

---

# **S1\_NRB Documentation**

***Release 1.3***

**the S1\_NRB Developers**

**May 24, 2023**

# CONTENTS

<b>1</b>	<b>General Topics</b>	<b>2</b>
1.1	Installation . . . . .	2
1.1.1	SNAP . . . . .	2
1.1.2	S1_NRB . . . . .	2
1.1.3	Docker . . . . .	3
1.2	Usage . . . . .	3
1.2.1	Configuration . . . . .	3
1.2.2	Command Line Interface . . . . .	5
1.3	S1-NRB Production . . . . .	6
1.3.1	MGRS Gridding . . . . .	6
1.3.2	Scene Management . . . . .	6
1.3.3	DEM Handling . . . . .	6
1.3.4	OSV Handling . . . . .	6
1.3.5	SNAP Processing . . . . .	7
1.3.6	NRB Formatting . . . . .	7
1.4	Geolocation Accuracy . . . . .	8
1.4.1	Limitations . . . . .	9
1.4.2	Development Status . . . . .	9
1.4.3	References . . . . .	9
1.5	Folder Structure . . . . .	9
<b>2</b>	<b>API Documentation</b>	<b>10</b>
2.1	Configuration . . . . .	10
2.2	Processing . . . . .	11
2.2.1	SNAP . . . . .	11
2.2.2	NRB . . . . .	17
2.2.3	ETAD . . . . .	21
2.2.4	DEM . . . . .	21
2.3	Tile Extraction . . . . .	24
2.4	Ancillary Functions . . . . .	25
2.5	Scene Search . . . . .	27
2.6	Metadata . . . . .	28
2.6.1	Extraction . . . . .	28
2.6.2	XML . . . . .	31
2.6.3	STAC . . . . .	32
<b>3</b>	<b>Examples</b>	<b>34</b>
3.1	Exploring S1-NRB data cubes . . . . .	34
3.1.1	Introduction . . . . .	34
3.1.2	Getting started . . . . .	34
<b>4</b>	<b>About</b>	<b>36</b>
4.1	Changelog . . . . .	36
4.1.1	1.3.0   2023-05-24 . . . . .	36

4.1.2	1.2.0   2022-12-29 . . . . .	36
4.1.3	1.1.0   2022-09-29 . . . . .	36
4.1.4	1.0.2   2022-08-24 . . . . .	37
4.1.5	1.0.1   2022-07-03 . . . . .	37
4.1.6	1.0.0   2022-06-23 . . . . .	37
4.1.7	0.4.2   2022-06-16 . . . . .	37
4.1.8	0.4.1   2022-06-01 . . . . .	37
4.1.9	0.4.0   2022-05-30 . . . . .	38
4.1.10	0.3.0   2022-03-30 . . . . .	38
4.1.11	0.2.0   2022-03-03 . . . . .	38
4.1.12	0.1.0   2022-01-14 . . . . .	38
<b>5</b>	<b>Indices and tables</b>	<b>39</b>
	<b>Python Module Index</b>	<b>40</b>
	<b>Index</b>	<b>41</b>

S1\_NRB is a prototype processor for the Sentinel-1 Normalised Radar Backscatter product. Further information about this product can be found [here](#).

## GENERAL TOPICS

### 1.1 Installation

#### 1.1.1 SNAP

S1\_NRB requires ESA's Sentinels Application Platform (SNAP) software to produce S1-NRB products. Version 1.0.0 has been developed based on SNAP 8. SNAP 9 is supported since version 1.0.2. Downloaders for different operating systems can be obtained from the [official webpage](#).

The following code can be used to replicate the software installation on a Linux OS:

```
VERSION=9
TARGET=~/.SNAP"$VERSION"

INSTALLER=esa-snap_sentinel_unix_"$VERSION"_0_0.sh
wget https://download.esa.int/step/snap/"$VERSION".0/installers/"$INSTALLER"
bash $INSTALLER -q -dir $TARGET
$TARGET/bin/snap --nosplash --nogui --modules --update-all
```

See also the web page on how to [update SNAP from the command line](#).

Alternatively, updates for individual modules and versions can be downloaded in the [SNAP Update Center](#). The latest bundle that was used during release of version 1.0.0 is available here: [https://step.esa.int/updatecenter/8.0\\_20220323-143356/](https://step.esa.int/updatecenter/8.0_20220323-143356/).

#### 1.1.2 S1\_NRB

The S1\_NRB package is not yet available via conda-forge or other common package distribution channels. For now, the following shall provide a convenient installation option provided that Anaconda or Miniconda has been installed:

1. Create and then activate the conda environment

```
conda env create --file https://raw.githubusercontent.com/SAR-ARD/S1_NRB/main/
environment.yaml
conda activate nrb_env
```

2. Install the S1\_NRB package into the environment

The package version can be changed as necessary. See the [Tags](#) section of the repository for available versions.

```
pip install git+https://github.com/SAR-ARD/S1_NRB.git@v1.0.0
```

### 1.1.3 Docker

Both SNAP and S1\_NRB can also be installed into a docker container using the Dockerfile that is provided with the package.

## 1.2 Usage

### 1.2.1 Configuration

Usage of the S1\_NRB package relies on a configuration file that needs to be set up by the user. The configuration file follows the INI format, which uses plain text to store properties as key-value pairs. INI files can be created and opened with any text editor. An example `config.ini` file for the S1\_NRB package can be found here:

[https://github.com/SAR-ARD/S1\\_NRB/blob/main/config.ini](https://github.com/SAR-ARD/S1_NRB/blob/main/config.ini)

The following provides an overview of the parameters the `config.ini` should contain and anything that should be considered when selecting their values:

- **mode**

Options: `all` | `nrb` | `rtc`

This parameter determines if the entire processing chain should be executed or only part of it.

- **aoi\_files & aoi\_geometry**

The area of interest (AOI) for which S1-NRB products should be created.

`aoi_files` can be used to define the area of interest via MGRS tile IDs, which must be provided comma-separated (e.g., `aoi_files = 32TNS, 32TMT, 32TMS`). `aoi_geometry` defines the area of interest via a full path to a vector file supported by `spatialist.vector.Vector`. This option will automatically search for overlapping MGRS tiles and use these for processing. Both parameters are optional and can be set to `None` or left empty. `aoi_files` overrides `aoi_geometry`. If neither is defined, all tiles overlapping with the scene search result are processed.

- **mindate & maxdate**

The time period to create S1-NRB products for. Allowed are all string representations that can be parsed by `dateutil.parser.parse()`.

- **date\_strict**

Treat dates as strict limits or also allow flexible limits to incorporate scenes whose acquisition period overlaps with the defined limit?

- `strict`: `start >= mindate & stop <= maxdate`
- `not strict`: `stop >= mindate & start <= maxdate`

- **sensor**

Options: `S1A` | `S1B`

The Sentinel-1 sensor/platform.

- **acq\_mode**

Options: `IW` | `EW` | `SM`

The acquisition mode of the source scenes that should be processed.

- **product**

Options: `GRD` | `SLC`

The product of the source scenes that should be processed.

- **work\_dir**

`work_dir` is the main directory in which any subdirectories and files are stored that are generated during processing. Needs to be provided as full path to an existing directory.

- **rtc\_dir, tmp\_dir, nrb\_dir, wbm\_dir & log\_dir**

Processing S1-NRB products creates many intermediate files that are expected to be stored in separate subdirectories. The default values provided in the example configuration file linked above are recommended and will automatically create subdirectories relative to the directory specified with `work_dir`. E.g., `nrb_dir = NRB` will create the subdirectory `<work_dir>/NRB`. Optionally, full paths to existing directories can be provided for all of these parameters.

- search option I: **scene\_dir & db\_file**

Metadata of any Sentinel-1 scene found in `scene_dir` will be stored in an SQLite database file created by `pyrosar.drivers.Archive`. With `db_file` either a full path to an existing database can be provided or it will be created in `work_dir` if only a filename is provided. E.g., `db_file = scenes.db` will automatically create the database file `<work_dir>/scenes.db`. `scene_dir` needs to be provided as full path to an existing directory and will be searched recursively for any Sentinel-1 scenes using the regex pattern `'^S1[AB].*(SAFE|zip)$'`.

- search option II: **stac\_catalog & stac\_collections**

Alternative to searching scenes in a directory and storing their metadata in an SQLite database, scenes can be queried from a STAC catalog. For this, a STAC URL and one or many collections can be defined with `stac_catalog` and `stac_collections` respectively. The scenes are expected to be locally accessible in unpacked folders with the `.SAFE` extension.

- **kml\_file**

The Sentinel-2 Military Grid Reference System (MGRS) tiling system establishes the basis of the processing chain and a local reference file containing the respective tile information for processing S1-NRB products is needed. The official KML file defined for the Sentinel-2 mission provided by ESA can be retrieved [here](#). With the `kml_file` parameter either a full path to this reference file can be provided or it is expected to be located in the directory provided with `work_dir` if only a filename is provided. E.g., the processor expects to find `<work_dir>/s2_grid.kml` if `kml_file = s2_grid.kml`.

- **dem\_type**

Options: Copernicus 10m EEA DEM | Copernicus 30m Global DEM II | Copernicus 30m Global DEM | GETASSE30

The Digital Elevation Model (DEM) that should be used for processing.

Note that water body masks are not available for “GETASSE30”, and will therefore not be included in the product data mask. “Copernicus 10m EEA DEM” and “Copernicus 30m Global DEM II” (both include water body masks) are retrieved from the [Copernicus Space Component Data Access system \(CSCDA\)](#), which requires registration. The processor reads username and password from the environment variables `DEM_USER` and `DEM_PASS` if possible and otherwise interactively asks for authentication if one of these DEM options is selected.

- **gdal\_threads**

Temporarily changes `GDAL_NUM_THREADS` during processing. Will be reset after processing has finished.

- **measurement**

Options: `gamma` | `sigma`

The backscatter measurement convention. Either creates gamma naught RTC ( $\gamma_T^0$ ) or ellipsoidal sigma naught ( $\sigma_E^0$ ).

- **annotation**

A comma-separated list to define the annotation layers to be created. Supported options:

- `dm`: data mask (four masks: not layover not shadow, layover, shadow, ocean water)
- `ei`: ellipsoidal incident angle (needed for computing geolocation accuracy)
- `em`: digital elevation model
- `id`: acquisition ID image (source scene ID per pixel)

- lc: RTC local contributing area
- li: local incident angle
- np: noise power (NESZ, per polarization)
- gs: gamma-sigma ratio: sigma0 RTC / gamma0 RTC (ignored if measurement is not gamma)
- sg: sigma-gamma ratio: gamma0 RTC / sigma0 ellipsoidal (ignored if measurement is not sigma)

Use one of the following to create no annotation layer:

- `annotation =`
- `annotation = None`
- `etad & etad_dir`

Determines if the [Extended Timing Annotation Dataset \(ETAD\) correction](#) should be performed or not. If `etad=True`, `etad_dir` is searched for ETAD products matching the respective input SLC and a new SLC is created in `tmp_dir`, which is then used for all other processing steps. If `etad=False`, `etad_dir` will be ignored.

## Sections

Configuration files in INI format can have different sections. Each section begins at a section name and ends at the next section name. The `config.ini` file used with the S1\_NRB package should at least have a dedicated section for processing related parameters. This section is by default named `[PROCESSING]` (see [example config file](#)).

A reserved section `[METADATA]` may contain user-specific metadata to be written to the product's metadata files. Currently supported fields:

- `access_url`
- `licence`
- `doi`
- `processing_center`

Users might create several processing sections in the same configuration file with parameter values that correspond to different processing scenarios (e.g., for different areas of interest). Note that each section must contain all necessary configuration parameters even if only a few are varied between the sections.

### 1.2.2 Command Line Interface

Once a configuration file has been created and all of its parameters have been properly defined, it can be used to start the processor using the command line interface (CLI) tool provided with the S1\_NRB package.

The following options are currently available.

