s1ard Documentation

Release 1.6

the s1ard Developers

May 17, 2024

CONTENTS

s1ard is a prototype processor to create the Sentinel-1 Analysis Ready Data (ARD) products Normalised Radar Backscatter (S1-NRB) and Ocean Radar Backscatter (S1-ORB). Further information about these products can be found here.

CHAPTER

ONE

GENERAL TOPICS

1.1 Installation

1.1.1 SNAP

s1ard requires ESA's Sentinels Application Platform (SNAP) software for SAR data processing. Version 1.0.0 has been developed based on SNAP 8. SNAP 9 is supported since version 1.0.2, SNAP 10 since version 1.7.0. 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=10
TARGET=~/SNAP"$VERSION"

INSTALLER=esa-snap_sentinel_linux-"$VERSION".0.0.sh
wget https://download.esa.int/step/snap/"$VERSION"_0/installers/"$INSTALLER"
bash $INSTALLER -q -dir $TARGET
$TARGET/bin/snap --nosplash --nogui --modules --update-all

# add SNAP location to the PATH environment variable in the .bashrc file
echo PATH=$PATH:$TARGET/snap/bin >> ~/.bashrc
```

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 s1ard

The s1ard 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.

Latest State on Github

1. Create and then activate the conda environment

```
conda env create --file https://raw.githubusercontent.com/SAR-ARD/s1ard/main/
↪environment.yaml
conda activate s1ard
```

2. Install the s1ard package into the environment

```
pip install git+https://github.com/SAR-ARD/s1ard.git
```

Specific Version

The package version can be changed as necessary. See the Tags section of the repository for available versions.

```
conda env create --file https://raw.githubusercontent.com/SAR-ARD/s1ard/v1.0.0/
↪environment.yaml
conda activate s1ard
pip install git+https://github.com/SAR-ARD/s1ard.git@v1.0.0
```

1.1.3 Docker

Both SNAP and s1ard can also be installed into a docker container using the Dockerfile that is provided with the package.

1.2 Usage

This section outlines how to configure and run the processor. Configuration is most conveniently kept in a `config.ini` configuration file but can also be modified via the command line.

Two different types of product were intended when developing the processor, Normalised Radar Backscatter (NRB) and Ocean Radar Backscatter (ORB). However, the processor does not strictly separate between them and products can be created that conform to both types.

To create an NRB product as defined by the CEOS ARD specification, the following configuration would be necessary:

```
mode = sar, nrb
measurement = gamma
annotation = dm, ei, em, id, lc, li, np, ratio
```

The generated backscatter is gamma nought RTC. Annotation layers are a data mask, ellipsoidal incident angle, elevation model, acquisition id mask, local contributing area, local incidence angle, noise power and a gamma-sigma ratio.

For ORB, the following configuration is foreseen:

```
mode = sar, orb
measurement = sigma
annotation = dm, id, ld, li, np, wm
```

Compared to NRB, the backscatter is now sigma nought RTC. The ellipsoidal incident angle is excluded because over ocean it is nearly identical to the local incident angle. Furthermore, the elevation model, local contributing area and backscatter ratio are excluded as they are not seen as necessary. Two new annotation layers are added. A look direction angle and a wind model.

See below for further details.

1.2.1 Configuration

Usage of the s1ard 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 s1ard package can be found here:

https://github.com/SAR-ARD/s1ard/blob/main/config.ini

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 s1ard package should at least have a dedicated section for processing related parameters. This section is by default named `[PROCESSING]`. 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.

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

Processing Section

mode

Options: `sar | nrb | orb`

This parameter determines what steps should be executed. `sar` will only start SAR preprocessing, whereas `nrb` and `orb` will only start ARD generation from existing SAR products preprocessed in `sar`. By defining both `sar` and one of the ARD modes as list, both SAR preprocessing and ARD generation can be run together:

```
mode = sar, nrb
```

scene

Define a single SAR scene filename instead of searching for scenes in a database. If this parameter is set, the 'mode' must be 'sar'. In case of a GRD, database search is still performed to collect neighbors.

aoi_tiles & aoi_geometry

Limit processing to a specific area of interest (AOI).

`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. Both parameters are optional and can be set to `None` or left empty. `aoi_tiles` overrides `aoi_geometry`. If neither is defined, all tiles overlapping with the scene search result are processed.

mindate & maxdate

Search for source scenes within the defined date range. 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.

datatake

The datatake ID of source scenes in hexadecimal representation, e.g. 04EBF7.

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.

tmp_dir, sar_dir, ard_dir, wbm_dir & log_dir

Processing 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., `ard_dir = ARD` will create the subdirectory `/<work_dir>/ARD`. 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 regular expression `'^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 ARD 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 authentication. 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 (γ^0_T) or sigma naught RTC (σ^0_T) backscatter.

annotation

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

- dm: data mask (six masks: not layover not shadow, layover, shadow, ocean, lakes, rivers)
- 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
- ld: range look direction angle
- li: local incident angle

- np: noise power (NESZ, per polarization)
- ratio: will automatically be replaced with the following, depending on selected `measurement`:
  - gs: gamma-sigma ratio: sigma0 RTC / gamma0 RTC (if `measurement = gamma`)
  - sg: sigma-gamma ratio: gamma0 RTC / sigma0 RTC (if `measurement = sigma`)
- wm: wind-modelled backscatter extracted from a Sentinel-1 OCN (ocean) product. The sub-product *owiNrcsCmod* is extracted, which is Ocean Wind (OWI) Normalised Radar Cross Section (NRCS) predicted using a CMOD model and ECMWF wind model data. For each OCN product, a Level-1 counterpart (SLC/GRD) exists. The OCN products and corresponding Level-1 products must be searchable in the same way via the two search options described above. If a sigma naught output layer exists (via `measurement = sigma` or *annotation* layer *ratio*), a co-polarization wind normalization ratio VRT is created by dividing the measurement by the wind-modelled backscatter.

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.

Metadata Section

format

A comma-separated list to define the metadata file formats to be created for each ARD product. Supported options:

- OGC: XML file according to OGC EO standard
- STAC: JSON file according to the SpatioTemporal Asset Catalog family of specifications

copy_original

Copy the original metadata of the source scene(s) into the ARD product directory? This will copy the manifest.safe file and annotation folder into the subdirectory: `/source/<ProductIdentifier>`.

access_url, licence, doi & processing_center

The metadata files created for each ARD product contain some fields that should not be hidden away and hardcoded with arbitrary values. Instead, they can be accessed here in order to more easily generate a complete set of metadata. These fields are mostly relevant if you want to produce ARD products systematically and make them available for others. If you don't see a need for them you can just leave the fields empty, use the default 'None' or delete this entire section.

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 s1ard package.

The following options are currently available.

Print a help message for the CLI tool:

```
s1ard --help
```

Print the processor version:

```
s1ard --version
```

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

```
s1ard -c /path/to/config.ini
```

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

```
s1ard -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`:

```
s1ard -c /path/to/config.ini --acq_mode IW --annotation dm,id
```

The argument *snap_gpt_args* is known to require an additional modification so that the - characters in the value are not mistaken for argument keys. In the example SNAP is instructed to use a maximum of 32GB memory, 20GB cache size and 16 threads.

```
s1ard -c /path/to/config.ini -- --snap_gpt_args "-J-Xmx32G -c 20G -x -q 16"
```

1.3 ARD Production

The following sections give a brief overview of the major components of creating a S1-NRB product. All steps are comprised in function *s1ard.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 *s1ard.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 `s1ard.archive.STACArchive`). See documentation section *Scene Search* for details.

