S1_NRB Documentation

Release 1.1

the S1_NRB Developers

Sep 29, 2022

CONTENTS

S1_NRB is a prototype processor for the Sentinel-1 Normalised Radar Backscatter product. Further information about this product can be found here.

CHAPTER

ONE

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/.

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

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 | snap`

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

- **aoi_tiles** & **aoi_geometry**

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

Only one of these parameters is required and the one that is not used can be set to `None` or left empty. `aoi_tiles` can be used to define the area of interest via MGRS tile IDs, which must be provided comma-separated (e.g., `aoi_tiles = 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.

- **mindate** & **maxdate**

The time period to create S1-NRB products for. Allowed date formats are `%Y-%m-%d` and `%Y-%m-%dT%H:%M:%S`.

- **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** & **scene_dir**

Both need to be provided as full paths to existing directories. `work_dir` is the main directory in which any subdirectories and files are stored that are generated during processing. `scene_dir` will be searched recursively for any Sentinel-1 scenes using the regex pattern `'^S1[AB].*\.zip$'`.

- **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.

- **db_file**

Any Sentinel-1 scenes found in `scene_dir` will be stored in a 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`.

- **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 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 "Copernicus 30m Global DEM" and "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 asks for authentification during runtime if one of these options is selected.

- **gdal_threads**

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

- **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).

Users might create several 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:

```
s1_nrb --help
```

Print a help message for the CLI tool.

```
s1_nrb --version
```

Print the processor version.

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

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

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

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

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 code snippet below demonstrates the tile reading mechanism of function *S1_NRB.tile_extraction.extract_tile()* (using class `spatialist.vector.Vector` and function `spatialist.vector.wkt2vector()`):

```
from lxml import html
from spatialist.vector import Vector, wkt2vector

def extract_tile(kml, tile):
    with Vector(kml, driver='KML') as vec:
        feat = vec.getFeatureByAttribute('Name', tile)
        attrib = html.fromstring(feat.GetField('Description')
        attrib = [x for x in attrib.xpath('//tr/td//text()') if x != ' ']
        attrib = dict(zip(attrib[0::2], attrib[1::2]))
        feat = None
    return wkt2vector(attrib['UTM_WKT'], int(attrib['EPSG']))
```

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).

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`). The following functions are called in sequence:

- *S1_NRB.snap.mli()*: creates multilooked image files (MLIs) per polarization including
  - Orbit state vector enhancement
  - (GRD only) border noise removal
  - Calibration to beta naught
  - Thermal noise removal (including generation of noise equivalent sigma zero (NESZ) noise power images)
  - (SLC only) debursting and swath merging
  - Multilooking
- *S1_NRB.snap.rtc()*: radiometric terrain flattening. Output is backscatter in gamma naught RTC (γ^0_T) and sigma naught RTC (σ^0_T) as well as the scattering area (β^0/γ^0_T).
- *S1_NRB.snap.gsr()*: computation of the gamma-sigma ratio (σ^0_T/γ^0_T).
- *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.

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 subsetted 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 another 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

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 sigma naught RTC backscatter computation 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:

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

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

σ_{DEM} = DEM accuracy at 1σ (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 (σ_{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

1.4.3 References

- [1] 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

1.5 Folder Structure

The following describes the structure created to store intermediate and final files during a processor run. 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.

CHAPTER

TWO

API DOCUMENTATION

2.1 Configuration

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

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, optional*) – 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.**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

main	Main function that initiates and controls the processing workflow.

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

Main function that initiates and controls the processing workflow.

Parameters

- **config_file** (*str*) – Full path to a *config.ini* file.
- **section_name** (*str, optional*) – Section name of the *config.ini* file that processing parameters should be parsed from. Default is 'PROCESSING'.
- **debug** (*bool, optional*) – Set pyroSAR logging level to DEBUG? Default is False.