Print a help message for the CLI tool:

```
s1_nrb --help
```

Print the processor version:

```
s1_nrb --version
```

Start the processor using parameters defined in the default section of a `config.ini` file:

```
s1_nrb -c /path/to/config.ini
```

Start the processor using parameters defined in section `SECTION_NAME` of a `config.ini` file:

```
s1_nrb -c /path/to/config.ini -s SECTION_NAME
```



Start the processor using parameters defined in the default section of a `config.ini` file but override some parameters, e.g. `acq_mode` and `annotation`:

```
s1_nrb -c /path/to/config.ini --acq_mode IW --annotation dm,id
```

## 1.3 S1-NRB Production

The following sections give a brief overview of the major components of creating a S1-NRB product. All steps are comprised in function `S1_NRB.processor.main()`. The pyroSAR package builds the foundation of the processor and its documentation is used to outline the processor details to conveniently link to all relevant functionality.

### 1.3.1 MGRS Gridding

The basis of the processing chain builds the Sentinel-2 Military Grid Reference System (MGRS) tiling system. Hence, a reference file is needed containing the respective tile information for processing S1-NRB products. A KML file is available online that will be used in the following steps:

```
S2A_OPER_GIP_TILPAR_MPC__20151209T095117_V20150622T000000_21000101T000000_B00.kml
```

This file contains all relevant information about individual tiles, in particular the EPSG code of the respective UTM zone and the geometry of the tile in UTM coordinates. The function `S1_NRB.tile_extraction.aoi_from_tile()` can be used to extract one or multiple tiles as `spatialist.vector.Vector` object.

### 1.3.2 Scene Management

The S1 images are managed in a local SQLite database to select scenes for processing (see pyroSAR's section on [Database Handling](#)) or are directly queried from a STAC catalog (see `S1_NRB.archive.STACArchive`).

After loading an MGRS tile as an `spatialist.vector.Vector` object and selecting all relevant overlapping scenes from the database, processing can commence.

### 1.3.3 DEM Handling

S1\_NRB offers a convenience function `S1_NRB.dem.mosaic()` for creating scene-specific DEM files from various sources. The function is based on `pyroSAR.auxdata.dem_autoload()` and `pyroSAR.auxdata.dem_create()` and will

- download all tiles of the selected source overlapping with a defined geometry
- create a GDAL VRT virtual mosaic from the tiles including gap filling over ocean areas
- create a new GeoTIFF from the VRT including geoid-ellipsoid height conversion if necessary (WGS84 heights are generally required for SAR processing but provided heights might be relative to a geoid like EGM2008).

### 1.3.4 OSV Handling

Sentinel-1 orbit state vector files (OSV) for enhancing the orbit location accuracy are downloaded directly by pyroSAR (see `pyroSAR.S1.OSV`), but can also be downloaded automatically by SNAP. For S1-NRB processing at least Restituted Orbit files (RESORB) are needed while the more accurate Precise Orbit Ephemerides (POEORB) delivered two weeks after scene acquisition do not provide much additional benefit.

### 1.3.5 SNAP Processing

The central function for processing backscatter data with SNAP is `S1_NRB.snap.process()`. It will perform all necessary steps to generate radiometrically terrain corrected gamma naught backscatter plus all relevant additional datasets like local incident angle and local contribution area (see argument `export_extra`). In a full processor run, the following functions are called in sequence:

- `S1_NRB.snap.pre()`: general pre-processing including
  - Orbit state vector enhancement
  - (GRD only) border noise removal
  - Calibration to beta naught (for RTC) and sigma naught (for NESZ)
  - Thermal noise removal (including generation of noise equivalent sigma zero (NESZ) noise power images)
  - (SLC only) debursting and swath merging
- `S1_NRB.snap.mli()`: creates multi-looked image files (MLIs) per polarization if the target pixel spacing is larger than the source pixel spacing.
- `S1_NRB.snap.rtc()`: radiometric terrain flattening. Output is backscatter in gamma naught RTC ( $\gamma_T^0$ ) and sigma naught RTC ( $\sigma_T^0$ ) as well as the scattering area ( $\beta^0/\gamma_T^0$ ).
- `S1_NRB.snap.gsr()`: computation of the gamma-sigma ratio ( $\sigma_T^0/\gamma_T^0$ ).
- `S1_NRB.snap.geo()`: geocoding. This function may be called multiple times if the scene overlaps with multiple UTM zones.

The output is a BEAM-DIMAP product which consists of a `dim` metadata file and a `data` folder containing the individual image layers in ENVI format (extension `img`). The function `S1_NRB.snap.find_datasets()` can be used to collect the individual images files for a scene.

Depending on the user configuration parameters `measurement` and `annotation`, some modifications to the workflow above are possible:

- `S1_NRB.snap.pre()` may only calibrate to sigma naught if no RTC is necessary (`measurement = sigma`)
- execution of `S1_NRB.snap.rtc()` may be skipped
- `S1_NRB.snap.gsr()` may be replaced by `S1_NRB.snap.sgr()` to create a sigma-gamma ratio ( $\gamma_T^0/\sigma_E^0$ )

### 1.3.6 NRB Formatting

During RTC processing, files covering a whole scene are created. In this last step, the scene-based structure is converted to the MGRS tile structure. If one tile overlaps with multiple scenes, these scenes are first virtually mosaiced using VRT files. The files are then subsetting to the actual tile extent, converted to Cloud Optimized GeoTIFFs (COG), and renamed to the S1-NRB naming scheme. All steps are performed by `S1_NRB.nrb.format()`. The actual file format conversion is done with `spatialist.auxil.gdalwarp()`, which is a simple wrapper around the `gdalwarp` utility of GDAL. The following is an incomplete code example highlighting the general procedure of converting the individual images. The `outfile` name is generated from information of the source images, the MGRS tile ID and the name of the respective file of the RTC processing step.

```
from spatialist import gdalwarp, Raster
from osgeo import gdal

write_options = ['BLOCKSIZE=512',
                 'COMPRESS=LERC_ZSTD',
                 'MAX_Z_ERROR=0.001']

with Raster(infiles, list_separate=False) as ras:
    source = ras.filename
```

(continues on next page)

(continued from previous page)

```
gdalwarp(src=source, dst=outfile,
         options={'format': 'COG',
                  'outputBounds': [xmin, ymin, xmax, ymax],
                  'creationOptions': write_options})
```

After all COG files have been created, GDAL VRT files are written for log scaling and conversion to other backscatter conventions using function `S1_NRB.nrb.create_vrt()`. The code below demonstrates the generation of a VRT file for log-scaling using `spatialist.auxil.gdalbuildvrt()` followed by an XML modification to insert the pixel function (a way to achieve this with GDAL's `gdalbuildvrt` functionality has not yet been found).

```
from lxml import etree
from spatialist import gdalbuildvrt

src = 'test.tif'
dst = 'test_db.vrt'

gdalbuildvrt(src=src, dst=dst)
tree = etree.parse(dst)
root = tree.getroot()
band = tree.find('VRTRasterBand')
band.attrib['subClass'] = 'VRTDerivedRasterBand'
pixfun = etree.SubElement(band, 'PixelFunctionType')
pixfun.text = 'dB'
arg = etree.SubElement(band, 'PixelFunctionArguments')
arg.attrib['fact'] = '10'
etree.indent(root)
tree.write(dst, pretty_print=True, xml_declaration=False, encoding='utf-8')
```

In a last step the OGC XML and STAC JSON metadata files will be written for the S1-NRB product.

## 1.4 Geolocation Accuracy

Item 4.3 of the CARD4L NRB specification requires, as minimum, an estimate of the absolute location error (ALE) “as bias and standard deviation, provided in slant range/azimuth, or Northing/Easting” [1]. As desired target the accuracy is less or equal 0.1 pixels radial root mean square error (rRMSE), which can be defined as:

$$RMSE_{planar} = \sqrt{RMSE_{SLC,Az}^2 + \left(\frac{RMSE_{SLC,Rg}}{\sin(\theta_{i,min})}\right)^2 + RMSE_{DEM,planar}^2 + RMSE_{proc}^2}$$

The error induced by the DEM can be described as:

$$RMSE_{DEM,planar} = \frac{\sigma_{DEM}}{\tan(\theta_{i,min})}$$

where

$\theta_{i,min}$  = The minimum possible angle of incidence

$RMSE_{SLC,Az/Rg}$  = Error induced by SLC source data in azimuth/range

$RMSE_{DEM,planar}$  = Error induced by DEM inaccuracy

$RMSE_{proc}$  = Error induced by other processing steps

$\sigma_{DEM}$  = DEM accuracy at  $1\sigma$  (LE68)

### 1.4.1 Limitations

Currently, the following simplifications need to be considered for the calculation of rRMSE values found in the metadata of each S1-NRB product:

- Processing induced errors ( $RMSE_{proc}$ ) and the error term related to DEM interpolation are not further considered and assumed to be 0.
- The DEM accuracy ( $\sigma_{DEM}$ ) is estimated on the global mean accuracy LE90 reported for the COP-DEM [2] under the assumption of gaussian distribution:
  - Global: LE90 = 2.57; LE68  $\approx$  1.56
- rRMSE is only calculated if a COP-DEM was used for processing, otherwise the value is set to None

### 1.4.2 Development Status

The development status is tracked and discussed in the following Github issue: [https://github.com/SAR-ARD/S1\\_NRB/issues/33](https://github.com/SAR-ARD/S1_NRB/issues/33)

### 1.4.3 References

- [1] [https://ceos.org/ard/files/PFS/NRB/v5.5/CARD4L-PFS\\_NRB\\_v5.5.pdf](https://ceos.org/ard/files/PFS/NRB/v5.5/CARD4L-PFS_NRB_v5.5.pdf)
- [2] [https://spacedata.copernicus.eu/documents/20126/0/GEO1988-CopernicusDEM-SPE-002\\_ProductHandbook\\_I1.00.pdf/082dd479-f908-bf42-51bf-4c0053129f7c?t=1586526993604](https://spacedata.copernicus.eu/documents/20126/0/GEO1988-CopernicusDEM-SPE-002_ProductHandbook_I1.00.pdf/082dd479-f908-bf42-51bf-4c0053129f7c?t=1586526993604)

## 1.5 Folder Structure

The following demonstrates a possible structure created to store intermediate and final files during a processor run. The listed files describe the output of the user configuration parameter `measurement` set to `gamma` and all output annotation layers enabled (`annotation = dm, ei, em, id, lc, li, np, gs`). The structure is based on the default configuration defined in the `config.ini` file and can be modified by a user. Folders are highlighted in bold.

This section is currently not supported with LaTeX/PDF as it was written with collapsible elements in HTML.

## API DOCUMENTATION

### 2.1 Configuration

<i>gdal_conf</i>	Stores GDAL configuration options for the current process.
<i>snap_conf</i>	Returns a dictionary of additional parameters for <i>S1_NRB.snap.process()</i> based on processing configurations provided by the config file.
<i>get_config</i>	Returns the content of a <i>config.ini</i> file as a dictionary.

`S1_NRB.config.gdal_conf(config)`

Stores GDAL configuration options for the current process.

**Parameters** `config` (*dict*) – Dictionary of the parsed config parameters for the current process.

**Returns** Dictionary containing GDAL configuration options for the current process.

**Return type** *dict*

`S1_NRB.config.get_config(config_file, proc_section='PROCESSING', **kwargs)`

Returns the content of a *config.ini* file as a dictionary.

**Parameters**

- **config\_file** (*str*) – Full path to the config file that should be parsed to a dictionary.
- **proc\_section** (*str*) – Section of the config file that processing parameters should be parsed from. Default is 'PROCESSING'.

**Returns** `out_dict` – Dictionary of the parsed config parameters.

**Return type** *dict*

`S1_NRB.config.get_keys(section)`

get all allowed configuration keys

**Parameters** `section` (*{'processing', 'metadata'}*) – the configuration section to get the allowed keys for.

**Returns** a list of keys

**Return type** *list[str]*

`S1_NRB.config.snap_conf(config)`

Returns a dictionary of additional parameters for *S1\_NRB.snap.process()* based on processing configurations provided by the config file.

**Parameters** `config` (*dict*) – Dictionary of the parsed config parameters for the current process.

**Returns** Dictionary of parameters that can be passed to *S1\_NRB.snap.process()*

**Return type** *dict*

## 2.2 Processing

<i>main</i>	Main function that initiates and controls the processing workflow.
-------------	--

`S1_NRB.processor.main(config_file, section_name='PROCESSING', debug=False, **kwargs)`

Main function that initiates and controls the processing workflow.

