.mapper_opendap_siwtacsst (module), 76
nansat.mappers.mapper_opendap_sstcci (module), 76
nansat.mappers.mapper_pathfinder52 (module), 77
nansat.mappers.mapper_radarsat2 (module), 77
nansat.mappers.mapper_sentinel1_l1 (module), 77
nansat.mappers.mapper_sentinel1_l2 (module), 79
nansat.mappers.mapper_topography (module), 79
nansat.mappers.mapper_viirs_l1 (module), 79
nansat.mappers.obpg (module), 80
nansat.mappers.opendap (module), 80
nansat.nansat (module), 25
nansat.node (module), 34
nansat.nsr (module), 36
nansat.pointbrowser (module), 36
nansat.tools (module), 37
<table>
<thead>
<tr>
<th>Module/Class/Function/Attribute</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nansat.vrt (module)</td>
<td>39</td>
</tr>
<tr>
<td>nansat.warnings (module)</td>
<td>46</td>
</tr>
<tr>
<td>NansatFutureWarning</td>
<td>18</td>
</tr>
<tr>
<td>NansatGeolocationError</td>
<td>18</td>
</tr>
<tr>
<td>NansatMissingProjectionError</td>
<td>18</td>
</tr>
<tr>
<td>NansatProjectionError</td>
<td>18</td>
</tr>
<tr>
<td>NansatReadError</td>
<td>18</td>
</tr>
<tr>
<td>Node (class in nansat.node)</td>
<td>34</td>
</tr>
<tr>
<td>node() (nansat.node.Node method)</td>
<td>35</td>
</tr>
<tr>
<td>nodeList() (nansat.node.Node method)</td>
<td>35</td>
</tr>
<tr>
<td>NSR (in module nansat)</td>
<td>46</td>
</tr>
<tr>
<td>numOfColor (nansat Figure attribute)</td>
<td>46</td>
</tr>
<tr>
<td>numOfColor (nansat.figure Figure attribute)</td>
<td>24</td>
</tr>
<tr>
<td>numOfTicks (nansat Figure attribute)</td>
<td>46</td>
</tr>
<tr>
<td>numOfTicks (nansat.figure Figure attribute)</td>
<td>24</td>
</tr>
<tr>
<td>OBPG12BaseClass (class in nansat.mappers.obpg)</td>
<td>80</td>
</tr>
<tr>
<td>onclick() (nansat.pointbrowser.PointBrowser method)</td>
<td>36</td>
</tr>
<tr>
<td>Opendap (class in nansat.mappers.opendap)</td>
<td>80</td>
</tr>
<tr>
<td>OUTPUT_SEPARATOR (nansat.Domain attribute)</td>
<td>48</td>
</tr>
<tr>
<td>OUTPUT_SEPARATOR (nansat.domain Domain attribute)</td>
<td>14</td>
</tr>
<tr>
<td>overlaps() (nansat.Domain method)</td>
<td>50</td>
</tr>
<tr>
<td>overlaps() (nansat.domain Domain method)</td>
<td>16</td>
</tr>
<tr>
<td>P2S (nansat.mappers.opendap.Opendap attribute)</td>
<td>80</td>
</tr>
<tr>
<td>palette (nansat Figure attribute)</td>
<td>64</td>
</tr>
<tr>
<td>palette (nansat.figure Figure attribute)</td>
<td>24</td>
</tr>
<tr>
<td>param2wkv (nansat.mappers.mapper_obpg_l3.Mapper attribute)</td>
<td>74</td>
</tr>
<tr>
<td>param2wkv (nansat.mappers.mapper_ocean_productivity.Mapper attribute)</td>
<td>74</td>
</tr>
<tr>
<td>parse_time() (in module nansat.tools)</td>
<td>37</td>
</tr>
<tr>
<td>path (nansat.Nansat attribute)</td>
<td>57</td>
</tr>
<tr>
<td>path (nansat.nansat.Nansat attribute)</td>
<td>31</td>
</tr>
<tr>
<td>pilling (nansat Figure attribute)</td>
<td>64</td>
</tr>