2.2.1 SNAP

core processing

process	Main function for RTC processing with SNAP.
geo	Geocoding.
gsr	Gamma-sigma ratio computation for either ellipsoidal and RTC sigma nought.
mli	Create a multi-looked image (MLI).
rtc	Radiometric Terrain Flattening.

ancillary functions

find_datasets	Find processed datasets for a scene in a certain CRS.
get_metadata	Get processing metadata needed for NRB metadata.
postprocess	Performs SLC 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
- dm: layover-shadow data mask

- ei: ellipsoidal incident angle
- gs: gamma-sigma ratio
- lc: local contributing area (aka scattering area)
- li: local incident angle
- np-hh: noise power HH polarization
- np-hv: noise power HV polarization
- np-vh: noise power VH polarization
- np-vv: noise power 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'*, ***bands*)

Geocoding.

Parameters

- **src** – 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
- **bands** – band ids for the input scenes in *args* as lists with keys bands<index>, e.g., bands1=['NESZ_VV'], bands2=['Gamma0_VV'], …

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.**gsr**(*src*, *dst*, *workflow*, *src_sigma=None*)

Gamma-sigma ratio computation for either ellipsoidal and 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 scene from which to read the sigma bands.

S1_NRB.snap.**mli**(*src*, *dst*, *workflow*, *spacing=None*, *rlks=None*, *azlks=None*, *allow_res_osv=True*)

Create a multi-looked image (MLI). The following operators are used (optional steps in brackets): Apply-Orbit-File(->Remove-GRD-Border-Noise)->Calibration->ThermalNoiseRemoval(->TOPSAR-Deburst->Multilook)

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*) – the number of range looks.
- **azlks** (*int*) – the number of azimuth looks.
- **allow_res_osv** (*bool*) – Also allow the less accurate RES orbit files to be used?

S1_NRB.snap.**postprocess**(*src*, *slc_clean_edges=True*, *slc_clean_edges_pixels=4*)

Performs SLC 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.
- **slc_clean_edges** (*bool*) – perform SLC edge cleaning?
- **slc_clean_edges_pixels** (*int*) – the number of pixels to erode during edge cleaning.

S1_NRB.snap.**process**(*scene*, *outdir*, *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*, *slc_clean_edges=True*, *slc_clean_edges_pixels=4*, *cleanup=True*)

Main function for RTC processing with SNAP.

Parameters

- **scene** (*str*) – The SAR scene file name.
- **outdir** (*str*) – The output directory for storing the final results.
- **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. Options:
  - DEM
  - gammaSigmaRatio
  - incidenceAngleFromEllipsoid
  - layoverShadowMask
  - localIncidenceAngle
  - projectedLocalIncidenceAngle
  - scatteringArea
- **allow_res_osv** (*bool*) – Also allow the less accurate RES orbit files to be used?
- **slc_clean_edges** (*bool*) – Erode noisy image edges? See `pyroSAR.snap.auxil.erode_edges()`. Does not apply to layover-shadow mask.
- **slc_clean_edges_pixels** (*int*) – The number of pixels to erode.
- **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_
↪21000101T000000_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*)

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?

2.2.2 NRB

calc_product_start_stop	Calculates the start and stop times of the NRB product.
create_acq_id_image	Creation of the Acquisition ID image.
create_data_mask	Creation of the Data Mask image.
create_rgb_vrt	Creation of the color composite VRT file.
create_vrt	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.
format	Finalizes the generation of Sentinel-1 NRB products after RTC processing has finished.
get_datasets	Identifies all source SLC/GRD scenes, finds matching output files in *datadir* and filters both lists depending on the actual overlap of each SLC footprint with the current MGRS tile geometry.

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

- **start** (*str*) – Start time of the NRB product formatted as *%Y%m%dT%H%M%S* in UTC.
- **stop** (*str*) – 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 SLC scenes and 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.
- **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, optional*) – 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*log10).
- **relpaths** (*bool, optional*) – 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, optional*) – 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, optional*) – 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, optional*) – arguments for *fun* passed as *PixelFunctionArguments*. Requires GDAL>=3.5 to be read.
- **options** (*dict, optional*) – Additional parameters passed to *gdal.BuildVRT*.
- **overviews** (*list[int], optional*) – Internal overview levels to be created for each raster file.
- **overview_resampling** (*str, optional*) – Resampling method for overview levels.