### Parameters

- **config\_file** (*str*) – Full path to a *config.ini* file.
- **section\_name** (*str*) – Section name of the *config.ini* file that processing parameters should be parsed from. Default is 'PROCESSING'.
- **debug** (*bool*) – Set pyroSAR logging level to DEBUG? Default is False.
- **\*\*kwargs** – extra arguments to override parameters in the config file. E.g. *acq\_mode*.

### 2.2.1 SNAP

#### core processing

<i>process</i>	Main function for SAR processing with SNAP.
<i>geo</i>	Range-Doppler geocoding.
<i>grd_buffer</i>	GRD extent buffering.
<i>gsr</i>	Gamma-sigma ratio computation for either ellipsoidal or RTC sigma nought.
<i>mli</i>	Multi-looking.
<i>pre</i>	General SAR preprocessing.
<i>rtc</i>	Radiometric Terrain Flattening.

#### ancillary functions

<i>find_datasets</i>	Find processed datasets for a scene in a certain CRS.
<i>get_metadata</i>	Get processing metadata needed for NRB metadata.
<i>postprocess</i>	Performs edge cleaning and sets the nodata value in the output ENVI HDR files.

`S1_NRB.snap.find_datasets(scene, outdir, epsg)`

Find processed datasets for a scene in a certain CRS.

### Parameters

- **scene** (*str*) – the file name of the SAR scene
- **outdir** (*str*) – the output directory in which to search for results
- **epsg** (*int*) – the EPSG code defining the output projection of the processed scenes.

### Returns

Either None if no datasets were found or a dictionary with the following keys and values pointing to the file names (polarization-specific keys depending on product availability):

- hh-g-lin: gamma nought RTC backscatter HH polarization
- hv-g-lin: gamma nought RTC backscatter HV polarization

- `vh-g-lin`: gamma nought RTC backscatter VH polarization
- `vv-g-lin`: gamma nought RTC backscatter VV polarization
- `hh-s-lin`: sigma nought ellipsoidal backscatter HH polarization
- `hv-s-lin`: sigma nought ellipsoidal backscatter HV polarization
- `vh-s-lin`: sigma nought ellipsoidal backscatter VH polarization
- `vv-s-lin`: sigma nought ellipsoidal backscatter VV polarization
- `dm`: layover-shadow data mask
- `ei`: ellipsoidal incident angle
- `gs`: gamma-sigma ratio
- `lc`: local contributing area (aka scattering area)
- `li`: local incident angle
- `sg`: sigma-gamma ratio
- `np-hh`: NESZ HH polarization
- `np-hv`: NESZ HV polarization
- `np-vh`: NESZ VH polarization
- `np-vv`: NESZ VV polarization

**Return type** `dict` or `None`

```
S1_NRB.snap.geo(*src, dst, workflow, spacing, crs, geometry=None, buffer=0.01, export_extra=None,
                 standard_grid_origin_x=0, standard_grid_origin_y=0, dem,
                 dem_resampling_method='BILINEAR_INTERPOLATION',
                 img_resampling_method='BILINEAR_INTERPOLATION', gpt_args=None, **bands)
```

Range-Doppler geocoding.

### Parameters

- **`src`** (`list[str]` or `None`) – variable number of input scene file names
- **`dst`** (`str`) – the file name of the target scene. Format is BEAM-DIMAP.
- **`workflow`** (`str`) – the target XML workflow file name
- **`spacing`** (`int` or `float`) – the target pixel spacing in meters
- **`crs`** (`int` or `str`) – the target coordinate reference system
- **`geometry`** (`dict` or `spatialist.vector.Vector` or `str` or `None`) – a vector geometry to limit the target product's extent
- **`buffer`** (`int` or `float`) – an additional buffer in degrees to add around `geometry`
- **`export_extra`** (`list[str]` or `None`) – a list of ancillary layers to write. Supported options:
  - `DEM`
  - `incidenceAngleFromEllipsoid`
  - `layoverShadowMask`
  - `localIncidenceAngle`
  - `projectedLocalIncidenceAngle`
- **`standard_grid_origin_x`** (`int` or `float`) – the X coordinate for pixel alignment
- **`standard_grid_origin_y`** (`int` or `float`) – the Y coordinate for pixel alignment
- **`dem`** (`str`) – the DEM file

- **dem\_resampling\_method** (*str*) – the DEM resampling method
- **img\_resampling\_method** (*str*) – the SAR image resampling method
- **gpt\_args** (*list[str]* or *None*) – a list of additional arguments to be passed to the gpt call
  - e.g. ['-x', '-c', '2048M'] for increased tile cache size and intermediate clearing
- **bands** – band ids for the input scenes in *src* as lists with keys bands<index>, e.g., bands1=['NESZ\_VV'], bands2=['Gamma0\_VV'], ...

See also:

`pyroSAR.snap.auxil.sub_parametrize`, `pyroSAR.snap.auxil.geo_parametrize`

`S1_NRB.snap.get_metadata(scene, outdir)`

Get processing metadata needed for NRB metadata.

#### Parameters

- **scene** (*str*) – the name of the SAR scene
- **outdir** (*str*) – the directory to search for processing output

Return type `dict`

`S1_NRB.snap.grd_buffer(src, dst, workflow, neighbors, buffer=100, gpt_args=None)`

GRD extent buffering. GRDs, unlike SLCs, do not overlap in azimuth. With this function, a GRD can be buffered using the neighboring acquisitions. First, all images are mosaicked using the *SliceAssembly* operator and then subsetting to the extent of the main scene including a buffer.

#### Parameters

- **src** (*str*) – the file name of the source scene
- **dst** (*str*) – the file name of the target scene. Format is BEAM-DIMAP.
- **workflow** (*str*) – the output SNAP XML workflow filename.
- **neighbors** (*list[str]*) – the file names of neighboring scenes
- **buffer** (*int*) – the buffer size in meters
- **gpt\_args** (*list[str]* or *None*) – a list of additional arguments to be passed to the gpt call
  - e.g. ['-x', '-c', '2048M'] for increased tile cache size and intermediate clearing

`S1_NRB.snap.gsr(src, dst, workflow, src_sigma=None, gpt_args=None)`

Gamma-sigma ratio computation for either ellipsoidal or RTC sigma nought.

#### Parameters

- **src** (*str*) – the file name of the source scene. Both gamma and sigma bands are expected unless *src\_sigma* is defined.
- **dst** (*str*) – the file name of the target scene. Format is BEAM-DIMAP.
- **workflow** (*str*) – the output SNAP XML workflow filename.
- **src\_sigma** (*str* or *None*) – the optional file name of a second source product from which to read the sigma band.
- **gpt\_args** (*list[str]* or *None*) – a list of additional arguments to be passed to the gpt call
  - e.g. ['-x', '-c', '2048M'] for increased tile cache size and intermediate clearing



`S1_NRB.snap.mli(src, dst, workflow, spacing=None, rlks=None, azlks=None, gpt_args=None)`

Multi-looking.

#### Parameters

- **src** (*str*) – the file name of the source scene
- **dst** (*str*) – the file name of the target scene. Format is BEAM-DIMAP.
- **workflow** (*str*) – the output SNAP XML workflow filename.
- **spacing** (*int* or *float*) – the target pixel spacing for automatic determination of looks using function `pyroSAR.ancillary.multilook_factors()`. Overridden by arguments *rlks* and *azlks* if they are not None.
- **rlks** (*int* or *None*) – the number of range looks.
- **azlks** (*int* or *None*) – the number of azimuth looks.
- **gpt\_args** (*list[str]* or *None*) – a list of additional arguments to be passed to the gpt call
  - e.g. `['-x', '-c', '2048M']` for increased tile cache size and intermediate clearing

See also:

`pyroSAR.snap.auxil.mli_parametrize`, `pyroSAR.ancillary.multilook_factors`

`S1_NRB.snap.postprocess(src, clean_edges=True, clean_edges_pixels=4)`

Performs edge cleaning and sets the nodata value in the output ENVI HDR files.

#### Parameters

- **src** (*str*) – the file name of the source scene. Format is BEAM-DIMAP.
- **clean\_edges** (*bool*) – perform edge cleaning?
- **clean\_edges\_pixels** (*int*) – the number of pixels to erode during edge cleaning.

`S1_NRB.snap.pre(src, dst, workflow, allow_res_osv=True, osv_continue_on_fail=False, output_noise=True, output_beta0=True, gpt_args=None)`

General SAR preprocessing. The following operators are used (optional steps in brackets): Apply-Orbit-File(->Remove-GRD-Border-Noise)->Calibration->ThermalNoiseRemoval(->TOPSAR-Deburst)

#### Parameters

- **src** (*str*) – the file name of the source scene
- **dst** (*str*) – the file name of the target scene. Format is BEAM-DIMAP.
- **workflow** (*str*) – the output SNAP XML workflow filename.
- **allow\_res\_osv** (*bool*) – Also allow the less accurate RES orbit files to be used?
- **osv\_continue\_on\_fail** (*bool*) – Continue processing if no OSV file can be downloaded or raise an error?
- **output\_noise** (*bool*) – output the noise power images?
- **output\_beta0** (*bool*) – output beta nought backscatter needed for RTC?
- **gpt\_args** (*list[str]* or *None*) – a list of additional arguments to be passed to the gpt call
  - e.g. `['-x', '-c', '2048M']` for increased tile cache size and intermediate clearing

See also:

`pyroSAR.snap.auxil.orb_parametrize`

```
S1_NRB.snap.process(scene, outdir, measurement, spacing, kml, dem,
                    dem_resampling_method='BILINEAR_INTERPOLATION',
                    img_resampling_method='BILINEAR_INTERPOLATION', rlks=None, azlks=None,
                    tmpdir=None, export_extra=None, allow_res_osv=True, clean_edges=True,
                    clean_edges_pixels=4, neighbors=None, gpt_args=None, cleanup=True)
```

Main function for SAR processing with SNAP.

#### Parameters

- **scene** (*str*) – The SAR scene file name.
- **outdir** (*str*) – The output directory for storing the final results.
- **measurement** (*{'sigma', 'gamma'}*) – the backscatter measurement convention:
  - gamma: RTC gamma nought ( $\gamma^0_T$ )
  - sigma: ellipsoidal signal nought ( $\sigma^0_E$ )
- **spacing** (*int* or *float*) – The output pixel spacing in meters.
- **kml** (*str*) – Path to the Sentinel-2 tiling grid KML file.
- **dem** (*str*) – The DEM filename. Can be created with `S1_NRB.dem.mosaic()`.
- **dem\_resampling\_method** (*str*) – The DEM resampling method.
- **img\_resampling\_method** (*str*) – The image resampling method.
- **rlks** (*int* or *None*) – The number of range looks.
- **azlks** (*int* or *None*) – The number of azimuth looks.
- **tmpdir** (*str* or *None*) – Path to a temporary directory for intermediate products.
- **export\_extra** (*list[str]* or *None*) – A list of ancillary layers to create. Default *None*: do not create any ancillary layers. Options:
  - DEM
  - gammaSigmaRatio:  $\sigma^0_T / \gamma^0_T$
  - sigmaGammaRatio:  $\gamma^0_T / \sigma^0_E$
  - incidenceAngleFromEllipsoid
  - layoverShadowMask
  - localIncidenceAngle
  - NESZ: noise equivalent sigma zero
  - projectedLocalIncidenceAngle
  - scatteringArea
- **allow\_res\_osv** (*bool*) – Also allow the less accurate RES orbit files to be used?
- **clean\_edges** (*bool*) – Erode noisy image edges? See `pyroSAR.snap.auxil.erode_edges()`. Does not apply to layover-shadow mask.
- **clean\_edges\_pixels** (*int*) – The number of pixels to erode.
- **neighbors** (*list[str]* or *None*) – (only applies to GRD) an optional list of neighboring scenes to add a buffer around the main scene using function `grd_buffer()`. If GRDs are processed completely independently, gaps are introduced due to a missing overlap.
- **gpt\_args** (*list[str]* or *None*) – a list of additional arguments to be passed to the gpt call
  - e.g. `['-x', '-c', '2048M']` for increased tile cache size and intermediate clearing

- **cleanup** (*bool*) – Delete intermediate files after successful process termination?