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

s1ard offers a convenience function *`s1ard.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 *`s1ard.snap.process()`*. It will perform all necessary steps to generate radiometrically terrain corrected gamma/sigma 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:

- *`s1ard.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
- *`s1ard.snap.mli()`*: creates multi-looked image files (MLIs) per polarization if the target pixel spacing is larger than the source pixel spacing.
- *`s1ard.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).
- *`s1ard.snap.gsr()`*: computation of the gamma-sigma ratio (σ^0_T/γ^0_T).
- *`s1ard.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 *s1ard.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:

- *s1ard.snap.gsr()* may be replaced by *s1ard.snap.sgr()* to create a sigma-gamma ratio (γ^0_T/σ^0_T)

1.3.6 ARD Formatting

During SAR 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 or S1-ORB naming scheme. All steps are performed by `s1ard.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 SAR 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 conversion to other backscatter conventions using function `s1ard.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 Scene Search

1.4.1 Intro

Scene search is a central component of the s1ard package. The Level-1 SLC/GRD source scenes and the ARD products are in a many-to-many relationship. One source scene is covered by multiple ARD products and an individual ARD product will in most cases be covered by two source scenes (see *Figure 1*).

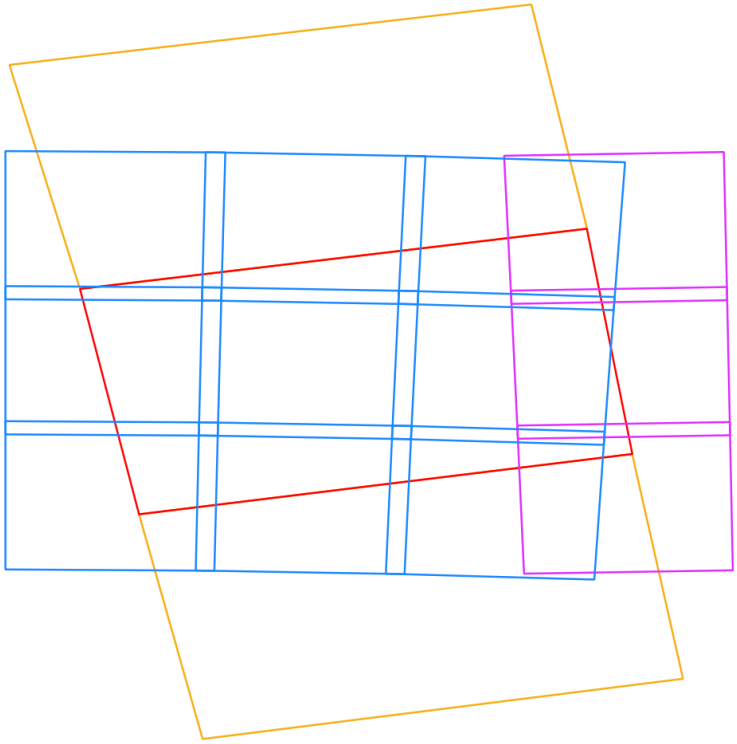

Fig. 1: Figure 1: Comparison of one central source scene (red), all tiles covering this scene in two different UTM zones (blue, purple), and two neighboring scenes needed for full coverage of all displayed tiles (yellow).

Hence, the processor must ensure that all source scenes relevant for a number of ARD products are processed. First, all locally available scenes are searched (see *config.ini* search options). Then, a check is performed to ensure all scenes were actually found by checking the data take ID. If the processor suspects a missing scene, it will cross-check with the ASF portal whether the scene is indeed missing. See *s1ard.search.check_acquisition_completeness()*.

Just providing a single scene to the processor is possible with the `scene` parameter, but this is only supported for `mode=sar`. For the ARD modes, multiple scenes are needed and the processor will need to collect them via the defined search method and parameters.

1.4.2 Search configuration with config.ini

For *Figure 1* we assume the following scene as the central one:

S1A_IW_GRDH_1SDV_20180829T170656_20180829T170721_023464_028DE0_F7BD

The following configuration in the *config.ini* file will select this central scene and its neighbors from the database and create 12 ARD products (as long as no geometry is defined via `aoi_tiles` or `aoi_geometry`). The search parameters match the acquisition characteristics of the scene. With `date_strict` we ensure that only scenes that were completely acquired in the defined time range are considered (i.e. not any earlier or later).

```
mindate = 20180829T170656
maxdate = 20180829T170721
date_strict = True
sensor = S1A
acq_mode = IW
product = GRD
scene_dir = path/to/scenes
db_file = scenes.db
```

1.4.3 Search with s1ard.search

In the background the *s1ard.search* module is used to do the scene search. This module contains various tools for searching Sentinel-1 scenes from multiple sources.

For the scene search option above (via `scene_dir` and `db_file`), the function *s1ard.search.scene_select()* and class `pyroSAR.drivers.Archive` are used for finding this scene and its neighbors:

```
from s1ard.search import scene_select
from pyroSAR.drivers import Archive
from spatialist.ancillary import finder

# SQLite database used for search
db_file = 'scenes.db'

# a folder containing Sentinel-1 scenes
scene_dir = '/path/to/scenes'

# KML file from the Sentinel-2 mission containing the MGRS tile geometries
kml_file = 'S2A_OPER_GIP_TILPAR_MPC__20151209T095117_V20150622T000000_21000101T000000_
↪B00.kml'

# find the Sentinel-1 scenes in the defined folder
scenes = finder(target=scene_dir, matchlist=['S1*.zip'])

# create/open the database file
with Archive(dbfile=db_file) as archive:
    # insert the found scenes into the database
    archive.insert(scenes)
    # search for scenes and overlapping MGRS tiles matching the defined parameters
    selection, aoi_tiles = scene_select(archive=archive, kml_file=kml_file,
                                        sensor='S1A', acquisition_mode='IW',
                                        product='GRD', mindate='20180829T170656',
                                        maxdate='20180829T170721', date_strict=True)
print('\n'.join(selection))
print(aoi_tiles)
```

This will output the three scenes and the 12 tiles displayed above:

```
/path/to/scenes/S1A_IW_GRDH_1SDV_20180829T170631_20180829T170656_023464_028DE0_9F36.
↪zip
/path/to/scenes/S1A_IW_GRDH_1SDV_20180829T170656_20180829T170721_023464_028DE0_F7BD.
↪zip
/path/to/scenes/S1A_IW_GRDH_1SDV_20180829T170721_20180829T170746_023464_028DE0_5310.
↪zip
['32TNR', '32TNS', '32TNT', '32TPR', '32TPS', '32TPT', '32TQR', '32TQS', '32TQT',
↪'33TUL', '33TUM', '33TUN']
```

1.4.4 STAC search

Note:

For full STAC search, the scenes need to exist in the local file system (i.e. not via e.g. HTTPS).
The shown examples can only be reproduced on DLR's terrabyte platform.

Similarly, scene search can be conducted using STAC. In the *config.ini*, the parameters `scene_dir` and `db_file` need to be replaced with `stac_catalog` and `stac_collections`:

```
stac_catalog = https://stac.terrabyte.lrz.de/public/api
stac_collections = sentinel-1-grd
```

Internally, the search interface class *s1ard.search.STACArchive* is used instead of pyroSAR.drivers.Archive as in the example above:

```
from s1ard.search import STACArchive

stac_catalog = 'https://stac.terrabyte.lrz.de/public/api'
stac_collection = 'sentinel-1-grd'

with STACArchive(url=stac_catalog, collections=stac_collection) as archive:
    selection, aoi_tiles = scene_select(archive=archive, ...
```

1.4.5 Basic scene search

Simple scene search (without selecting neighbors and MGRS tiles) can be done with the *select* methods of the respective driver classes.

pyroSAR

See pyroSAR.drivers.Archive.select().

```
from pyroSAR.drivers import Archive

db_file = 'scenes.db'
scene_dir = '/path/to/scenes'

with Archive(dbfile=db_file) as archive:
    selection = archive.select(sensor='S1A', acquisition_mode='IW',
                               product='GRD', mindate='20180829T170656',
                               maxdate='20180829T170721', date_strict=True)
print('\n'.join(selection))
```

STAC

See *s1ard.search.STACArchive.select()*.