<tr>
<td>pilling (nansat.figure Figure attribute)</td>
<td>24</td>
</tr>
<tr>
<td>pillingLegend (nansat.figure Figure attribute)</td>
<td>64</td>
</tr>
<tr>
<td>pillingLegend (nansat.figure Figure attribute)</td>
<td>24</td>
</tr>
<tr>
<td>PointBrowser (class in nansat.pointbrowser)</td>
<td>36</td>
</tr>
<tr>
<td>points (nansat.pointbrowser.PointBrowser attribute)</td>
<td>37</td>
</tr>
<tr>
<td>prepare_export_gif() (nansat.vrtVRT method)</td>
<td>44</td>
</tr>
<tr>
<td>prepare_export_netcdff() (nansat.vrtVRT method)</td>
<td>45</td>
</tr>
<tr>
<td>process() (nansat Figure method)</td>
<td>64</td>
</tr>
<tr>
<td>process() (nansat.figure Figure method)</td>
<td>24</td>
</tr>
<tr>
<td>ratio (nansat Figure attribute)</td>
<td>65</td>
</tr>
<tr>
<td>ratio (nansat.figure Figure attribute)</td>
<td>24</td>
</tr>
<tr>
<td>RAW_RASTER_BAND_SOURCE_XML (nansat.vrt.VRT attribute)</td>
<td>39</td>
</tr>
<tr>
<td>rawxml() (nansat.node.Node method)</td>
<td>35</td>
</tr>
<tr>
<td>read_annotation() (nansat.mappers.mapper_sentinel1_l1.Mapper method)</td>
<td>78</td>
</tr>
<tr>
<td>read_binary_line() (nansat.mappers.sentinel1_l1.Mapper method)</td>
<td>67</td>
</tr>
<tr>
<td>read_calibration() (nansat.mappers.mapper_sentinel1_l1.Mapper method)</td>
<td>78</td>
</tr>
<tr>
<td>read_manifest_data() (nansat.mappers.mapper_sentinel1_l1.Mapper method)</td>
<td>78</td>
</tr>
<tr>
<td>read_offset_from_header() (nansat.mappers.sentinel1_l1.Mapper method)</td>
<td>67</td>
</tr>
<tr>
<td>read_scaling_gads() (nansat.mappers.sentinel1_l1.Mapper method)</td>
<td>67</td>
</tr>
<tr>
<td>read_vsi() (nansat.vrt.VRT static method)</td>
<td>45</td>
</tr>
<tr>
<td>register_colormaps() (in module nansat.tools)</td>
<td>38</td>
</tr>
<tr>
<td>remove_keys() (in module nansat.tools)</td>
<td>38</td>
</tr>
<tr>
<td>reparse_projection() (nansat.mappers.mapper_generic.Mapper method)</td>
<td>70</td>
</tr>
<tr>
<td>replaceAttribute() (nansat.node.Node method)</td>
<td>36</td>
</tr>
<tr>
<td>replaceNode() (nansat.node.Node method)</td>
<td>36</td>
</tr>
<tr>
<td>replaceTag() (nansat.node.Node method)</td>
<td>36</td>
</tr>
<tr>
<td>reproject() (nansat.Nansat method)</td>
<td>57</td>
</tr>
<tr>
<td>reproject() (nansat.nansat.Nansat method)</td>
<td>31</td>
</tr>
<tr>
<td>reproject_gcps() (nansat.Domain method)</td>
<td>50</td>
</tr>
<tr>
<td>reproject_gcps() (nansat.domains Domain method)</td>
<td>17</td>
</tr>
<tr>
<td>reproject_gcps() (nansat.vrt.VRT method)</td>
<td>45</td>
</tr>
<tr>
<td>REPROJECT_TRANSFORMER (nansat.vrt.VRT attribute)</td>
<td>39</td>
</tr>
<tr>
<td>resize() (nansat.Nansat method)</td>
<td>57</td>
</tr>
<tr>
<td>resize() (nansat.nansat.Nansat method)</td>
<td>31</td>
</tr>
<tr>
<td>save() (nansat.figure Figure method)</td>
<td>24</td>
</tr>
<tr>
<td>set_metadata() (nansat.Nansat method)</td>