## Examples

```
>>> from S1_NRB import snap
>>> scene = 'S1A_IW_SLC__1SDV_20200103T170700_20200103T170727_030639_0382D5_6A12_
↳zip'
>>> kml = 'S2A_OPER_GIP_TILPAR_MPC__20151209T095117_V20150622T000000_
↳210000101T000000_B00.kml'
>>> dem = 'S1A_IW_SLC__1SDV_20200103T170700_20200103T170727_030639_0382D5_6A12_
↳DEM_EEA10.tif'
>>> outdir = '.'
>>> spacing = 10
>>> rlks = 5
>>> azlks = 1
>>> export_extra = ['localIncidenceAngle', 'incidenceAngleFromEllipsoid',
↳'scatteringArea', 'layoverShadowMask', 'gammaSigmaRatio']
>>> snap.process(scene=scene, outdir=outdir, spacing=spacing, kml=kml, dem=dem,
↳rlks=rlks, azlks=azlks, export_extra=export_extra)
```

`S1_NRB.snap.rtc(src, dst, workflow, dem, dem_resampling_method='BILINEAR_INTERPOLATION',  
sigma0=True, scattering_area=True, dem_oversampling_multiple=2, gpt_args=None)`

Radiometric Terrain Flattening.

### Parameters

- **src** (*str*) – the file name of the source scene
- **dst** (*str*) – the file name of the target scene. Format is BEAM-DIMAP.
- **workflow** (*str*) – the output SNAP XML workflow filename.
- **dem** (*str*) – the input DEM file name.
- **dem\_resampling\_method** (*str*) – the DEM resampling method.
- **sigma0** (*bool*) – output sigma0 RTC backscatter?
- **scattering\_area** (*bool*) – output scattering area image?
- **dem\_oversampling\_multiple** (*int*) – a factor to multiply the DEM oversampling factor computed by SNAP. The SNAP default of 1 has been found to be insufficient with stripe artifacts remaining in the image.
- **gpt\_args** (*list[str]* or *None*) – a list of additional arguments to be passed to the gpt call
  - e.g. ['-x', '-c', '2048M'] for increased tile cache size and intermediate clearing

`S1_NRB.snap.sgr(src, dst, workflow, src_gamma=None, gpt_args=None)`

Sigma-gamma ratio computation.

### Parameters

- **src** (*str*) – the file name of the source scene. Both sigma and gamma bands are expected unless *src\_gamma* is defined.
- **dst** (*str*) – the file name of the target scene. Format is BEAM-DIMAP.
- **workflow** (*str*) – the output SNAP XML workflow filename.
- **src\_gamma** (*str* or *None*) – the optional file name of a second source product from which to read the gamma band.
- **gpt\_args** (*list[str]* or *None*) – a list of additional arguments to be passed to the gpt call

- e.g. `['-x', '-c', '2048M']` for increased tile cache size and intermediate clearing

## 2.2.2 NRB

<code>calc_product_start_stop</code>	Calculates the start and stop times of the NRB product.
<code>create_acq_id_image</code>	Creation of the Acquisition ID image.
<code>create_data_mask</code>	Creation of the Data Mask image.
<code>create_rgb_vrt</code>	Creation of the color composite VRT file.
<code>create_vrt</code>	Creates a VRT file for the specified source dataset(s) and adds a pixel function that should be applied on the fly when opening the VRT file.
<code>format</code>	Finalizes the generation of Sentinel-1 NRB products after RTC processing has finished.
<code>get_datasets</code>	Collect processing output for a list of scenes.

`S1_NRB.nrb.calc_product_start_stop(src_ids, extent, epsg)`

Calculates the start and stop times of the NRB product. The geolocation grid points including their azimuth time information are extracted first from the metadata of each source SLC. These grid points are then used to interpolate the azimuth time for the lower right and upper left (ascending) or upper right and lower left (descending) corners of the MGRS tile extent.

### Parameters

- **src\_ids** (`list[pyroSAR.drivers.ID]`) – List of `ID` objects of all source SLC scenes that overlap with the current MGRS tile.
- **extent** (`dict`) – Spatial extent of the MGRS tile, derived from a `Vector` object.
- **epsg** (`int`) – The coordinate reference system as an EPSG code.

### Returns

- `tuple[str]`
- – Start time of the NRB product formatted as `%Y%m%dT%H%M%S` in UTC.
- – Stop time of the NRB product formatted as `%Y%m%dT%H%M%S` in UTC.

`S1_NRB.nrb.create_acq_id_image(outname, ref_tif, datasets, src_ids, extent, epsg, driver, creation_opt, overviews, dst_nodata)`

Creation of the Acquisition ID image.

### Parameters

- **outname** (`str`) – Full path to the output data mask file.
- **ref\_tif** (`str`) – Full path to any GeoTIFF file of the NRB product.
- **datasets** (`list[dict]`) – List of processed output files that match the source SLC scenes and overlap with the current MGRS tile.
- **src\_ids** (`list[pyroSAR.drivers.ID]`) – List of `ID` objects of all source SLC scenes that overlap with the current MGRS tile.
- **extent** (`dict`) – Spatial extent of the MGRS tile, derived from a `Vector` object.
- **epsg** (`int`) – The CRS used for the NRB product; provided as an EPSG code.
- **driver** (`str`) – GDAL driver to use for raster file creation.
- **creation\_opt** (`list[str]`) – GDAL creation options to use for raster file creation. Should match specified GDAL driver.
- **overviews** (`list[int]`) – Internal overview levels to be created for each raster file.

- **dst\_nodata** (*int* or *str*) – Nodata value to write to the output raster.

`S1_NRB.nrb.create_data_mask(outname, datasets, extent, epsg, driver, creation_opt, overviews, overview_resampling, dst_nodata, wbm=None)`

Creation of the Data Mask image.

#### Parameters

- **outname** (*str*) – Full path to the output data mask file.
- **datasets** (*list[dict]*) – List of processed output files that match the source scenes and overlap with the current MGRS tile. An error will be thrown if not all datasets contain a key *datamask*. The function will return without an error if not all datasets contain a key *dm*.
- **extent** (*dict*) – Spatial extent of the MGRS tile, derived from a *Vector* object.
- **epsg** (*int*) – The coordinate reference system as an EPSG code.
- **driver** (*str*) – GDAL driver to use for raster file creation.
- **creation\_opt** (*list[str]*) – GDAL creation options to use for raster file creation. Should match specified GDAL driver.
- **overviews** (*list[int]*) – Internal overview levels to be created for each raster file.
- **overview\_resampling** (*str*) – Resampling method for overview levels.
- **dst\_nodata** (*int* or *str*) – Nodata value to write to the output raster.
- **wbm** (*str* or *None*) – Path to a water body mask file with the dimensions of an MGRS tile.

`S1_NRB.nrb.create_rgb_vrt(outname, infiles, overviews, overview_resampling)`

Creation of the color composite VRT file.

#### Parameters

- **outname** (*str*) – Full path to the output VRT file.
- **infiles** (*list[str]*) – A list of paths pointing to the linear scaled measurement backscatter files.
- **overviews** (*list[int]*) – Internal overview levels to be defined for the created VRT file.
- **overview\_resampling** (*str*) – Resampling method applied to overview pyramids.

`S1_NRB.nrb.create_vrt(src, dst, fun, relpaths=False, scale=None, offset=None, args=None, options=None, overviews=None, overview_resampling=None)`

Creates a VRT file for the specified source dataset(s) and adds a pixel function that should be applied on the fly when opening the VRT file.

#### Parameters

- **src** (*str* or *list[str]*) – The input dataset(s).
- **dst** (*str*) – The output dataset.
- **fun** (*str*) – A *PixelFunctionType* that should be applied on the fly when opening the VRT file. The function is applied to a band that derives its pixel information from the source bands. A list of possible options can be found here: <https://gdal.org/drivers/raster/vrt.html#default-pixel-functions>. Furthermore, the option ‘decibel’ can be specified, which will implement a custom pixel function that uses Python code for decibel conversion ( $10 \cdot \log_{10}$ ).
- **relpaths** (*bool*) – Should all *SourceFilename* XML elements with attribute *@relativeToVRT="0"* be updated to be paths relative to the output VRT file? Default is False.

- **scale** (*int* or *None*) – The scale that should be applied when computing “real” pixel values from scaled pixel values on a raster band. Will be ignored if *fun*='decibel'.
- **offset** (*float* or *None*) – The offset that should be applied when computing “real” pixel values from scaled pixel values on a raster band. Will be ignored if *fun*='decibel'.
- **args** (*dict* or *None*) – arguments for *fun* passed as *PixelFunctionArguments*. Requires GDAL>=3.5 to be read.
- **options** (*dict* or *None*) – Additional parameters passed to *gdal.BuildVRT*.
- **overviews** (*list[int]* or *None*) – Internal overview levels to be created for each raster file.
- **overview\_resampling** (*str* or *None*) – Resampling method for overview levels.

## Examples

linear gamma0 backscatter as input:

```
>>> src = 's1a-iw-nrb-20220601t052704-043465-0530a1-32tpt-vh-g-lin.tif'
```

decibel scaling I: use *log10* pixel function and additional *Scale* parameter. Known to display well in QGIS, but *Scale* is ignored when reading array in Python.

```
>>> dst = src.replace('-lin.tif', '-log1.vrt')
>>> create_vrt(src=src, dst=dst, fun='log10', scale=10)
```

decibel scaling II: use custom Python pixel function. Requires additional environment variable GDAL\_VRT\_ENABLE\_PYTHON set to YES.

```
>>> dst = src.replace('-lin.tif', '-log2.vrt')
>>> create_vrt(src=src, dst=dst, fun='decibel')
```

decibel scaling III: use *dB* pixel function with additional *PixelFunctionArguments*. Works best but requires GDAL>=3.5.

```
>>> dst = src.replace('-lin.tif', '-log3.vrt')
>>> create_vrt(src=src, dst=dst, fun='dB', args={'fact': 10})
```

**S1\_NRB.nrb.format**(*config, scenes, datadir, outdir, tile, extent, epsg, wbm=None, dem\_type=None, multithread=True, compress=None, overviews=None, kml=None, annotation=None, update=False*)

Finalizes the generation of Sentinel-1 NRB products after RTC processing has finished. This includes the following: - Creating all measurement and annotation datasets in Cloud Optimized GeoTIFF (COG) format - Creating additional annotation datasets in Virtual Raster Tile (VRT) format - Applying the NRB product directory structure & naming convention - Generating metadata in XML and JSON formats for the NRB product as well as source SLC datasets

### Parameters

- **config** (*dict*) – Dictionary of the parsed config parameters for the current process.
- **scenes** (*list[str]*) – List of scenes to process. Either a single scene or multiple, matching scenes (consecutive acquisitions). All scenes are expected to overlap with *extent* and an error will be thrown if the processing output cannot be found for any of the scenes.
- **datadir** (*str*) – The directory containing the datasets processed from the source scenes using pyroSAR.
- **outdir** (*str*) – The directory to write the final files to.