Note:

`maxdate` is increased by one second because the STAC catalog time stamp is more precise than the defined one and `2018-08-29T17:07:21Z < 2018-08-29T17:07:21.014592Z`.
`check_exist=False` is defined to not check the existence of the scene in the local file system.

```
from s1ard.search import STACArchive

stac_catalog = 'https://stac.terrabyte.lrz.de/public/api'
stac_collection = 'sentinel-1-grd'

with STACArchive(url=stac_catalog, collections=stac_collection) as archive:
    selection = archive.select(sensor='S1A', acquisition_mode='IW',
                               product='GRD', mindate='20180829T170656',
                               maxdate='20180829T170722', date_strict=True,
                               check_exist=False)
print('\n'.join(selection))
```

ASF

See *s1ard.search.ASFArchive.select()*.

```
from s1ard.search import ASFArchive

with ASFArchive() as archive:
    selection = archive.select(sensor='S1A', acquisition_mode='IW',
                               product='GRD', mindate='20180829T170656',
                               maxdate='20180829T170722', date_strict=True)
print('\n'.join(selection))
```

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 the following output annotation layers enabled `annotation = dm, ei, em, id, lc, li, np, ratio`, thus creating S1-NRB products. 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.

1.6 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" [3]. 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)

1.6.1 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 [1] 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`

1.6.2 Development Status

The development status is tracked and discussed in the following Github issue: https://github.com/SAR-ARD/s1ard/issues/33

1.7 Equivalent Number of Looks (ENL)

The Equivalent Number of Looks (ENL) describes the degree of averaging applied to SAR measurements during data formation and postprocessing and is an indirect measure of speckle reduction (e.g., due to multilooking or speckle filtering).

In case of linear scaled backscatter data, ENL can be calculated as:

$$ENL = \frac{\mu^2}{\sigma^2}$$

where μ is the mean and σ is the standard deviation of the image. ([4], section A1.1.7)

The ENL value stored in the metadata of each S1-NRB product is calculated as suggested in [2], where ENL is first calculated for small pixel windows over the cross-polarized backscatter image and afterwards the median value of the distribution is selected.

1.7.1 Calculate ENL per image

While only the median value is currently stored in the metadata of each S1-NRB product, it is possible to calculate ENL as described above for entire images using the function *s1ard.metadata.extract.calc_enl()*. The following code example shows how to calculate ENL for 25x25 pixel windows and return the result as a numpy array. The visualization of the resulting array is shown in Figure 1.

```
from s1ard.metadata.extract import calc_enl

tif = "s1a-iw-nrb-20220721t051225-044194-05465e-33tuf-vh-s-lin.tif"
enl_arr = calc_enl(tif=tif, block_size=25, return_arr=True)
```

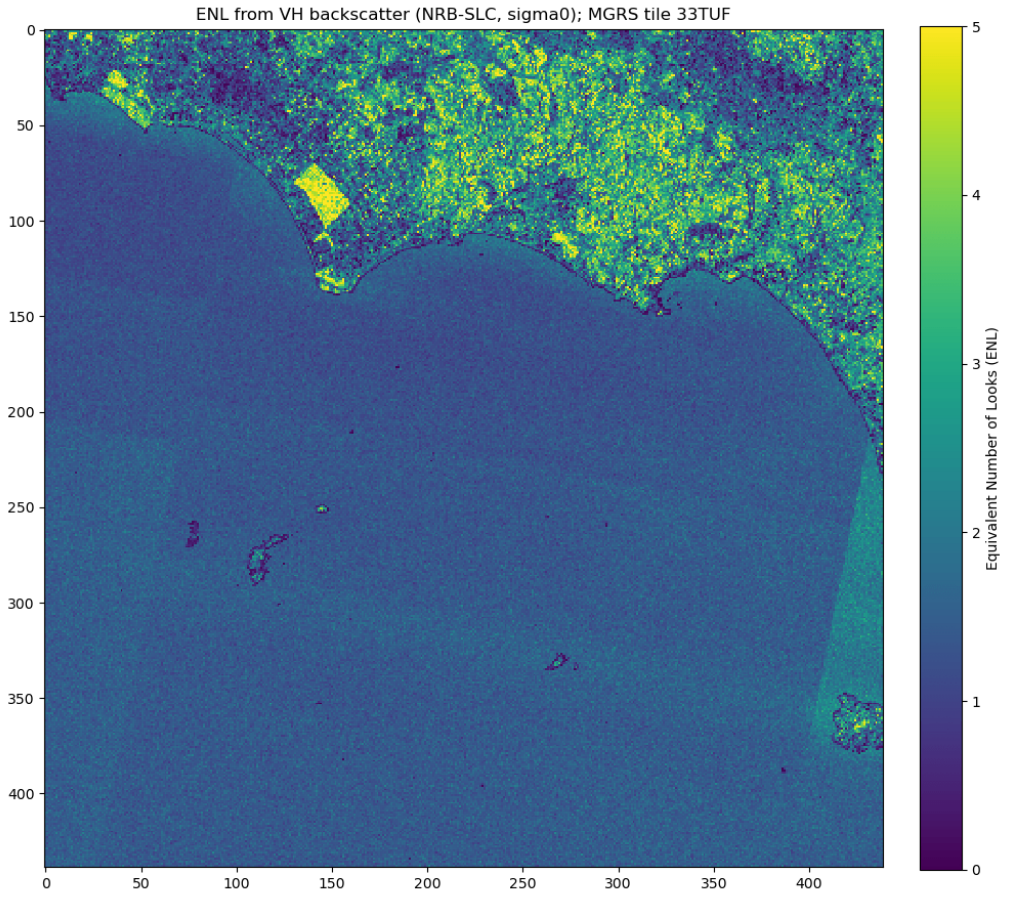

Fig. 2: Figure 1: Visualized ENL array for a S1-NRB product processed from a Sentinel-1A SLC scene in IW mode for MGRS tile 33TUF (coastline between Rome and Naples, Italy).

1.7.2 Comparison between GRDH and NRB

[4] provides estimates of ENL for different Sentinel-1 products (average over all swaths), e.g. ENL of 4.4 for GRDH in IW mode, and a description of the estimation process in section D1. The following shows a simple comparison between the GRDH product:

S1A_IW_GRDH_1SDV_20220721T051222_20220721T051247_044194_05465E_5807

and a S1-NRB product derived from the equivalent SLC product and processed for MGRS tile 33TUF:

S1A_IW_SLC__1SDV_20220721T051221_20220721T051249_044194_05465E_BACD

ENL was calculated for a selection of homogeneous forest areas, which are highlighted in Figure 2. The green outline traces the north-western corner of MGRS tile 33TUF (see Fig. 1). The resulting scatter plot (Figure 3) shows consistently higher ENL values for the GRDH product (Avg. ENL: 4.81) in comparison to the S1-NRB product (Avg. ENL: 4.59).