Examples

linear 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*, *multithread=True*, *compress=None*, *overviews=None*)

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 an individual scene or multiple, matching scenes (consecutive acquisitions).
- **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, optional*) – Path to a water body mask file with the dimensions of an MGRS tile.
- **multithread** (*bool, optional*) – 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, optional*) – Compression algorithm to use. See https://gdal.org/drivers/raster/gtiff.html#creation-options for options. Defaults to 'LERC_DEFLATE'.
- **overviews** (*list[int], optional*) – Internal overview levels to be created for each GeoTIFF file. Defaults to [2, 4, 9, 18, 36]

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

Return type str

S1_NRB.nrb.**get_datasets**(*scenes*, *datadir*, *tile*, *extent*, *epsg*)

Identifies all source SLC/GRD scenes, finds matching output files in *datadir* and filters both lists depending on the actual overlap of each SLC footprint with the current MGRS tile geometry.

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.
- **tile** (*str*) – ID of an 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

- **ids** (list[`pyroSAR.drivers.ID`]) – List of `ID` objects of all source SLC scenes that overlap with the current MGRS tile.
- **datasets** (*list[dict]*) – List of RTC processing output files that match each `ID` object of *scenes*. The format is a list of dictionaries per scene with keys as described by e.g. *S1_NRB.snap.find_datasets()*.

2.2.3 ETAD

process	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

mosaic	Create a new scene-specific DEM mosaic GeoTIFF file.
prepare	Downloads DEM 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

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*) – The coordinate reference system as an EPSG code.
- **kml_file** (*str*) – The KML file containing the MGRS tile geometries.
- **dem_dir** (*str*) – 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**(*geometries*, *dem_type*, *dem_dir*, *wbm_dir*, *kml_file*, *threads*, *epsg=None*, *username=None*, *password=None*)

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

Parameters

- **geometries** (*list[spatialist.vector.Vector]*) – A list of geometries for which to prepare the DEM tiles.
- **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*) – The WBM target directory.
- **kml_file** (*str*) – The KML file containing the MGRS tile geometries.
- **threads** (*int*) – The number of threads to pass to pyroSAR.auxdata.dem_create().
- **epsg** (*int, optional*) – The coordinate reference system as an EPSG code.
- **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'.

2.3 Tile Extraction

aoi_from_tiles	Returns the bounding box of a list of MGRS tile IDs as a Vector object.
description2dict	Convert the HTML description field of the MGRS tile KML file to a dictionary.
extract_tile	Extract one or multiple MGRS tiles from the global Sentinel-2 tiling grid and return it as a Vector object.
get_tile_dict	Creates a dictionary with information for each unique MGRS tile ID that is being processed (extent, epsg code) as well as alignment coordinates that can be passed to the *standardGridOriginX* and *standardGridOriginY* parameters of *S1_NRB.snap.process()*.
tiles_from_aoi	Return a list of MGRS tile IDs or vector objects overlapping with an area of interest (AOI) provided as a Vector object.

S1_NRB.tile_extraction.**aoi_from_tiles**(*kml*, *tiles*)

Returns the bounding box of a list of MGRS tile IDs as a Vector object.

Parameters

- **kml** (*str*) – Path to the Sentinel-2 tiling grid KML file.
- **tiles** (*list[str]*) – A list of unique MGRS tile IDs.

Return type 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

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.**extract_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

S1_NRB.tile_extraction.**get_tile_dict**(*kml_file*, *spacing*, *aoi_geometry=None*, *aoi_tiles=None*)

Creates a dictionary with information for each unique MGRS tile ID that is being processed (extent, epsg code) as well as alignment coordinates that can be passed to the *standardGridOriginX* and *standardGridOriginY* parameters of *S1_NRB.snap.process()*.

Parameters

- **kml_file** (*str*) – The KML file containing the MGRS tile geometries.
- **spacing** (*int*) – The target pixel spacing in meters, which is passed to the RTC processing function (e.g. *S1_NRB.snap.process()*).
- **aoi_geometry** (*str or None*) – A vector geometry file name.
- **aoi_tiles** (*list[str] or None*) – a list with MGRS tile names.

Returns **tile_dict** – The output dictionary containing information about each unique MGRS tile ID and alignment coordinates.