- **tile** (*str*) – ID of an MGRS tile.
- **extent** (*dict*) – Spatial extent of the MGRS tile, derived from a [Vector](#) object.
- **epsg** (*int*) – The CRS used for the NRB product; provided as an EPSG code.
- **wbm** (*str* or *None*) – Path to a water body mask file with the dimensions of an MGRS tile.
- **dem\_type** (*str* or *None*) – if defined, a DEM layer will be added to the product. The suffix *em* (elevation model) is used. Default *None*: do not add a DEM layer.
- **multithread** (*bool*) – Should *gdalwarp* use multithreading? Default is True. The number of threads used, can be adjusted in the *config.ini* file with the parameter *gdal\_threads*.
- **compress** (*str* or *None*) – Compression algorithm to use. See <https://gdal.org/drivers/raster/gtiff.html#creation-options> for options. Defaults to 'LERC\_DEFLATE'.
- **overviews** (*list[int]* or *None*) – Internal overview levels to be created for each GeoTIFF file. Defaults to [2, 4, 9, 18, 36]
- **kml** (*str* or *None*) – The KML file containing the MGRS tile geometries. Only needs to be defined if *dem\_type!=None*.
- **annotation** (*list[str]* or *None*) – an optional list to select the annotation layers. Default *None*: create all layers if the source products contain the required input layers. Options:
  - dm: data mask (four masks: not layover not shadow, layover, shadow, ocean water)
  - ei: ellipsoidal incident angle
  - em: digital elevation model
  - id: acquisition ID image (source scene ID per pixel)
  - lc: RTC local contributing area
  - li: local incident angle
  - np: noise power (NESZ, per polarization)
  - gs: gamma-sigma ratio: sigma0 RTC / gamma0 RTC
  - sg: sigma-gamma ratio: gamma0 RTC / sigma0 ellipsoidal
- **update** (*bool*) – modify existing products so that only missing files are re-created?

**Returns** Either the time spent executing the function in seconds or 'Already processed - Skip!'

**Return type** *str*

`S1_NRB.nrb.get_datasets(scenes, datadir, extent, epsg)`

Collect processing output for a list of scenes. Reads metadata from all source SLC/GRD scenes, finds matching output files in *datadir* and filters both lists depending on the actual overlap of each SLC/GRD valid data coverage with the current MGRS tile geometry. If no output is found for any scene the function will raise an error. To obtain the extent of valid data coverage, first a binary mask raster file is created with the name *data-mask.tif*, which is stored in the same folder as the processing output as found by [find\\_datasets\(\)](#). Then, the boundary of this binary mask is computed and stored as *datamask.gpkg* (see function [spatialist.vector.boundary\(\)](#)). If the provided *extent* does not overlap with this boundary, the output is discarded. This scenario might occur when the scene's geometry read from its metadata overlaps with the tile but the actual extent of data does not.

#### Parameters

- **scenes** (*list[str]*) – List of scenes to process. Either an individual scene or multiple, matching scenes (consecutive acquisitions).



- **datadir** (*str*) – The directory containing the datasets processed from the source scenes using pyroSAR. The function will raise an error if the processing output cannot be found for all scenes in *datadir*.
- **extent** (*dict*) – Spatial extent of the MGRS tile, derived from a [Vector](#) object.
- **epsg** (*int*) – The coordinate reference system as an EPSG code.

#### Returns

- **ids** (list[pyroSAR.drivers.ID]) – List of [ID](#) objects of all source SLC/GRD scenes that overlap with the current MGRS tile.
- **datasets** (list[dict]) – List of RTC processing output files that match each [ID](#) object of *ids*. The format is a list of dictionaries per scene with keys as described by e.g. [S1\\_NRB.snap.find\\_datasets\(\)](#).

See also:

[S1\\_NRB.snap.find\\_datasets\(\)](#)

## 2.2.3 ETAD

<a href="#"><i>process</i></a>	Apply ETAD correction to a Sentinel-1 SLC product.
--------------------------------	--

`S1_NRB.etad.process(scene, etad_dir, out_dir, log)`

Apply ETAD correction to a Sentinel-1 SLC product.

#### Parameters

- **scene** ([pyroSAR.drivers.ID](#)) – The Sentinel-1 SLC scene.
- **etad\_dir** (*str*) – The directory containing ETAD products. This will be searched for products matching the defined SLC.
- **out\_dir** (*str*) – The directory to store results. The ETAD product is unpacked to this directory if necessary. Two new sub-directories SLC\_original SLC\_ETAD and are created, which contain the original unpacked scene and the corrected one respectively.
- **log** ([logging.Logger](#)) – A logger object to write log info.

**Returns** The corrected scene as a pyroSAR ID object.

**Return type** [pyroSAR.drivers.ID](#)

## 2.2.4 DEM

<a href="#"><i>mosaic</i></a>	Create a new scene-specific DEM mosaic GeoTIFF file.
<a href="#"><i>prepare</i></a>	Downloads DEM and WBM tiles and restructures them into the MGRS tiling scheme including re-projection and vertical datum conversion.

`S1_NRB.dem.authenticate(dem_type, username=None, password=None)`

Query the username and password. If None, environment variables DEM\_USER and DEM\_PASS are read. If they are also None, the user is queried interactively.

#### Parameters

- **dem\_type** (*str*) – the DEM type. Needed for determining whether authentication is needed.



- **username** (*str* or *None*) – The username for accessing the DEM tiles. If *None* and authentication is required for the selected DEM type, the environment variable ‘DEM\_USER’ is read. If this is not set, the user is prompted interactively to provide credentials.
- **password** (*str* or *None*) – The password for accessing the DEM tiles. If *None*: same behavior as for username but with env. variable ‘DEM\_PASS’.

**Returns** the username and password

**Return type** tuple[*str* or *None*]

`S1_NRB.dem.mosaic(geometry, dem_type, outname, epsg=None, kml_file=None, dem_dir=None, username=None, password=None, threads=4)`

Create a new scene-specific DEM mosaic GeoTIFF file. Can be created from MGRS-tiled DEMs as created by `S1_NRB.dem.prepare()` or ad hoc using `pyroSAR.auxdata.dem_autoload()` and `pyroSAR.auxdata.dem_create()`. In the former case the arguments *username*, *password* and *threads* are ignored and all tiles found in *dem\_dir* are read. In the latter case the arguments *epsg*, *kml\_file* and *dem\_dir* are ignored and the DEM is only mosaiced and geoid-corrected.

#### Parameters

- **geometry** (*spatialist.vector.Vector*) – The geometry to be covered by the mosaic.
- **dem\_type** (*str*) – The DEM type.
- **outname** (*str*) – The name of the mosaic.
- **epsg** (*int* or *None*) – The coordinate reference system as an EPSG code.
- **kml\_file** (*str* or *None*) – The KML file containing the MGRS tile geometries.
- **dem\_dir** (*str* or *None*) – The directory containing the DEM MGRS tiles.
- **username** (*str* or *None*) – The username for accessing the DEM tiles. If *None* and authentication is required for the selected DEM type, the environment variable ‘DEM\_USER’ is read. If this is not set, the user is prompted interactively to provide credentials.
- **password** (*str* or *None*) – The password for accessing the DEM tiles. If *None*: same behavior as for username but with env. variable ‘DEM\_PASS’.
- **threads** (*int*) – The number of threads to pass to `pyroSAR.auxdata.dem_create()`.

`S1_NRB.dem.prepare(vector, dem_type, dem_dir, wbm_dir, kml_file, dem_strict=True, tilenames=None, threads=None, username=None, password=None)`

Downloads DEM and WBM tiles and restructures them into the MGRS tiling scheme including re-projection and vertical datum conversion.

#### Parameters

- **vector** (*spatialist.vector.Vector*) – The vector object for which to prepare the DEM and WBM tiles. CRS must be EPSG:4236.
- **dem\_type** (*str*) – The DEM type.
- **dem\_dir** (*str* or *None*) – The DEM target directory. DEM preparation can be skipped if set to *None*.
- **wbm\_dir** (*str* or *None*) – The WBM target directory. WBM preparation can be skipped if set to *None*.
- **kml\_file** (*str*) – The KML file containing the MGRS tile geometries.
- **dem\_strict** (*bool*) – strictly only create DEM tiles in the native CRS of the MGRS tile or also allow reprojection to ensure full coverage of the vector object in every CRS.

- **tilenames** (*list[str]* or *None*) – an optional list of MGRS tile names. Default *None*: process all overlapping tiles.
- **threads** (*int* or *None*) – The number of threads to pass to `pyroSAR.auxdata.dem_create()`. Default *None*: use the value of `GDAL_NUM_THREADS` without modification.
- **username** (*str* or *None*) – The username for accessing the DEM tiles. If *None* and authentication is required for the selected DEM type, the environment variable ‘DEM\_USER’ is read. If this is not set, the user is prompted interactively to provide credentials.
- **password** (*str* or *None*) – The password for accessing the DEM tiles. If *None*: same behavior as for username but with env. variable ‘DEM\_PASS’.

## Examples

```
>>> from S1_NRB import dem
>>> from spatialist import bbox
>>> ext = {'xmin': 12, 'xmax': 13, 'ymin': 50, 'ymax': 51}
>>> kml = 'S2A_OPER_GIP_TILPAR_MPC__20151209T095117_V20150622T000000_
→21000101T000000_B00.kml'
# strictly only create overlapping DEM tiles in their native CRS.
# Will create tiles 32UQA, 32UQB, 33UUR and 33UUS.
>>> with bbox(coordinates=ext, crs=4326) as vec:
>>>     dem.prepare(vector=vec, dem_type='Copernicus 30m Global DEM',
>>>                 dem_dir='DEM', wbm_dir=None, dem_strict=True,
>>>                 kml_file=kml, threads=4)
# Process all overlapping DEM tiles to each CRS.
# Will additionally create tiles 32UQA_32633, 32UQB_32633, 33UUR_32632 and 33UUS_
→32632.
>>> with bbox(coordinates=ext, crs=4326) as vec:
>>>     dem.prepare(vector=vec, dem_type='Copernicus 30m Global DEM',
>>>                 dem_dir='DEM', wbm_dir=None, dem_strict=False,
>>>                 kml_file=kml, threads=4)
```

See also:

[`S1\_NRB.tile\_extraction.tile\_from\_aoi`](#)

`S1_NRB.dem.to_mgrs(tile, dst, kml, dem_type, overviews, tr, format='COG', create_options=None, threads=None, pbar=False)`

Create an MGRS-tiled DEM file.

### Parameters

- **tile** (*str*) – the MGRS tile ID
- **dst** (*str*) – the destination file name
- **kml** (*str*) – The KML file containing the MGRS tile geometries.
- **dem\_type** (*str*) – The DEM type.
- **overviews** (*list[int]*) – The overview levels
- **tr** (*tuple[int or float]*) – the target resolution as (x, y)
- **format** (*str*) – the output file format
- **create\_options** (*list[str]* or *None*) – additional creation options to be passed to `spatialist.auxil.gdalwarp()`.

- **threads** (*int* or *None*) – The number of threads to pass to `pyroSAR.auxdata.dem_create()`. Default *None*: use the value of `GDAL_NUM_THREADS` without modification.
- **pbar** (*bool*) –

## 2.3 Tile Extraction

<code>aoi_from_tile</code>	Extract one or multiple MGRS tiles from the global Sentinel-2 tiling grid and return it as a <code>Vector</code> object.
<code>description2dict</code>	Convert the HTML description field of the MGRS tile KML file to a dictionary.
<code>tile_from_aoi</code>	Return a list of MGRS tile IDs or vector objects overlapping one or multiple areas of interest.

`S1_NRB.tile_extraction.aoi_from_tile(kml, tile)`

Extract one or multiple MGRS tiles from the global Sentinel-2 tiling grid and return it as a `Vector` object.

### Parameters

- **kml** (*str*) – Path to the Sentinel-2 tiling grid KML file.
- **tile** (*str* or *list[str]*) – The MGRS tile ID(s) that should be extracted and returned as a vector object. Can also be expressed as `<tile ID>_<EPSG code>` (e.g. `33TUN_32632`). In this case the geometry of the tile is reprojected to the target EPSG code, its corner coordinates rounded to multiples of 10, and a new `Vector` object created.

**Returns** either a single object or a list depending on *tile*

**Return type** `spatialist.vector.Vector` or `list[spatialist.vector.Vector]`

### Notes

The global Sentinel-2 tiling grid can be retrieved from: [https://sentinel.esa.int/documents/247904/1955685/S2A\\_OPER\\_GIP\\_TILPAR\\_MPC\\_\\_20151209T095117\\_V20150622T000000\\_21000101T000000\\_B00.kml](https://sentinel.esa.int/documents/247904/1955685/S2A_OPER_GIP_TILPAR_MPC__20151209T095117_V20150622T000000_21000101T000000_B00.kml)

`S1_NRB.tile_extraction.description2dict(description)`

Convert the HTML description field of the MGRS tile KML file to a dictionary.