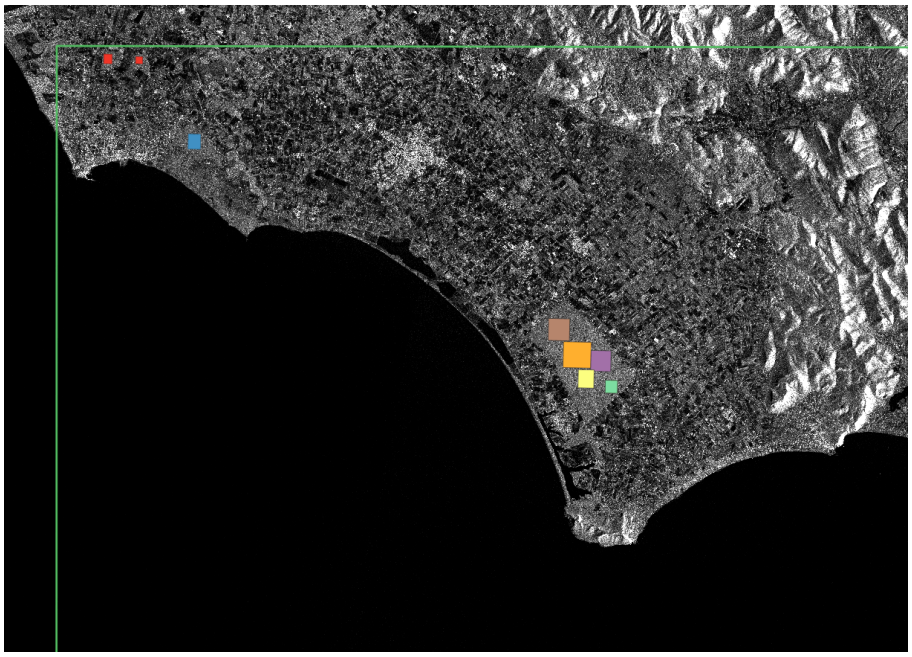

Fig. 3: Figure 2: Selection of homogeneous forest areas for ENL comparison between GRDH and NRB. Green outline: North-western corner of MGRS tile 33TUF; Background image: VH backscatter of the GRDH product.

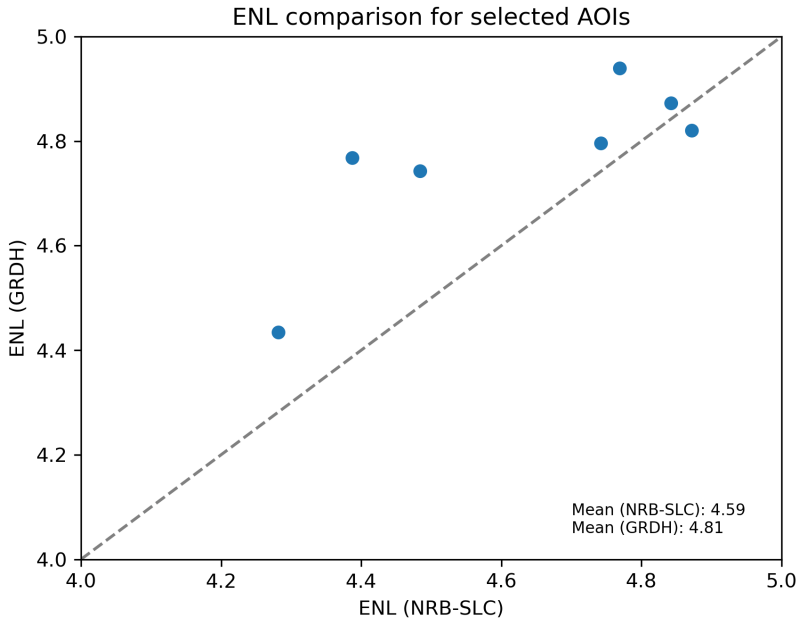

Fig. 4: Figure 3: Scatter plot comparing ENL values between GRDH and NRB, calculated for selected areas (see Fig. 2).

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 *s1ard.snap.process()* based on processing configurations provided by the config file.
get_config	Returns the content of a *config.ini* file as a dictionary.

s1ard.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

s1ard.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

s1ard.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]

s1ard.config.**snap_conf**(*config*)

Returns a dictionary of additional parameters for *s1ard.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 *s1ard.snap.process()*

Return type
dict

2.2 Processing

main	Main function that initiates and controls the processing workflow.

s1ard.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

process	Main function for SAR processing with SNAP.
geo	Range-Doppler geocoding.
grd_buffer	GRD extent buffering.
gsr	Gamma-sigma ratio computation for either ellipsoidal or RTC sigma nought.
mli	Multi-looking.
pre	General SAR preprocessing.
rtc	Radiometric Terrain Flattening.
sgr	Sigma-gamma ratio computation.
look_direction	Compute the per-pixel range look direction angle.

ancillary functions

find_datasets	Find processed datasets for a scene in a certain Coordinate Reference System (CRS).
get_metadata	Get processing metadata needed for ARD product metadata.
postprocess	Performs edge cleaning and sets the nodata value in the output ENVI HDR files.
nrt_slice_num	Compute a slice number for a scene acquired NRT Slicing mode.

s1ard.snap.**find_datasets**(*scene*, *outdir*, *epsg*)

Find processed datasets for a scene in a certain Coordinate Reference System (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)
- ld: range look direction angle
- 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

s1ard.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

s1ard.snap.**get_metadata**(*scene*, *outdir*)

Get processing metadata needed for ARD product metadata.

Parameters

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

Return type
dict

s1ard.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 subsetted to the extent of the main scene including a buffer.

Parameters

- **src** (*str*) – the file name of the source scene in BEAM-DIMAP format.
- **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

s1ard.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

s1ard.snap.**look_direction**(*dim*)

Compute the per-pixel range look direction angle. This adds a new layer to an existing BEAM-DIMAP product.

Steps performed:

- read geolocation grid points
- limit grid point list to those relevant to the image
- for each point, compute the range direction angle to the next point in range direction.
- interpolate the grid to the full image dimensions

Notes

- The interpolation depends on the location of the grid points relative to the image. Hence, by subsetting the image by an amount of pixels/lines different to the grid point sampling rate, the first and last points will no longer be in the first and last line respectively.
- The list might get very large when merging the scene with neighboring acquisitions using SliceAssembly and this longer list significantly changes the interpolation result. The difference in interpolation can be mitigated by reducing the list of points to those inside the image and those just outside of it.

Parameters

dim (*str*) – a BEAM-DIMAP metadata file (extension .dim)

s1ard.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`

s1ard.snap.**nrt_slice_num**(*dim*)

Compute a slice number for a scene acquired NRT Slicing mode. In this mode both *sliceNumber* and *totalSlices* are 0 in the manifest.safe file. *sliceNumber* is however needed in function *grd_buffer()* for the SNAP operator *SliceAssembly*. The time from *segmentStartTime* to *last_line_time* is divided by the acquisition duration (*last_line_time* - *first_line_time*). *totalSlices* is set to 100, which is expected to exceed the maximum possible value.

Parameters

dim (*str*) – the scene in BEAM-DIMAP format

s1ard.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.

s1ard.snap.**pre**(*src*, *dst*, *workflow*, *allow_res_osv=True*, *osv_continue_on_fail=False*, *output_noise=True*, *output_beta0=True*, *output_sigma0=True*, *output_gamma0=False*, *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?