<td>58</td>
</tr>
<tr>
<td>set_metadata() (nansat.nansat.Nansat method)</td>
<td>32</td>
</tr>
<tr>
<td>set_offset_size() (nansat.vrt.VRT method)</td>
<td>45</td>
</tr>
<tr>
<td>setAttribute() (nansat.node.Node method)</td>
<td>36</td>
</tr>
<tr>
<td>setup_ads_parameters() (nansat.mappers.sentinel1_l1.Mapper method)</td>
<td>67</td>
</tr>
<tr>
<td>shape() (nansat.Domain method)</td>
<td>50</td>
</tr>
<tr>
<td>shape() (nansat.domain Domain method)</td>
<td>17</td>
</tr>
<tr>
<td>shift_cropped_gcps() (nansat.vrt.VRT method)</td>
<td>45</td>
</tr>
<tr>
<td>shift_cropped_geo_transform() (nansat.vrt.VRT method)</td>
<td>45</td>
</tr>
<tr>
<td>split_complex_bands() (nansat.vrt.VRT method)</td>
<td>45</td>
</tr>
<tr>
<td>srcDSProjection (nansat.mappers.opendap_globcurrent.Mapper attribute)</td>
<td>74</td>
</tr>
<tr>
<td>srcDSProjection (nansat.mappers.opendap_globcurrent_thredds.Mapper attribute)</td>
<td>75</td>
</tr>
<tr>
<td>srcDSProjection (nansat.mappers.opendap_occci.Mapper attribute)</td>
<td>75</td>
</tr>
</tbody>
</table>
srcDSProjection (nansat.mappers.mapper_opendap_osisaf.Mapper attribute), 76
srcDSProjection (nansat.mappers.mapper_opendap_siwtacsst.Mapper attribute), 76
srcDSProjection (nansat.mappers.mapper_opendap_sstcci.Mapper attribute), 77
structFmt (nansat.mappers.envisat.Envisat attribute), 67
subsetArraySize (nansat.Figure attribute), 25
subsetArraySize (nansat.figure.Figure attribute), 25
t0 (nansat.mappers.mapper_opendap_osisaf.Mapper attribute), 76
t0 (nansat.mappers.mapper_opendap_siwtacsst.Mapper attribute), 76
tagList() (nansat.node.Node method), 36
test_mapper() (nansat.mappers.opendap.Opendap method), 80
test_openable() (in module nansat.tools), 38
text_ax (nansat.pointbrowser.PointBrowser attribute), 37
time_coverage() (nansat.mappers.mapper_cmems.Mapper method), 70
time_coverage_end (nansat.Nansat attribute), 58
time_coverage_end (nansat.nansat.Nansat attribute), 32
time_coverage_start (nansat.Nansat attribute), 58
time_coverage_start (nansat.nansat.Nansat attribute), 32
timeCalendarStart (nansat.mappers.mapper_opendap_globcurrent.Mapper attribute), 74
timeCalendarStart (nansat.mappers.mapper_opendap_globcurrent_thredds.Mapper attribute), 75
timeCalendarStart (nansat.mappers.mapper_opendap_occci.Mapper attribute), 75
timeCalendarStart (nansat.mappers.mapper_opendap_sstcci.Mapper attribute), 77
times() (nansat.mappers.mapper_netcdf Cf.Mapper method), 73
timeVarName (nansat.mappers.mapper_opendap_globcurrent.Mapper attribute), 74
timeVarName (nansat.mappers.mapper_opendap_globcurrent_thredds.Mapper attribute), 75
timeVarName (nansat.mappers.mapper_opendap_occci.Mapper attribute), 75
timeVarName (nansat.mappers.mapper_opendap_osisaf.Mapper attribute), 76
timeVarName (nansat.mappers.mapper_opendap_siwtacsst.Mapper attribute), 76