**Parameters** **description** (*str*) – The plain text of the *Description* field

**Returns** **attrib** – A dictionary with keys ‘TILE\_ID’, ‘EPSG’, ‘MGRS\_REF’, ‘UTM\_WKT’ and ‘LL\_WKT’. The value of field ‘EPSG’ is of type integer, all others are strings.

**Return type** `dict`

`S1_NRB.tile_extraction.tile_from_aoi(vector, kml, epsg=None, strict=True, return_geometries=False, tilenames=None)`

Return a list of MGRS tile IDs or vector objects overlapping one or multiple areas of interest.

### Parameters

- **vector** (*spatialist.vector.Vector* or *list[spatialist.vector.Vector]*) – The vector object(s) to read. CRS must be EPSG:4236.
- **kml** (*str*) – Path to the Sentinel-2 tiling grid KML file.
- **epsg** (*int* or *list[int]* or *None*) – Define which EPSG code(s) are allowed for the tile selection. If *None*, all tile IDs are returned regardless of projection.

- **strict** (*bool*) – Strictly only return the names/geometries of the overlapping tiles in the target projection or also allow reprojection of neighbouring tiles? In the latter case a tile name takes the form <tile ID>\_<EPSG code>, e.g. *33TUN\_32632*. Only applies if argument *epsg* is of type *int* or a list with one element.
- **return\_geometries** (*bool*) – return a list of `spatialist.vector.Vector` geometry objects (or just the tile names)?
- **tilenames** (*list[str]* or *None*) – an optional list of MGRS tile names to limit the selection

**Returns** *tiles* – A list of unique MGRS tile IDs or `spatialist.vector.Vector` objects with an attribute *mgrs* containing the tile ID.

**Return type** `list[str or spatialist.vector.Vector]`

### Notes

The global Sentinel-2 tiling grid can be retrieved from: [https://sentinel.esa.int/documents/247904/1955685/S2A\\_OPER\\_GIP\\_TILPAR\\_MPC\\_\\_20151209T095117\\_V20150622T000000\\_21000101T000000\\_B00.kml](https://sentinel.esa.int/documents/247904/1955685/S2A_OPER_GIP_TILPAR_MPC__20151209T095117_V20150622T000000_21000101T000000_B00.kml)

## 2.4 Ancillary Functions

<code>check_acquisition_completeness</code>	Check presence of neighboring acquisitions.
<code>check_scene_consistency</code>	Check the consistency of a scene selection.
<code>check_spacing</code>	Check whether the spacing fits into the MGRS tile boundaries.
<code>generate_unique_id</code>	Returns a unique product identifier as a hexadecimal string.
<code>get_max_ext</code>	Gets the maximum extent from a list of geometries.
<code>group_by_time</code>	Group scenes by their acquisition time difference.
<code>log</code>	Format and handle log messages during processing.
<code>set_logging</code>	Set logging for the current process.
<code>vrt_add_overviews</code>	Add overviews to an existing VRT file.

`S1_NRB.ancillary.check_acquisition_completeness(scenes, archive)`

Check presence of neighboring acquisitions. Check that for each scene a predecessor and successor can be queried from the database unless the scene is at the start or end of the data take. This ensures that no scene that could be covering an area of interest is missed during processing.

#### Parameters

- **scenes** (*list[pyroSAR.drivers.ID]*) – a list of scenes
- **archive** (*pyroSAR.drivers.Archive*) – an open scene archive connection

**Raises** `RuntimeError` –

`S1_NRB.ancillary.check_scene_consistency(scenes)`

Check the consistency of a scene selection. The following pyroSAR object attributes must be the same:

- sensor
- acquisition\_mode
- product
- frameNumber (data take ID)

**Parameters** *scenes* (*list[str or pyroSAR.drivers.ID]*) –

Raises `RuntimeError` –

`S1_NRB.ancillary.check_spacing(spacing)`

Check whether the spacing fits into the MGRS tile boundaries.

**Parameters** `spacing` (`int` or `float`) – the target pixel spacing in meters

`S1_NRB.ancillary.generate_unique_id(encoded_str)`

Returns a unique product identifier as a hexadecimal string. The CRC-16 algorithm used to compute the unique identifier is CRC-CCITT (0xFFFF).

**Parameters** `encoded_str` (`bytes`) – A string that should be used to generate a unique id from. The string needs to be encoded; e.g.: `'abc'.encode()`

**Returns** `p_id` – The unique product identifier.

**Return type** `str`

`S1_NRB.ancillary.get_max_ext(geometries, buffer=None)`

Gets the maximum extent from a list of geometries.

**Parameters**

- **geometries** (`list` [`spatialist.vector.Vector`]) – List of `Vector` geometries.
- **buffer** (`float` or `None`) – The buffer in units of the geometries' CRS to add to the extent.

**Returns** `max_ext` – The maximum extent of the selected `Vector` geometries including the chosen buffer.

**Return type** `dict`

`S1_NRB.ancillary.group_by_time(scenes, time=3)`

Group scenes by their acquisition time difference.

**Parameters**

- **scenes** (`list` [`pyroSAR.drivers.ID` or `str`]) – a list of image names
- **time** (`int` or `float`) – a time difference in seconds by which to group the scenes. The default of 3 seconds incorporates the overlap between SLCs.

**Returns** a list of sub-lists containing the file names of the grouped scenes

**Return type** `list` [`list` [`pyroSAR.drivers.ID`]]

`S1_NRB.ancillary.log(handler, mode, proc_step, scenes, msg)`

Format and handle log messages during processing.

**Parameters**

- **handler** (`logging.Logger`) – The log handler for the current process.
- **mode** (`{'info', 'warning', 'exception'}`) – Calls the respective logging helper function. E.g., `handler.info()`.
- **proc\_step** (`str`) – The processing step for which the message is logged.
- **scenes** (`str` or `list` [`str`]) – Scenes that are currently being processed.
- **msg** (`Any`) – The message that should be logged.

`S1_NRB.ancillary.set_logging(config, debug=False)`

Set logging for the current process.

**Parameters**

- **config** (`dict`) – Dictionary of the parsed config parameters for the current process.
- **debug** (`bool`) – Set pyroSAR logging level to `DEBUG`?

**Returns** `log_local` – The log handler for the current process.

**Return type** `logging.Logger`

`S1_NRB.ancillary.vrt_add_overviews(vrt, overviews, resampling='AVERAGE')`

Add overviews to an existing VRT file. Existing overviews will be overwritten.

#### Parameters

- **vrt** (`str`) – the VRT file
- **overviews** (`list[int]`) – the overview levels
- **resampling** (`str`) – the overview resampling method

## 2.5 Scene Search

`class S1_NRB.archive.STACArchive(url, collections)`

Bases: `object`

Search for scenes in a SpatioTemporal Asset Catalog. Scenes are expected to be unpacked with a folder suffix `.SAFE`. The interface is kept consistent with `pyroSAR.drivers.Archive`.

#### Parameters

- **url** (`str`) – the catalog URL
- **collections** (`str` or `list[str]`) – the catalog collection(s) to be searched

`close()`

`select(sensor=None, product=None, acquisition_mode=None, mindate=None, maxdate=None, vectorobject=None, date_strict=True, check_exist=True)`

Select scenes from the catalog.

#### Parameters

- **sensor** (`str` or `list[str]` or `None`) – S1A or S1B
- **product** (`str` or `list[str]` or `None`) – GRD or SLC
- **acquisition\_mode** (`str` or `list[str]` or `None`) – IW, EW or SM
- **mindate** (`str` or `datetime.datetime` or `None`) – the minimum acquisition date
- **maxdate** (`str` or `datetime.datetime` or `None`) – the maximum acquisition date
- **vectorobject** (`spatialist.vector.Vector` or `None`) – a geometry with which the scenes need to overlap
- **date\_strict** (`bool`) – treat dates as strict limits or also allow flexible limits to incorporate scenes whose acquisition period overlaps with the defined limit?
  - strict: `start >= mindate & stop <= maxdate`
  - not strict: `stop >= mindate & start <= maxdate`
- **check\_exist** (`bool`) – check whether found files exist locally?

**Returns** the locations of the scene directories with suffix `.SAFE`

**Return type** `list[str]`

See also:

`pystac_client.Client.search`

## 2.6 Metadata

### 2.6.1 Extraction

<code>calc_geolocation_accuracy</code>	Calculates the radial root mean square error, which is a target requirement of the CARD4L NRB specification (Item 4.3).
<code>calc_performance_estimates</code>	Calculates the performance estimates specified in CARD4L NRB 1.6.9 for all noise power images if available.
<code>etree_from_sid</code>	Retrieve the manifest and annotation XML data of a scene as a dictionary using an <code>pyroSAR.drivers.ID</code> object.
<code>extract_pslr_islr</code>	Extracts all values for Peak Side Lobe Ratio (PSLR) and Integrated Side Lobe Ratio (ISLR) from the annotation metadata of a scene and calculates the mean value for all swaths.
<code>find_in_annotation</code>	Search for a pattern in all XML annotation files provided and return a dictionary of results.
<code>geometry_from_vec</code>	Get geometry information for usage in STAC and XML metadata from a <code>spatialist.vector.Vector</code> object.
<code>get_header_size</code>	Gets the header size of a GeoTIFF file in bytes.
<code>get_prod_meta</code>	Returns a metadata dictionary, which is generated from the name of a product scene using a regular expression pattern and from a measurement GeoTIFF file of the same product scene using the <code>Raster</code> class.
<code>meta_dict</code>	Creates a dictionary containing metadata for a product scene, as well as its source scenes.
<code>vec_from_srccoords</code>	Creates a single <code>Vector</code> object from a list of footprint coordinates of source scenes.

`S1_NRB.metadata.extract.calc_geolocation_accuracy(swath_identifier, ei_tif, dem_type, etad)`

Calculates the radial root mean square error, which is a target requirement of the CARD4L NRB specification (Item 4.3). For more information see: <https://s1-nrb.readthedocs.io/en/latest/general/geoaccuracy.html>. Currently only the Copernicus DEM is supported.

#### Parameters

- **swath\_identifier** (*str*) – Swath identifier dependent on acquisition mode.
- **ei\_tif** (*str*) – Path to the annotation GeoTIFF layer ‘Ellipsoidal Incident Angle’ of the current product.
- **dem\_type** (*str*) – The DEM type used for processing.
- **etad** (*bool*) – Was the ETAD correction applied?

**Returns** `rmse_planar` – The calculated rRMSE value rounded to two decimal places or `None` if a DEM other than Copernicus is used.

**Return type** `float` or `None`

`S1_NRB.metadata.extract.calc_performance_estimates(files)`

Calculates the performance estimates specified in CARD4L NRB 1.6.9 for all noise power images if available.

**Parameters** **files** (*list[str]*) – List of paths pointing to the noise power images the estimates should be calculated for.



**Returns out** – Dictionary containing the calculated estimates for each available polarization.

**Return type** dict

`S1_NRB.metadata.extract.etree_from_sid(sid)`

Retrieve the manifest and annotation XML data of a scene as a dictionary using an `pyroSAR.drivers.ID` object.

**Parameters sid** (`pyroSAR.drivers.ID`) – A pyroSAR ID object generated with e.g. `pyroSAR.drivers.identify()`.

**Returns** A dictionary containing the parsed `etree.ElementTree` objects for the manifest and annotation XML files.

**Return type** dict

`S1_NRB.metadata.extract.extract_pslr_islr(annotation_dict)`

Extracts all values for Peak Side Lobe Ratio (PSLR) and Integrated Side Lobe Ratio (ISLR) from the annotation metadata of a scene and calculates the mean value for all swaths.

**Parameters annotation\_dict** (dict) – A dictionary of annotation files in the form: {'swath ID': `lxml.etree._Element` object}

**Returns**

a tuple with the following values:

- pslr: Mean PSLR value for all swaths of the scene.
- islr: Mean ISLR value for all swaths of the scene.

**Return type** tuple[float]

`S1_NRB.metadata.extract.find_in_annotation(annotation_dict, pattern, single=False, out_type='str')`

Search for a pattern in all XML annotation files provided and return a dictionary of results.