- **output_sigma0** (*bool*) – output sigma nought backscatter needed for NESZ?
- **output_gamma0** (*bool*) – output gamma nought backscatter needed?
- **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`

s1ard.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 (γ^0_T)
  - sigma: RTC sigma nought (σ^0_T)
- **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 *s1ard.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: σ^0_T/γ^0_T
  - sigmaGammaRatio: γ^0_T/σ^0_T
  - incidenceAngleFromEllipsoid
  - layoverShadowMask
  - localIncidenceAngle
  - NESZ: noise equivalent sigma zero
  - projectedLocalIncidenceAngle
  - scatteringArea
  - lookDirection: range look direction angle
- **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 compeletely independently, gaps are introduced due to a missing overlap. If *neighbors* is None or an empty list, buffering is skipped.
- **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 s1ard 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)
```

s1ard.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

s1ard.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 ARD

calc_product_start_stop	Calculates the start and stop times of the ARD 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 Analysis Ready Data (ARD) products after SAR processing has finished.
get_datasets	Collect processing output for a list of scenes.
wind_normalization	Create wind normalization layers.

s1ard.ard.**calc_product_start_stop**(*src_ids*, *extent*, *epsg*)

Calculates the start and stop times of the ARD 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 and stop time of the ARD product formatted as *YYYYmmddTHHMMSS* in UTC.

Return type

tuple[str]

s1ard.ard.**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 ARD 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 ARD 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.

s1ard.ard.**create_data_mask**(*outname*, *datasets*, *extent*, *epsg*, *driver*, *creation_opt*, *overviews*, *overview_resampling*, *dst_nodata*, *product_type*, *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.
- **product_type** (*str*) – The type of ARD product that is being created. Either 'NRB' or 'ORB'.
- **wbm** (*str or None*) – Path to a water body mask file with the dimensions of an MGRS tile. Optional if *product_type='NRB', mandatory if `product_type='ORB'*.

s1ard.ard.**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.

s1ard.ard.**create_vrt**(*src*, *dst*, *fun*, *relpaths=False*, *scale=None*, *offset=None*, *dtype=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*) – 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'*.
- **dtype** (*str or None*) – the data type of the written VRT file; default None: same data type as source data. data type notations of GDAL (e.g. *Float32*) and numpy (e.g. *int8*) are supported.
- **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})
```

s1ard.ard.**format**(*config*, *product_type*, *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 Analysis Ready Data (ARD) products after SAR 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 ARD product directory structure & naming convention
- Generating metadata in XML and JSON formats for the ARD product as well as source SLC datasets

Parameters

- **config** (*dict*) – Dictionary of the parsed config parameters for the current process.
- **product_type** (*str*) – The type of ARD product to be generated. Options: 'NRB' or 'ORB'.
- **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 SAR 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 ARD 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, water)
  - ei: ellipsoidal incident angle
  - em: digital elevation model
  - id: acquisition ID image (source scene ID per pixel)
  - lc: RTC local contributing area

  - ld: range look direction angle
  - 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
  - wm: OCN product wind model; requires OCN scenes via argument *scenes_ocn*
- **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

s1ard.ard.**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 *datamask.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 SAR 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 SAR 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. *s1ard.snap.find_datasets()*.

See also:

s1ard.snap.find_datasets()

s1ard.ard.**wind_normalization**(*src*, *dst_wm*, *dst_wn*, *measurement*, *gapfill*, *bounds*, *epsg*, *driver*, *creation_opt*, *dst_nodata*, *multithread*, *resolution=915*)

Create wind normalization layers. A wind model annotation layer is created and optionally a wind normalization VRT.

Parameters

- **src** (*list[str]*) – A list of OCN products as prepared by *s1ard.ocn.extract()*
- **dst_wm** (*str*) – The name of the wind model layer in the ARD product

- **dst_wn** (*str or None*) – The name of the wind normalization VRT. If None, no VRT will be created. Requires *measurement* to point to a file.
- **measurement** (*str or None*) – The name of the measurement file used for wind normalization in *dst_wn*. If None, no wind normalization VRT will be created.
- **gapfill** (*bool*) – Perform additional gap filling (*s1ard.ocn.gapfill()*)? This is recommended if the Level-1 source product of *measurement* is GRD in which case gaps are introduced between subsequently acquired scenes.
- **bounds** (*list[float]*) – the bounds of the MGRS tile
- **epsg** (*int*) – The EPSG code of the MGRS tile
- **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.
- **dst_nodata** (*float*) – Nodata value to write to the output raster.
- **multithread** (*bool*) – Should *gdalwarp* use multithreading?
- **resolution** (*int, optional*) – The target pixel resolution in meters. 915 is chosen as default because it is closest to the OCN product resolution (1000) and still fits into the MGRS bounds (`109800 % 915 == 0`).

2.2.3 ETAD

process	Apply ETAD correction to a Sentinel-1 SLC product.

s1ard.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 and WBM tiles and restructures them into the MGRS tiling scheme including reprojection and vertical datum conversion.

s1ard.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]

s1ard.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 *s1ard.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().

s1ard.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 overalapping 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 s1ard 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:

s1ard.tile_extraction.tile_from_aoi

s1ard.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*)

2.2.5 OCN

extract	Extract an OCN product's image variable and write it to a new GeoTIFF file.
gapfill	Fill gaps of an image file using GDAL.

s1ard.ocn.**extract**(*src*, *dst*, *variable*)

Extract an OCN product's image variable and write it to a new GeoTIFF file. Coordinates are extracted from the corresponding latitude and longitude image variables and the corner coordinates written as ground control points (GCPs) to the output file.

Parameters

- **src** (*str*) – path to OCN product SAFE folder
- **dst** (*str*) – the name of the GeoTIFF file to write
- **variable** (*str*) – name of the layer to extract from the OCN product, e.g. *owiNrcsCmod*

s1ard.ocn.**gapfill**(*src*, *dst*, *md*, *si*)

Fill gaps of an image file using GDAL.

Parameters

- **src** (*str*) – the source image file
- **dst** (*str*) – the destination image file with gaps filled
- **md** (*int*) – the interpolation maximum distance
- **si** (*int*) – the number of smoothing iterations

See also:

osgeo.gdal.FillNodata

2.3 Tile Extraction

aoi_from_scene	Get processing AOIs for a SAR scene.
aoi_from_tile	Extract one or multiple MGRS tiles from the global Sentinel-2 tiling grid and return it as a `Vector` object.
description2dict	Convert the HTML description field of the MGRS tile KML file to a dictionary.
tile_from_aoi	Return a list of MGRS tile IDs or vector objects overlapping one or multiple areas of interest.

s1ard.tile_extraction.**aoi_from_scene**(*scene*, *kml*, *multi=True*, *percent=1*)

Get processing AOIs for a SAR scene. The MGRS grid requires a SAR scene to be geocoded to multiple UTM zones depending on the overlapping MGRS tiles and their projection. This function returns the following for each UTM zone group:

- the extent in WGS84 coordinates (key *extent*)
- the EPSG code of the UTM zone (key *epsg*)
- the Easting coordinate for pixel alignment (key *align_x*)
- the Northing coordinate for pixel alignment (key *align_y*)

A minimum overlap of the AOIs with the SAR scene is ensured by buffering the AOIs if necessary. The minimum overlap can be controlled with parameter *percent*.

Parameters

- **scene** (*pyroSAR.drivers.ID*) – the SAR scene object
- **kml** (*str*) – Path to the Sentinel-2 tiling grid KML file.
- **multi** (*bool*) – split into multiple AOIs per overlapping UTM zone or just one AOI covering the whole scene. In the latter case the best matching UTM zone is auto-detected (using function `spatialist.auxil.utm_autodetect()`).
- **percent** (*int or float*) – the minimum overlap in percent of each AOI with the SAR scene. See function *s1ard.ancillary.buffer_min_overlap()*.

Returns

a list of dictionaries with keys *extent*, *epsg*, *align_x*, *align_y*

Return type

list[dict]

s1ard.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

s1ard.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

s1ard.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

2.4 Ancillary Functions

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

s1ard.ancillary.**buffer_min_overlap**(*geom1*, *geom2*, *percent=1*)

Buffer a geometry to a minimum overlap with a second geometry. The geometry is iteratively buffered until the minimum overlap is reached. If the overlap of the input geometries is already larger than the defined threshold, a copy of the original geometry is returned.