TITLE_LOCATION_X (nansat.Figure attribute), 61
TITLE_LOCATION_X (nansat.figure.Figure attribute), 21
TITLE_LOCATION_Y (nansat.Figure attribute), 61
TITLE_LOCATION_Y (nansat.figure.Figure attribute), 21
transform_coordinates() (nansat.vrt.VRT static method), 45
transform_points() (nansat.Domain method), 50
transform_points() (nansat.domain.Domain method), 17
transform_points() (nansat.vrt.VRT method), 46
transparency (nansat.figure.Figure attribute), 25
transparency (nansat.Figure attribute), 65
undo() (nansat.Nansat method), 58
undo() (nansat.nansat.Nansat method), 32
UNWANTED_METADATA (nansat.exporter.Exporter attribute), 18
varname2wkv (nansat.mappers.globcolour.Globcolour attribute), 67
VRT (class in nansat.vrt), 39
vrt (nansat.Domain attribute), 51
vrt (nansat.domain.Domain attribute), 17
time_coverage() (nansat.mappers.mapper_cmems.Mapper method), 70
time_coverage_end (nansat.Nansat attribute), 58
time_coverage_end (nansat.nansat.Nansat attribute), 32
time_coverage_start (nansat.Nansat attribute), 58
time_coverage_start (nansat.nansat.Nansat attribute), 32
timeCalendarStart (nansat.mappers.mapper_opendap_globcurrent.Mapper attribute), 74
timeCalendarStart (nansat.mappers.mapper_opendap_globcurrent_thredds.Mapper attribute), 75
timeCalendarStart (nansat.mappers.mapper_opendap_occci.Mapper attribute), 75
timeCalendarStart (nansat.mappers.mapper_opendap_sstcci.Mapper attribute), 77
times() (nansat.mappers.mapper_netcdf Cf.Mapper method), 73
timeVarName (nansat.mappers.mapper_opendap_globcurrent.Mapper attribute), 74
timeVarName (nansat.mappers.mapper_opendap_globcurrent_thredds.Mapper attribute), 75
timeVarName (nansat.mappers.mapper_opendap_osisaf.Mapper attribute), 76
timeVarName (nansat.mappers.mapper_opendap_siwtacsst.Mapper attribute), 76
x_vrt (nansat.geolocation.Geolocation attribute), 25
xml (nansat.vrt.VRT attribute), 46
xml() (nansat.node.Node method), 36
xName (nansat.mappers.mapper_opendap_globcurrent Mapper attribute), 74
xName (nansat.mappers.mapper_opendap_globcurrent_thredds.Mapper attribute), 75
xName (nansat.mappers.mapper_opendap_occci.Mapper attribute), 75
xName (nansat.mappers.mapper_opendap_osisaf.Mapper attribute), 76
xName (nansat.mappers.mapper_opendap_siwtacsst.Mapper attribute), 76
xName (nansat.mappers.mapper_opendap_sstcci.Mapper attribute), 77

Y

y_vrt (nansat.geolocation.Geolocation attribute), 25
yName (nansat.mappers.mapper_opendap_globcurrent.Mapper attribute), 74
yName (nansat.mappers.mapper_opendap_globcurrent_thredds.Mapper attribute), 75
yName (nansat.mappers.mapper_opendap_occci.Mapper attribute), 75
yName (nansat.mappers.mapper_opendap_osisaf.Mapper attribute), 76
yName (nansat.mappers.mapper_opendap_siwtacsst.Mapper attribute), 76
yName (nansat.mappers.mapper_opendap_sstcci.Mapper attribute), 77