**Parameters**

- **annotation\_dict** (dict) – A dict of annotation files in the form: {'swath ID': `lxml.etree._Element` object}
- **pattern** (str) – The pattern to search for in each annotation file.
- **single** (bool) – If True, the results found in each annotation file are expected to be the same and therefore only a single value will be returned instead of a dict. If the results differ, an error is raised. Default is False.
- **out\_type** (str) – Output type to convert the results to. Can be one of the following:
  - 'str' (default)
  - 'float'
  - 'int'

**Returns out** – A dictionary of the results containing a list for each of the annotation files. E.g., {'swath ID': list[str or float or int]}

**Return type** dict

`S1_NRB.metadata.extract.geometry_from_vec(vectorobject)`

Get geometry information for usage in STAC and XML metadata from a `spatialist.vector.Vector` object.

**Parameters vectorobject** (`spatialist.vector.Vector`) – The vector object to extract geometry information from.

**Returns out** – A dictionary containing the geometry information extracted from the vector object.

**Return type** dict



**S1\_NRB.metadata.extract.get\_header\_size(*tif*)**

Gets the header size of a GeoTIFF file in bytes. The code used in this function and its helper function `_get_block_offset` were extracted from the following source:

[https://github.com/OSGeo/gdal/blob/master/swig/python/gdal-utils/osgeo\\_utils/samples/validate\\_cloud\\_optimized\\_geotiff.py](https://github.com/OSGeo/gdal/blob/master/swig/python/gdal-utils/osgeo_utils/samples/validate_cloud_optimized_geotiff.py)

Copyright (c) 2017, Even Rouault

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

**Parameters** *tif* (*str*) – A path to a GeoTIFF file of the currently processed NRB product.

**Returns** *header\_size* – The size of all IFD headers of the GeoTIFF file in bytes.

**Return type** *int*

**S1\_NRB.metadata.extract.get\_prod\_meta(*product\_id*, *tif*, *src\_ids*, *rtc\_dir*)**

Returns a metadata dictionary, which is generated from the name of a product scene using a regular expression pattern and from a measurement GeoTIFF file of the same product scene using the `Raster` class.

**Parameters**

- **product\_id** (*str*) – The top-level product folder name.
- **tif** (*str*) – The path to a measurement GeoTIFF file of the product scene.
- **src\_ids** (*list*[*pyroSAR.drivers.ID*]) – List of `ID` objects of all source SLC scenes that overlap with the current MGRS tile.
- **rtc\_dir** (*str*) – A path pointing to the processed datasets of the product.

**Returns** A dictionary containing metadata for the product scene.

**Return type** *dict*

**S1\_NRB.metadata.extract.meta\_dict(*config*, *target*, *src\_ids*, *rtc\_dir*, *proc\_time*, *start*, *stop*, *compression*)**

Creates a dictionary containing metadata for a product scene, as well as its source scenes. The dictionary can then be utilized by `parse()` and `parse()` to generate XML and STAC JSON metadata files, respectively.

**Parameters**

- **config** (*dict*) – Dictionary of the parsed config parameters for the current process.
- **target** (*str*) – A path pointing to the NRB product scene being created.
- **src\_ids** (*list*[*pyroSAR.drivers.ID*]) – List of `ID` objects of all source scenes that overlap with the current MGRS tile.
- **rtc\_dir** (*str*) – The RTC processing output directory.
- **proc\_time** (*datetime.datetime*) – The processing time object used to generate the unique product identifier.
- **start** (*datetime.datetime*) – The product start time.
- **stop** (*datetime.datetime*) – The product stop time.
- **compression** (*str*) – The compression type applied to raster files of the product.

**Returns** *meta* – A dictionary containing a collection of metadata for product as well as source scenes.

**Return type** *dict*

`S1_NRB.metadata.extract.vec_from_srccoords(coord_list)`

Creates a single `Vector` object from a list of footprint coordinates of source scenes.

**Parameters** `coord_list` (`list[list[tuple[float]]]`) – List containing for each source scene a list of coordinate pairs as retrieved from the metadata stored in an `ID` object.

**Return type** `spatialist.vector.Vector`

## 2.6.2 XML

<code>parse</code>	Wrapper for <code>source_xml()</code> and <code>product_xml()</code> .
<code>product_xml</code>	Function to generate product-level metadata for an NRB product in <i>OGC 10-157r4</i> compliant XML format.
<code>source_xml</code>	Function to generate source-level metadata for an NRB product in <i>OGC 10-157r4</i> compliant XML format.

`S1_NRB.metadata.xml.parse(meta, target, tifs, exist_ok=False)`

Wrapper for `source_xml()` and `product_xml()`.

### Parameters

- **meta** (`dict`) – Metadata dictionary generated with `meta_dict()`.
- **target** (`str`) – A path pointing to the root directory of a product scene.
- **tifs** (`list[str]`) – List of paths to all GeoTIFF files of the currently processed NRB product.
- **exist\_ok** (`bool`) – Do not create files if they already exist?

`S1_NRB.metadata.xml.product_xml(meta, target, tifs, nsmmap, exist_ok=False)`

Function to generate product-level metadata for an NRB product in *OGC 10-157r4* compliant XML format.

### Parameters

- **meta** (`dict`) – Metadata dictionary generated with `meta_dict()`
- **target** (`str`) – A path pointing to the root directory of a product scene.
- **tifs** (`list[str]`) – List of paths to all GeoTIFF files of the currently processed NRB product.
- **nsmmap** (`dict`) – Dictionary listing abbreviation (key) and URI (value) of all necessary XML namespaces.
- **exist\_ok** (`bool`) – Do not create files if they already exist?

`S1_NRB.metadata.xml.source_xml(meta, target, nsmmap, exist_ok=False)`

Function to generate source-level metadata for an NRB product in *OGC 10-157r4* compliant XML format.

### Parameters

- **meta** (`dict`) – Metadata dictionary generated with `meta_dict()`
- **target** (`str`) – A path pointing to the root directory of a product scene.
- **nsmmap** (`dict`) – Dictionary listing abbreviation (key) and URI (value) of all necessary XML namespaces.
- **exist\_ok** (`bool`) – Do not create files if they already exist?

## 2.6.3 STAC

<code>parse</code>	Wrapper for <code>source_json()</code> and <code>product_json()</code> .
<code>product_json</code>	Function to generate product-level metadata for an NRB product in STAC compliant JSON format.
<code>source_json</code>	Function to generate source-level metadata for an NRB product in STAC compliant JSON format.
<code>make_catalog</code>	For a given directory of Sentinel-1 NRB products, this function will create a high-level STAC <code>Catalog</code> object serving as the STAC endpoint and lower-level STAC <code>Collection</code> objects for each subdirectory corresponding to a unique MGRS tile ID.

`S1_NRB.metadata.stac.make_catalog(directory, recursive=True, silent=False)`

For a given directory of Sentinel-1 NRB products, this function will create a high-level STAC `Catalog` object serving as the STAC endpoint and lower-level STAC `Collection` objects for each subdirectory corresponding to a unique MGRS tile ID.

WARNING: The directory content will be reorganized into subdirectories based on unique MGRS tile IDs if this is not yet the case.

### Parameters

- **directory** (*str*) – Path to a directory that contains Sentinel-1 NRB products.
- **recursive** (*bool*) – Search for NRB products in *directory* recursively? Default is True.
- **silent** (*bool*) – Should the output during directory reorganization be suppressed? Default is False.

**Returns** `nrb_catalog` – STAC Catalog object

**Return type** `pystac.catalog.Catalog`

### Notes

The returned STAC Catalog object contains Item asset hrefs that are absolute, whereas the actual on-disk files contain relative asset hrefs corresponding to the self-contained Catalog-Type. The returned in-memory STAC Catalog object deviates in this regard to ensure compatibility with the stackstac library: <https://github.com/gjoseph92/stackstac/issues/20>

`S1_NRB.metadata.stac.parse(meta, target, tifs, exist_ok=False)`

Wrapper for `source_json()` and `product_json()`.

### Parameters

- **meta** (*dict*) – Metadata dictionary generated with `meta_dict()`
- **target** (*str*) – A path pointing to the root directory of a product scene.
- **tifs** (*list[str]*) – List of paths to all GeoTIFF files of the currently processed NRB product.
- **exist\_ok** (*bool*) – Do not create files if they already exist?

`S1_NRB.metadata.stac.product_json(meta, target, tifs, exist_ok=False)`

Function to generate product-level metadata for an NRB product in STAC compliant JSON format.

### Parameters

- **meta** (*dict*) – Metadata dictionary generated with `meta_dict()`.
- **target** (*str*) – A path pointing to the root directory of a product scene.

- **tifs** (*list[str]*) – List of paths to all GeoTIFF files of the currently processed NRB product.
- **exist\_ok** (*bool*) – Do not create files if they already exist?

`S1_NRB.metadata.stac.source_json(meta, target, exist_ok=False)`

Function to generate source-level metadata for an NRB product in STAC compliant JSON format.

#### Parameters

- **meta** (*dict*) – Metadata dictionary generated with `meta_dict()`.
- **target** (*str*) – A path pointing to the root directory of a product scene.
- **exist\_ok** (*bool*) – Do not create files if they already exist?

## 3.1 Exploring S1-NRB data cubes

### 3.1.1 Introduction

This example notebook will give a short demonstration of how S1-NRB products can be explored as on-the-fly data cubes with little effort by utilizing the STAC metadata provided with each product. It is not intended to demonstrate how to process the S1-NRB products in the first place. For this information please refer to the [usage instructions](#).

A lightning talk related to this topic has been given during the [Cloud-Native Geospatial Outreach Event 2022](#), which can be found [here](#).

Follow [this link](#) for a better visualization of this notebook!

**Sentinel-1 Normalised Radar Backscatter** Sentinel-1 Normalised Radar Backscatter (S1-NRB) is a newly developed Analysis Ready Data (ARD) product for the European Space Agency that offers high-quality, radiometrically terrain corrected (RTC) Synthetic Aperture Radar (SAR) backscatter and is designed to be compliant with the CEOS ARD for Land (CARD4L) [NRB specification](#). You can find more detailed information about the S1-NRB product [here](#).

**SpatioTemporal Asset Catalog (STAC)** All S1-NRB products include metadata in JSON format compliant with the [SpatioTemporal Asset Catalog \(STAC\)](#) specification. STAC uses several sub-specifications ([Item](#), [Collection](#) & [Catalog](#)) to create a hierarchical structure that enables efficient querying and access of large volumes of geospatial data.

### 3.1.2 Getting started

After following the [installation instructions](#) you need to install an additional package into the activated conda environment:

```
conda activate nrb_env
conda install stackstac
```

Let's assume you have a collection of S1-NRB scenes located on your local disk, a fileserver or somewhere in the cloud. As mentioned in the [Introduction](#), each S1-NRB scene includes metadata as a STAC Item, describing the scene's temporal, spatial and product specific properties.

The **only step necessary to get started** with analysing your collection of scenes, is the creation of STAC Collection and Catalog files, which connect individual STAC Items and thereby create a hierarchy of STAC objects. S1\_NRB includes the utility function [make\\_catalog](#), which will create these files for you. Please note that [make\\_catalog](#) expects a directory structure based on MGRS tile IDs, which allows for efficient data querying and access. After user confirmation it will take care of reorganizing your S1-NRB scenes if this directory structure doesn't exist yet.

```
[3]: import numpy as np
import stackstac
from S1_NRB.metadata.stac import make_catalog
```

(continues on next page)

(continued from previous page)

```
nrb_catalog = make_catalog(directory='./NRB_thuringia', silent=True)
```

WARNING:

./NRB\_thuringia

and the NRB products it contains will be reorganized into subdirectories based on  
 ↳ unique MGRS tile IDs if this directory structure does not yet exist.

Do you wish to continue? [yes|no] yes

```
#### New STAC endpoint created: ./NRB_thuringia/catalog.json
```

The STAC Catalog can then be used with libraries such as `stackstac`, which “turns a STAC Collection into a lazy `xarray.DataArray`, backed by `dask`”.

The term *lazy* describes a `method of execution` that only computes results when actually needed and thereby enables computations on larger-than-memory datasets. `xarray` is a Python library for working with labeled multi-dimensional arrays of data, while the Python library `dask` facilitates parallel computing in a flexible way.