Parameters

- **geom1** (*spatialist.vector.Vector*) – the geometry to be buffered
- **geom2** (*spatialist.vector.Vector*) – the reference geometry to intersect with
- **percent** (*int or float*) – the minimum overlap in percent of *geom1*

s1ard.ancillary.**buffer_time**(*start*, *stop*, ***kwargs*)

Time range buffering

Parameters

- **start** (*str*) – the start time in format '%Y%m%dT%H%M%S'
- **stop** (*str*) – the stop time in format '%Y%m%dT%H%M%S'
- **kwargs** – time arguments passed to `datetime.timedelta()`

s1ard.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 –

s1ard.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

s1ard.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

s1ard.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

s1ard.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]]

s1ard.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.

s1ard.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

s1ard.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

ASF	Simple SAR metadata handler for scenes in the ASF archive.
ASFArchive	Search for scenes in the Alaska Satellite Facility (ASF) catalog.
STACArchive	Search for scenes in a SpatioTemporal Asset Catalog.
asf_select	Search scenes in the Alaska Satellite Facility (ASF) data catalog.
check_acquisition_completeness	Check presence of neighboring acquisitions.
collect_neighbors	Collect a scene's neighboring acquisitions in a data take.
scene_select	Central scene search utility.

class s1ard.search.**ASF**(*meta*)

Bases: ID

Simple SAR metadata handler for scenes in the ASF archive. The interface is consistent with the driver classes in pyroSAR.drivers but does not implement the full functionality due to limited content of the CMR metadata catalog. Registered attributes:

- acquisition_mode
- coordinates
- frameNumber
- orbit
- orbitNumber_abs
- orbitNumber_rel
- polarizations
- product
- projection
- sensor
- start
- stop

scanMetadata()

scan SAR scenes for metadata attributes. The returned dictionary is registered as attribute *meta* by the class upon object initialization. This dictionary furthermore needs to return a set of standardized attribute keys, which are directly registered as object attributes.

Returns
the derived attributes

Return type
dict

class s1ard.search.**ASFArchive**

Bases: object

Search for scenes in the Alaska Satellite Facility (ASF) catalog.

static select(*sensor=None*, *product=None*, *acquisition_mode=None*, *mindate=None*, *maxdate=None*, *vectorobject=None*, *date_strict=True*, *return_value='url'*)

Select scenes from the ASF catalog. This is a simple wrapper around the function *asf_select()* to be consistent with the interfaces of *STACArchive()* and pyroSAR.drivers.Archive.

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
- **return_value** (*str or list[str]*) – the metadata return value; see *asf_select()* for details

See also:

asf_select

Returns
the scene metadata attributes as specified with *return_value*; see *asf_select()* for details

Return type
list[str or tuple[str] or *ASF*]

class s1ard.search.**STACArchive**(*url*, *collections*, *timeout=60*, *max_retries=20*)

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 *ASFArchive()* and pyroSAR.drivers.Archive.

Parameters

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

- **timeout** (*int*) – the allowed timeout in seconds
- **max_retries** (*int or None*) – the number of times to retry requests. Set to None to disable retries.

See also:

pystac_client.Client.open, pystac_client.stac_api_io.StacApiIO

close()

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

Select scenes from the catalog. Used STAC keys:

- platform
- start_datetime
- end_datetime
- sar:instrument_mode
- sar:product_type
- s1:datatake (custom)

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
- **frameNumber** (*int or list[int] or None*) – the data take ID in decimal representation. Requires custom STAC key *s1:datatake*.
- **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

s1ard.search.**asf_select**(*sensor*, *product*, *acquisition_mode*, *mindate*, *maxdate*, *vectorobject=None*, *return_value='url'*, *date_strict=True*)

Search scenes in the Alaska Satellite Facility (ASF) data catalog. This is a simple interface to the asf_search package.

Parameters

- **sensor** (*str*) – S1A or S1B
- **product** (*str*) – GRD or SLC
- **acquisition_mode** (*str*) – IW, EW or SM
- **mindate** (*str or datetime.datetime*) – the minimum acquisition date
- **maxdate** (*str or datetime.datetime*) – the maximum acquisition date
- **vectorobject** (*spatialist.vector.Vector or None*) – a geometry with which the scenes need to overlap
- **return_value** (*str or list[str]*) – the metadata return value; if *ASF*, an *ASF* object is returned; further string options specify certain properties to return: *beamModeType*, *browse*, *bytes*, *centerLat*, *centerLon*, *faradayRotation*, *fileID*, *flightDirection*, *groupID*, *granuleType*, *insarStackId*, *md5sum*, *offNadirAngle*, *orbit*, *pathNumber*, *platform*, *pointingAngle*, *polarization*, *processingDate*, *processingLevel*, *sceneName*, *sensor*, *startTime*, *stopTime*, *url*, *pgeVersion*, *fileName*, *frameNumber*; all options except *ASF* can also be combined in a list
- **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

Returns

the scene metadata attributes as specified with *return_value*; the return type is a list of strings, tuples or *ASF* objects depending on whether *return_type* is of type string, list or *ASF*.

Return type

list[str or tuple[str] or *ASF*]

s1ard.search.**check_acquisition_completeness**(*archive*, *scenes*)

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. In case a scene is suspected to be missing, the Alaska Satellite Facility (ASF) online catalog is cross-checked. An error will only be raised if the locally missing scene is present in the ASF catalog.

Parameters

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

Raises

RuntimeError –

See also:

s1ard.search.asf_select

s1ard.search.**collect_neighbors**(*archive*, *scene*)

Collect a scene's neighboring acquisitions in a data take.

Parameters

- **archive** (*pyroSAR.drivers.Archive or STACArchive or ASFArchive*) – an open scene archive connection
- **scene** (*pyroSAR.drivers.ID*) – the Sentinel-1 scene to be checked

Returns

the file names of the neighboring scenes

Return type
list[str]

s1ard.search.**scene_select**(*archive*, *kml_file*, *aoi_tiles=None*, *aoi_geometry=None*, ***kwargs*)

Central scene search utility. Selects scenes from a database and returns their file names together with the MGRS tile names for which to process ARD products. The list of MGRS tile names is either identical to the list provided with *aoi_tiles*, the list of all tiles overlapping with *aoi_geometry* or *vectorobject* (via *kwargs*), or the list of all tiles overlapping with an initial scene search result if no geometry has been defined via *aoi_tiles* or *aoi_geometry*. In the latter (most complex) case, the search procedure is as follows:

- perform a first search matching all other search parameters
- derive all MGRS tile geometries overlapping with the selection
- derive the minimum and maximum acquisition times of the selection as search parameters *mindate* and *maxdate*
- extend the *mindate* and *maxdate* search parameters by one minute
- perform a second search with the extended acquisition date parameters and the derived MGRS tile geometries

As consequence, if one defines the search parameters to only return one scene, the neighboring acquisitions will also be returned. This is because the scene overlaps with a set of MGRS tiles of which many or all will also overlap with these neighboring acquisitions. To ensure full coverage of all MGRS tiles, the neighbors of the scene in focus have to be processed too.

This function has three ways to define search geometries. In order of priority overriding others: *aoi_tiles* > *aoi_geometry* > *vectorobject* (via *kwargs*). In the latter two cases, the search geometry is extended to the bounding box of all MGRS tiles overlapping with the initial geometry to ensure full coverage of all tiles.