Return type dict

S1_NRB.tile_extraction.**tiles_from_aoi**(*vectorobject*, *kml*, *epsg=None*, *strict=True*, *return_geometries=False*)

Return a list of MGRS tile IDs or vector objects overlapping with an area of interest (AOI) provided as a `Vector` object.

Parameters

- **vectorobject** (*spatialist.vector.Vector*) – The vector object to read.
- **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 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)?

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

2.4 Ancillary Functions

generate_unique_id	Returns a unique product identifier as a hexa-decimal string generated from the time of execution in isoformat.
get_max_ext	Gets the maximum extent from a list of geometries.
log	Format and handle log messages during processing.
set_logging	Set logging for the current process.

S1_NRB.ancillary.**check_spacing**(*spacing*)

perform a 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 hexa-decimal string generated from the time of execution in isoformat. 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, optional*) – The buffer in degrees 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.**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** (*str*) – One of ['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 massage 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, optional*) – Set pyroSAR logging level to DEBUG? Default is False.

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

Return type logging.Logger

2.5 Metadata

2.5.1 Extraction

calc_geolocation_accuracy	Calculates the radial root mean square error, which is a target requirement of the CARD4L NRB specification (Item 4.3).
calc_performance_estimates	Calculates the performance estimates specified in CARD4L NRB 1.6.9 for all noise power images if available.
etree_from_sid	Retrieve the manifest and annotation XML data of a scene as a dictionary using an `ID` object.
extract_pslr_islr	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.
find_in_annotation	Search for a pattern in all XML annotation files provided and return a dictionary of results.
geometry_from_vec	Get geometry information for usage in STAC and XML metadata from a `spatialist.vector.Vector` object.
get_header_size	Gets the header size of a GeoTIFF file in bytes.
get_prod_meta	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.
meta_dict	Creates a dictionary containing metadata for a product scene, as well as its source scenes.
vec_from_srccoords	Creates a single `Vector` 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

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.

Return type float

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 ID object.

Parameters **sid** (pyroSAR.drivers.ID) – A pyroSAR ID object generated with 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

- **pslr** (*float*) – Mean PSLR value for all swaths of the scene.
- **islr** (*float*) – Mean ISLR value for all swaths of the scene.

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, optional*) – 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, optional*) – 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, 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 vectorobject.

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

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, 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

2.5.2 XML

parse	Wrapper for *source_xml()* and *product_xml()*.
product_xml	Function to generate product-level metadata for an NRB product in *OGC 10-157r4* compliant XML format.
source_xml	Function to generate source-level metadata for an NRB product in *OGC 10-157r4* 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, optional*) – Do not create files if they already exist?

S1_NRB.metadata.xml.**product_xml**(*meta*, *target*, *tifs*, *nsmap*, *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.
- **nsmap** (*dict*) – Dictionary listing abbreviation (key) and URI (value) of all necessary XML namespaces.

- **exist_ok** (*bool, optional*) – Do not create files if they already exist?

S1_NRB.metadata.xml.**source_xml**(*meta*, *target*, *nsmap*, *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.
- **nsmap** (*dict*) – Dictionary listing abbreviation (key) and URI (value) of all necessary XML namespaces.
- **exist_ok** (*bool, optional*) – Do not create files if they already exist?

2.5.3 STAC

parse	Wrapper for *source_json()* and *product_json()*.
product_json	Function to generate product-level metadata for an NRB product in STAC compliant JSON format.
source_json	Function to generate source-level metadata for an NRB product in STAC compliant JSON format.
make_catalog	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.

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, optional*) – Search for NRB products in *directory* recursively? Default is True.
- **silent** (*bool, optional*) – 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, optional*) – 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, optional*) – 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, optional*) – Do not create files if they already exist?

CHAPTER

THREE

EXAMPLES

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, will be tested in the near future.

[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.

CHAPTER

FOUR

ABOUT

4.1 Changelog

4.1.1 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

4.1.2 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

4.1.3 1.0.1 | 2022-07-03

- dem handling improvements (#45)

Full Changelog

4.1.4 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

4.1.5 0.4.2 | 2022-06-16

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

Full Changelog

4.1.6 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

4.1.7 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

4.1.8 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

4.1.9 0.2.0 | 2022-03-03

Full Changelog

4.1.10 0.1.0 | 2022-01-14

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

R

S

T

V