Compatibility with `odc-stac`, a very `similar library` to `stackstac`, has also been implemented.

```
[4]: aoi = (10.638066, 50.708415, 11.686751, 50.975775)
ds = stackstac.stack(items=nrb_catalog, bounds_latlon=aoi,
                    dtype=np.dtype('float32'), chunksize=(-1, 1, 1024, 1024))
ds

[4]: <xarray.DataArray 'stackstac-f9b5b2607432a2a973be5982262095c8' (time: 121,
                                                                    band: 10,
                                                                    y: 3189, x: 7471)>
dask.array<fetch_raster_window, shape=(121, 10, 3189, 7471), dtype=float32,
↳ chunksize=(121, 1, 1024, 1024), chunktype=numpy.ndarray>
Coordinates: (12/55)
  * time                (time) datetime64[ns] 2020-01-03T1...
  id                   (time) <U57 'S1A_IW_NRB__1SDV_2020...
  * band               (band) <U25 'noise-power-vh' ... '...
  * x                  (x) float64 6.15e+05 ... 6.897e+05
  * y                  (y) float64 5.651e+06 ... 5.619e+06
  constellation        <U10 'sentinel-1'
  ...
  processing:facility   <U3 'FSU'
  raster:bands         (band) object [{ 'unit': 'dB', 'nod...
  title               (band) <U30 'Noise Power' ... 'Acq...
  file:header_size    (band) int64 6794 6794 ... 7642 6914
  file:byte_order      <U13 'little-endian'
  epsg                 int64 32632
Attributes:
  spec:      RasterSpec(epsg=32632, bounds=(614990.0, 5618680.0, 689700.0...
  crs:       epsg:32632
  transform: | 10.00, 0.00, 614990.00|\n| 0.00,-10.00, 5650570.00|\n| 0.0...
  resolution: 10.0
```

As you can see in the output above, the collection of S1-NRB scenes was successfully loaded as an `xarray.DataArray`. The metadata attributes included in all STAC Items are now available as coordinate arrays (see [here](#) for clarification of Xarray’s terminology) and can be utilized during analysis.

It is now possible to explore and analyse the S1-NRB data cube. The most important tools in this regard are the already mentioned `xarray` and `dask`. Both are widely used and a lot of tutorials and videos can be found online, e.g. in the `xarray` Docs (1, 2) or the [Pangeo Tutorial Gallery](#).

## 4.1 Changelog

### 4.1.1 1.3.0 | 2023-05-24

- SNAP RTC: increase DEM oversampling by a factor of two (#78)
- nrb.format: do not hardcode src\_nodata and read it from the data instead (#79)
- enable configuration via command line arguments (#80)
- improved date parsing (#81)
- scene search via STAC (#82)
- enhanced time filtering (#84)
- general processor improvements (#85)

[Full Changelog](#)

### 4.1.2 1.2.0 | 2022-12-29

- improved geometry handling (#71)
- DEM handling improvements (#72)
- GRD buffering by (#73)
- add DEM as additional output layer (#70)
- sigma0 processing and annotation layer configuration (#74)

[Full Changelog](#)

### 4.1.3 1.1.0 | 2022-09-29

- documentation improvements (#60)
- installation update (#61)
- Process restructuring (#63)
- minor structural changes and bug fixes (#65)
- documentation update reflecting the recent process restructuring (#66)
- renamed processing mode 'snap' to 'rtc' (#67)

[Full Changelog](#)

#### **4.1.4 1.0.2 | 2022-08-24**

- Fix error in handling of temporary VRTs (#50)
- Adjustments to VRT log scaling (#52)
- [metadata] read nodata values directly from files (instead of hard-coding them) (#53)
- use type identifier in scene-specific DEM file names (#55)
- Add VRT assets to STAC files (#56)
- Fix and improve metadata geometry handling (#57)
- SNAP 9 compatibility (#58)

[Full Changelog](#)

#### **4.1.5 1.0.1 | 2022-07-03**

- dem handling improvements (#45)

[Full Changelog](#)

#### **4.1.6 1.0.0 | 2022-06-23**

- Dockerfile to build S1\_NRB image (#27)
- adjustments to nodata value (#28)
- renamed XML tag 'nrb' to 's1-nrb' (#36)
- Metadata & Config Improvements (#30)
- Geolocation accuracy (#40)
- various bug fixes and documentation improvements

[Full Changelog](#)

#### **4.1.7 0.4.2 | 2022-06-16**

- Update documentation (#27)
- find unpacked .SAFE scenes in scene\_dir (instead of just .zip) (aea53a5)

[Full Changelog](#)

#### **4.1.8 0.4.1 | 2022-06-01**

- handle ETAD products as zip, tar, and SAFE (#25)
- set dem download authentication via env. variables (#26)
- various bug fixes

[Full Changelog](#)



#### 4.1.9 0.4.0 | 2022-05-30

- outsourced and restructured DEM preparation functionality (#18)
- outsourced ETAD correction to dedicated module (#19)
- XML validation & improvements (#17)
- Restructuring and cleanup (#20)
- outsourced NRB formatting to dedicated module (#21)
- extended acquisition mode support (#22)
- Set up sphinx documentation (#23)
- AOI scene selection (#24)

[Full Changelog](#)

#### 4.1.10 0.3.0 | 2022-03-30

- Updated metadata module (#9)
- Modified *prepare\_dem* interface (#10)
- Various improvements (#11)
- Modified working directory structure (#12)
- Updated *ancillary.py* (#13)
- Added ETAD correction (#14)
- Improved RGB composite (#15)
- Store DEM/WBM tiles in UTM zones different to the native MGRS zone (#16)

[Full Changelog](#)

#### 4.1.11 0.2.0 | 2022-03-03

[Full Changelog](#)

#### 4.1.12 0.1.0 | 2022-01-14

## INDICES AND TABLES

- genindex
- modindex
- search

## PYTHON MODULE INDEX

### S

- `S1_NRB.ancillary`, [25](#)
- `S1_NRB.archive`, [27](#)
- `S1_NRB.config`, [10](#)
- `S1_NRB.dem`, [21](#)
- `S1_NRB.etad`, [21](#)
- `S1_NRB.metadata.extract`, [28](#)
- `S1_NRB.metadata.stac`, [32](#)
- `S1_NRB.metadata.xml`, [31](#)
- `S1_NRB.nrb`, [17](#)
- `S1_NRB.processor`, [11](#)
- `S1_NRB.snap`, [11](#)
- `S1_NRB.tile_extraction`, [24](#)

## INDEX

### A

aoi\_from\_tile() (in module *S1\_NRB.tile\_extraction*), 24  
 authenticate() (in module *S1\_NRB.dem*), 21

### C

calc\_geolocation\_accuracy() (in module *S1\_NRB.metadata.extract*), 28  
 calc\_performance\_estimates() (in module *S1\_NRB.metadata.extract*), 28  
 calc\_product\_start\_stop() (in module *S1\_NRB.nrb*), 17  
 check\_acquisition\_completeness() (in module *S1\_NRB.ancillary*), 25  
 check\_scene\_consistency() (in module *S1\_NRB.ancillary*), 25  
 check\_spacing() (in module *S1\_NRB.ancillary*), 26  
 close() (*S1\_NRB.archive.STACArchive* method), 27  
 create\_acq\_id\_image() (in module *S1\_NRB.nrb*), 17  
 create\_data\_mask() (in module *S1\_NRB.nrb*), 18  
 create\_rgb\_vrt() (in module *S1\_NRB.nrb*), 18  
 create\_vrt() (in module *S1\_NRB.nrb*), 18

### D

description2dict() (in module *S1\_NRB.tile\_extraction*), 24

### E

etree\_from\_sid() (in module *S1\_NRB.metadata.extract*), 29  
 extract\_pslr\_islr() (in module *S1\_NRB.metadata.extract*), 29

### F

find\_datasets() (in module *S1\_NRB.snap*), 11  
 find\_in\_annotation() (in module *S1\_NRB.metadata.extract*), 29  
 format() (in module *S1\_NRB.nrb*), 19

### G

gdal\_conf() (in module *S1\_NRB.config*), 10  
 generate\_unique\_id() (in module *S1\_NRB.ancillary*), 26  
 geo() (in module *S1\_NRB.snap*), 12

geometry\_from\_vec() (in module *S1\_NRB.metadata.extract*), 29  
 get\_config() (in module *S1\_NRB.config*), 10  
 get\_datasets() (in module *S1\_NRB.nrb*), 20  
 get\_header\_size() (in module *S1\_NRB.metadata.extract*), 30  
 get\_keys() (in module *S1\_NRB.config*), 10  
 get\_max\_ext() (in module *S1\_NRB.ancillary*), 26  
 get\_metadata() (in module *S1\_NRB.snap*), 13  
 get\_prod\_meta() (in module *S1\_NRB.metadata.extract*), 30  
 grd\_buffer() (in module *S1\_NRB.snap*), 13  
 group\_by\_time() (in module *S1\_NRB.ancillary*), 26  
 gsr() (in module *S1\_NRB.snap*), 13

### L

log() (in module *S1\_NRB.ancillary*), 26

### M

main() (in module *S1\_NRB.processor*), 11  
 make\_catalog() (in module *S1\_NRB.metadata.stac*), 32  
 meta\_dict() (in module *S1\_NRB.metadata.extract*), 30  
 mli() (in module *S1\_NRB.snap*), 13  
 module  
   *S1\_NRB.ancillary*, 25  
   *S1\_NRB.archive*, 27  
   *S1\_NRB.config*, 10  
   *S1\_NRB.dem*, 21  
   *S1\_NRB.etad*, 21  
   *S1\_NRB.metadata.extract*, 28  
   *S1\_NRB.metadata.stac*, 32  
   *S1\_NRB.metadata.xml*, 31  
   *S1\_NRB.nrb*, 17  
   *S1\_NRB.processor*, 11  
   *S1\_NRB.snap*, 11  
   *S1\_NRB.tile\_extraction*, 24  
 mosaic() (in module *S1\_NRB.dem*), 22

### P

parse() (in module *S1\_NRB.metadata.stac*), 32  
 parse() (in module *S1\_NRB.metadata.xml*), 31  
 postprocess() (in module *S1\_NRB.snap*), 14  
 pre() (in module *S1\_NRB.snap*), 14  
 prepare() (in module *S1\_NRB.dem*), 22

`process()` (in module *S1\_NRB.etad*), 21  
`process()` (in module *S1\_NRB.snap*), 14  
`product_json()` (in module *S1\_NRB.metadata.stac*),  
32  
`product_xml()` (in module *S1\_NRB.metadata.xml*),  
31

## R

`rtc()` (in module *S1\_NRB.snap*), 16

## S

*S1\_NRB.ancillary*  
module, 25  
*S1\_NRB.archive*  
module, 27  
*S1\_NRB.config*  
module, 10  
*S1\_NRB.dem*  
module, 21  
*S1\_NRB.etad*  
module, 21  
*S1\_NRB.metadata.extract*  
module, 28  
*S1\_NRB.metadata.stac*  
module, 32  
*S1\_NRB.metadata.xml*  
module, 31  
*S1\_NRB.nrb*  
module, 17  
*S1\_NRB.processor*  
module, 11  
*S1\_NRB.snap*  
module, 11  
*S1\_NRB.tile\_extraction*  
module, 24  
`select()` (*S1\_NRB.archive.STACArchive* method), 27  
`set_logging()` (in module *S1\_NRB.ancillary*), 26  
`sgr()` (in module *S1\_NRB.snap*), 16  
`snap_conf()` (in module *S1\_NRB.config*), 10  
`source_json()` (in module *S1\_NRB.metadata.stac*),  
33  
`source_xml()` (in module *S1\_NRB.metadata.xml*), 31  
*STACArchive* (class in *S1\_NRB.archive*), 27

## T

`tile_from_aoi()` (in module  
*S1\_NRB.tile\_extraction*), 24  
`to_mgrs()` (in module *S1\_NRB.dem*), 23

## V

`vec_from_srccoords()` (in module  
*S1\_NRB.metadata.extract*), 30  
`vrt_add_overviews()` (in module  
*S1\_NRB.ancillary*), 27