Parameters

- **archive** (*pyroSAR.drivers.Archive or STACArchive or ASFArchive*) – an open scene archive connection
- **kml_file** (*str*) – the KML file containing the MGRS tile geometries.
- **aoi_tiles** (*list[str] or None*) – a list of MGRS tile names for spatial search
- **aoi_geometry** (*str or None*) – the name of a vector geometry file for spatial search
- **kwargs** – further search arguments passed to pyroSAR.drivers.Archive.select() or *STACArchive.select()* or *ASFArchive.select()*

Returns

- the list of scenes
- the list of MGRS tiles

Return type
tuple[list[str], list[str]]

2.6 Metadata

2.6.1 Extraction

calc_enl	Calculate the Equivalent Number of Looks (ENL) for a linear-scaled backscatter measurement GeoTIFF file.
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.
calc_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.
copy_src_meta	Copies the original metadata of the source scenes to the ARD product directory.
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.
get_src_meta	Retrieve the manifest and annotation XML data of a scene as a dictionary using an `pyroSAR.drivers.ID` object.
meta_dict	Creates a dictionary containing metadata for a product scene, as well as its source scenes.

`s1ard.metadata.extract.`**`calc_enl`**(*tif*, *block_size=30*, *return_arr=False*, *decimals=2*)

Calculate the Equivalent Number of Looks (ENL) for a linear-scaled backscatter measurement GeoTIFF file. The calculation is performed block-wise for the entire image and by default the median ENL value is returned.

Parameters

- **`tif`** (*str*) – The path to a linear-scaled backscatter measurement GeoTIFF file.
- **`block_size`** (*int, optional*) – The block size to use for the calculation. Remainder pixels are discarded, if the array dimensions are not evenly divisible by the block size. Default is 30, which calculates ENL for 30x30 pixel blocks.
- **`return_arr`** (*bool, optional*) – If True, the calculated ENL array is returned. Default is False.
- **`decimals`** (*int, optional*) – Number of decimal places to round the calculated ENL value to. Default is 2.

Returns

The median ENL value or array of ENL values if *return_enl_arr* is True.

Return type

float or numpy.ndarray

References

[2]

s1ard.metadata.extract.**calc_geolocation_accuracy**(*swath_identifier*, *ei_tif*, *etad*, *decimals=2*)

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.
- **etad** (*bool*) – Was the ETAD correction applied?
- **decimals** (*int, optional*) – Number of decimal places to round the calculated rRMSE value to. Default is 2.

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

s1ard.metadata.extract.**calc_performance_estimates**(*files*, *decimals=2*)

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.
- **decimals** (*int, optional*) – Number of decimal places to round the calculated values to. Default is 2.

Returns

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

Return type

dict

s1ard.metadata.extract.**calc_pslr_islr**(*annotation_dict*, *decimals=2*)

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}
- **decimals** (*int, optional*) – Number of decimal places to round the calculated values to. Default is 2.

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]

s1ard.metadata.extract.**calc_wm_ref_stats**(*wm_ref_files*, *epsg*, *bounds*, *resolution=915*)

Calculates the mean wind model reference speed and direction for the wind model annotation layer.

Parameters

- **wm_ref_files** (*list[str]*) – List of paths pointing to the wind model reference files.
- **epsg** (*int*) – The EPSG code of the current MGRS tile.
- **bounds** (*list[float]*) – The bounds of the current MGRS tile.
- **resolution** (*int, optional*) – The resolution of the wind model reference files in meters. Default is 915.

Returns

a tuple with the following values in the following order:

- Mean wind model reference speed.
- Mean wind model reference direction.

Return type
tuple[float]

s1ard.metadata.extract.**copy_src_meta**(*ard_dir*, *src_ids*)

Copies the original metadata of the source scenes to the ARD product directory.

Parameters

- **ard_dir** (*str*) – A path pointing to the current ARD product directory.
- **src_ids** (*list[pyroSAR.drivers.ID]*) – List of ID objects of all source scenes that overlap with the current MGRS tile.

Return type
None

s1ard.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

s1ard.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

s1ard.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

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 ARD product.

Returns
header_size – The size of all IFD headers of the GeoTIFF file in bytes.

Return type
int

s1ard.metadata.extract.**get_prod_meta**(*product_id*, *tif*, *src_ids*, *sar_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.
- **sar_dir** (*str*) – A path pointing to the processed SAR datasets of the product.

Returns
A dictionary containing metadata for the product scene.

Return type
dict

s1ard.metadata.extract.**get_src_meta**(*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

s1ard.metadata.extract.**meta_dict**(*config*, *target*, *src_ids*, *sar_dir*, *proc_time*, *start*, *stop*, *compression*, *product_type*, *wm_ref_files=None*)

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 OGC 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 current ARD product directory.
- **src_ids** (*list[pyroSAR.drivers.ID]*) – List of ID objects of all source scenes that overlap with the current MGRS tile.
- **sar_dir** (*str*) – The SAR 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.
- **product_type** (*str*) – The type of ARD product that is being created. Either 'NRB' or 'ORB'.
- **wm_ref_files** (*list[str], optional*) – A list of paths pointing to wind model reference files. Default is None.

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

Return type
dict

2.6.2 XML

parse	Wrapper for *source_xml()* and *product_xml()*.
product_xml	Function to generate product-level metadata for an ARD product in *OGC 10-157r4* compliant XML format.
source_xml	Function to generate source-level metadata for an ARD product in *OGC 10-157r4* compliant XML format.

s1ard.metadata.xml.**parse**(*meta*, *target*, *assets*, *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.
- **assets** (*list[str]*) – List of paths to all GeoTIFF and VRT assets of the currently processed ARD product.
- **exist_ok** (*bool*) – Do not create files if they already exist?

s1ard.metadata.xml.**product_xml**(*meta*, *target*, *assets*, *nsmap*, *ard_ns*, *exist_ok=False*)

Function to generate product-level metadata for an ARD 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.
- **assets** (*list[str]*) – List of paths to all GeoTIFF and VRT assets of the currently processed ARD product.
- **nsmap** (*dict*) – Dictionary listing abbreviation (key) and URI (value) of all necessary XML namespaces.
- **ard_ns** (*str*) – Abbreviation of the ARD namespace. E.g., *s1-nrb* for the NRB ARD product.
- **exist_ok** (*bool*) – Do not create files if they already exist?

s1ard.metadata.xml.**source_xml**(*meta*, *target*, *nsmap*, *ard_ns*, *exist_ok=False*)

Function to generate source-level metadata for an ARD 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.
- **ard_ns** (*str*) – Abbreviation of the ARD namespace. E.g., *s1-nrb* for the NRB ARD product.
- **exist_ok** (*bool*) – Do not create files if they already exist?

2.6.3 STAC

parse	Wrapper for *source_json()* and *product_json()*.
product_json	Function to generate product-level metadata for an ARD product in STAC compliant JSON format.
source_json	Function to generate source-level metadata for an ARD product in STAC compliant JSON format.
make_catalog	For a given directory of Sentinel-1 ARD 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.

s1ard.metadata.stac.**make_catalog**(*directory*, *product_type*, *recursive=True*, *silent=False*)

For a given directory of Sentinel-1 ARD 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 the ARD type and unique MGRS tile IDs if this is not yet the case.

Parameters

- **directory** (*str*) – Path to a directory that contains ARD products.
- **product_type** (*str*) – Type of ARD products. Options: 'NRB' or 'ORB'.
- **recursive** (*bool, optional*) – Search *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

s1ard.metadata.stac.**parse**(*meta*, *target*, *assets*, *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.
- **assets** (*list[str]*) – List of paths to all GeoTIFF and VRT assets of the currently processed ARD product.
- **exist_ok** (*bool*) – Do not create files if they already exist?

s1ard.metadata.stac.**product_json**(*meta*, *target*, *assets*, *exist_ok=False*)

Function to generate product-level metadata for an ARD 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.
- **assets** (*list[str]*) – List of paths to all GeoTIFF and VRT assets of the currently processed ARD product.
- **exist_ok** (*bool*) – Do not create files if they already exist?

s1ard.metadata.stac.**source_json**(*meta*, *target*, *exist_ok=False*)

Function to generate source-level metadata for an ARD 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?

CHAPTER

THREE

EXAMPLES

3.1 Exploring s1ard 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 s1ard
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. `s1ard` 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 s1ard.metadata.stac import make_catalog
```

(continues on next page)

(continued from previous page)

```
nrb_catalog = make_catalog(directory='./NRB_thuringia', product_type='NRB',␣
↪silent=True)
```

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
```

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 1.6.2 | 2023-11-23

- Update metadata links (#165)
- Fix missing datamask layers in metadata (#164)
- Add wind normalisation metadata fields (#166)
- documentation updates (#167)
- [metadata.xml.product_xml] add geo acc. reference only if performed (#168)
- require pyroSAR>=0.23.0 (#169)

Full v1.6.2 Changelog

4.1.2 1.6.1 | 2023-11-17

- use relative paths in wind normalization VRT (#163)

Full v1.6.1 Changelog

4.1.3 1.6.0 | 2023-11-15

- central documentation literature management (#151)
- Use the official Continuum Docker base image (#152)
- re-introduce recently lost radiometric terrain correction (#154)
- strip line breaks from all parameters passed via the command line (#155)
- increase OCN gap fill distance (#156)
- data mask modifications (#157)
- [config] corrected list of allowed modes (#158)
- search OCN scenes by buffered start and stop time (#160)
- separate ocean, rivers and lakes into separate data mask bands (#161)

Full v1.6.0 Changelog

4.1.4 1.5.0 | 2023-10-12

- Replace *gs* and *sg* annotation options with *ratio* (#116)
- Metadata/review (#117)
- Equivalent Number of Looks (#113)
- [copy_src_meta] fixed bug in reading zip content on Windows (#124)
- Documentation: Table of abbreviations (#123)
- fixed bug in GRD buffering of ascending scenes (#126)
- new annotation layer "range look direction angle" (#103)
- ENL calculation: Suppress warnings and increase default block_size (#127)
- Add missing pyproj dependency (#128)
- Simplified datamask for ORB product (#122)
- Update .readthedocs.yaml (#129)
- [nrb.create_vrt] fixed bug in handling default 'options=None' (#132)
- [docs] point to right environment.yaml when installing specific version (#133)
- Fix missing STAC FileExtension entries (#131)
- Accommodate ORB product (#121)
- rename config default annotation IDs gs and sg to ratio (#135)
- [snap.process] skip GRD buffering if list is empty (#139)
- Refer to original source metadata in source XML and JSON (#136)
- wind normalization (#138)
- Look direction angle improvements (#141)
- do not look for source metadata files if copying is not user-configured (#142)
- change EW spacing from 20 to 40 m (#143)
- XML product metadata improvements (#137)
- Metadata/review (#140)
- wind normalization - GDAL options (#144)
- Require pyroSAR >=0.22.0 and update license year (#145)
- documentation improvements (#146)
- STACArchive file path handling (#148)
- geometry buffering for minimum overlap (#147)
- apply RTC to sigma0 (#149)
- config 'mode': removed 'all', added 'orb'; renamed module 'nrb' to 'ard' (#150)

Full v1.5.0 Changelog

4.1.5 1.4.0 | 2023-07-04

- various bug fixes (#94)
- datatake gap handling (#95)
- new configuration parameter 'datatake' (#96)
- increased STAC access robustness (#97)
- STACArchive bug fixes (#98)
- Optional *datatake* parameter (#99)
- bug fixes (#100)
- Bug fix to allow *annotation* = *None* (#102)
- Save original source metadata (#104)
- do not continue on error (#105)
- Always use ESA border noise removal (#106)
- [nrb] remove dataset if mask is nodata-only (#108)
- Bug fix: Save original source metadata (#109)
- New metadata config parameters (#110)
- support for scenes acquired in NRT Slicing mode (#112)

Full v1.4.0 Changelog

4.1.6 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 v1.3.0 Changelog

4.1.7 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 v1.2.0 Changelog

4.1.8 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 v1.1.0 Changelog

4.1.9 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 v1.0.2 Changelog

4.1.10 1.0.1 | 2022-07-03

- dem handling improvements (#45)

Full v1.0.1 Changelog

4.1.11 1.0.0 | 2022-06-23

- Dockerfile to build s1ard image (#31)
- 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 v1.0.0 Changelog

4.1.12 0.4.2 | 2022-06-16

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

Full v0.4.2 Changelog

4.1.13 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 v0.4.1 Changelog

4.1.14 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 v0.4.0 Changelog

4.1.15 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 v0.3.0 Changelog

4.1.16 0.2.0 | 2022-03-03

Full v0.2.0 Changelog

4.2 Abbreviations

Abbreviation	Meaning
AOI	Area Of Interest
ARD	Analysis Ready Data
ASF	Alaska Satellite Facility
CARD	CEOS Analysis Ready Data
CARD4L	CEOS Analysis Ready Data for Land
CEOS	Committee on Earth Observation Satellites
COG	Cloud Optimized GeoTIFF
CRS	Coordinate Reference System
DEM	Digital Elevation Model
DSM	Digital Surface Model
EGM	Earth Gravitational Model
EPSG	European Petroleum Survey Group Geodesy
ESA	European Space Agency
ETAD	Extended Timing Annotation Dataset
EW	Extra Wide Swath Mode
GETASSE	Global Earth Topography And Sea Surface Elevation
GRD	Ground Range Detected
ISLR	Integrated Side Lobe Ratio
IW	Interferometric Wide Swath Mode
KML	Keyhole Markup Language
MGRS	Military Grid Reference System
NESZ	Noise Equivalent Sigma Zero
NRB	Normalised Radar Backscatter
NRT	Near Real Time
OGC	Open Geospatial Consortium
ORB	Ocean Radar Backscatter
OSV	Orbit State Vector
PSLR	Peak Side Lobe Ratio
RTC	Radiometric Terrain Correction
SAR	Synthetic Aperture Radar
SLC	Single Look Complex
SM	Stripmap Mode
SNAP	Sentinel Application Platform
STAC	SpatioTemporal Asset Catalog
UTM	Universal Transverse Mercator
VRT	Virtual Raster Tile
WBM	Water Body Mask
WGS84	World Geodetic System 1984
WKT	Well Known Text
XML	Extensible Markup Language

BIBLIOGRAPHY

[1] Airbus. Copernicus DEM Product Handbook. Technical Report 5.0, Airbus, 2022. URL: https://spacedata.copernicus.eu/documents/20123/122407/GEO1988-CopernicusDEM-SPE-002_ProductHandbook_I5.0+%281%29.pdf/706ee17d-2cce-f1fa-a73e-1686d28f09dd?t=1679657087883.

[2] S.N. Anfinsen, A.P. Doulgeris, and T. Eltoft. Estimation of the Equivalent Number of Looks in Polarimetric Synthetic Aperture Radar Imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 47(11):3795–3809, 2009. doi:10.1109/TGRS.2009.2019269.

[3] CEOS. Analysis Ready Data for Land: Normalized Radar Backscatter. Technical Report 5.5, CEOS, 2021. URL: https://ceos.org/ard/files/PFS/NRB/v5.5/CARD4L-PFS_NRB_v5.5.pdf.

[4] CLS. Sentinel-1 Product Definition. Technical Report 2.7, CLS, 2016. URL: https://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/document-library/-/asset_publisher/1dO7RF5fJMbd/content/sentinel-1-product-definition.

PYTHON MODULE INDEX

S

INDEX

N

P

R

S

T

V

W