OpenSarToolkit

Release 0.12.15

Andreas Vollrath

Jan 05, 2022

CONTENTS


Open SAR Toolkit – Jupyter Notebook Tutorials

CHAPTER

ONE

OBJECTIVE

This python package lowers the entry barrier for accessing and pre-processing Sentinel-1 data for land applications and allows users with little knowledge on SAR and python to produce various Analysis-Ready-Data products.

CHAPTER

TWO

FUNCTIONALITY

The Open SAR Toolkit (OST) bundles the full workflow for the generation of Analysis-Ready-Data (ARD) of Sentinel-1 for Land in a single high-level python package. It includes functions for data inventory and advanced sorting as well as downloading from various mirrors. The whole pre-processing is bundled in a single function and different types of ARD can be selected, but also customised. OST does include advanced types of ARD such as combined production of calibrated backscatter, interferometric coherence and the dual-polarimetric H-A-Alpha decomposition. Time-series and multi-temporal statistics (i.e. timescans) can be produced for each of these layers and the generation of spatially-seamless large-scale mosaic over time is possible a well.

The Open SAR Toolkit realises this by using an object-oriented approach, providing classes for single scene processing, GRD and SLC batch processing routines. The SAR processing itself relies on ESA's Sentinel-1 Toolbox as well as some geospatial python libraries and the Orfeo Toolbox for mosaicking.

Please refer to our documentation to get started.

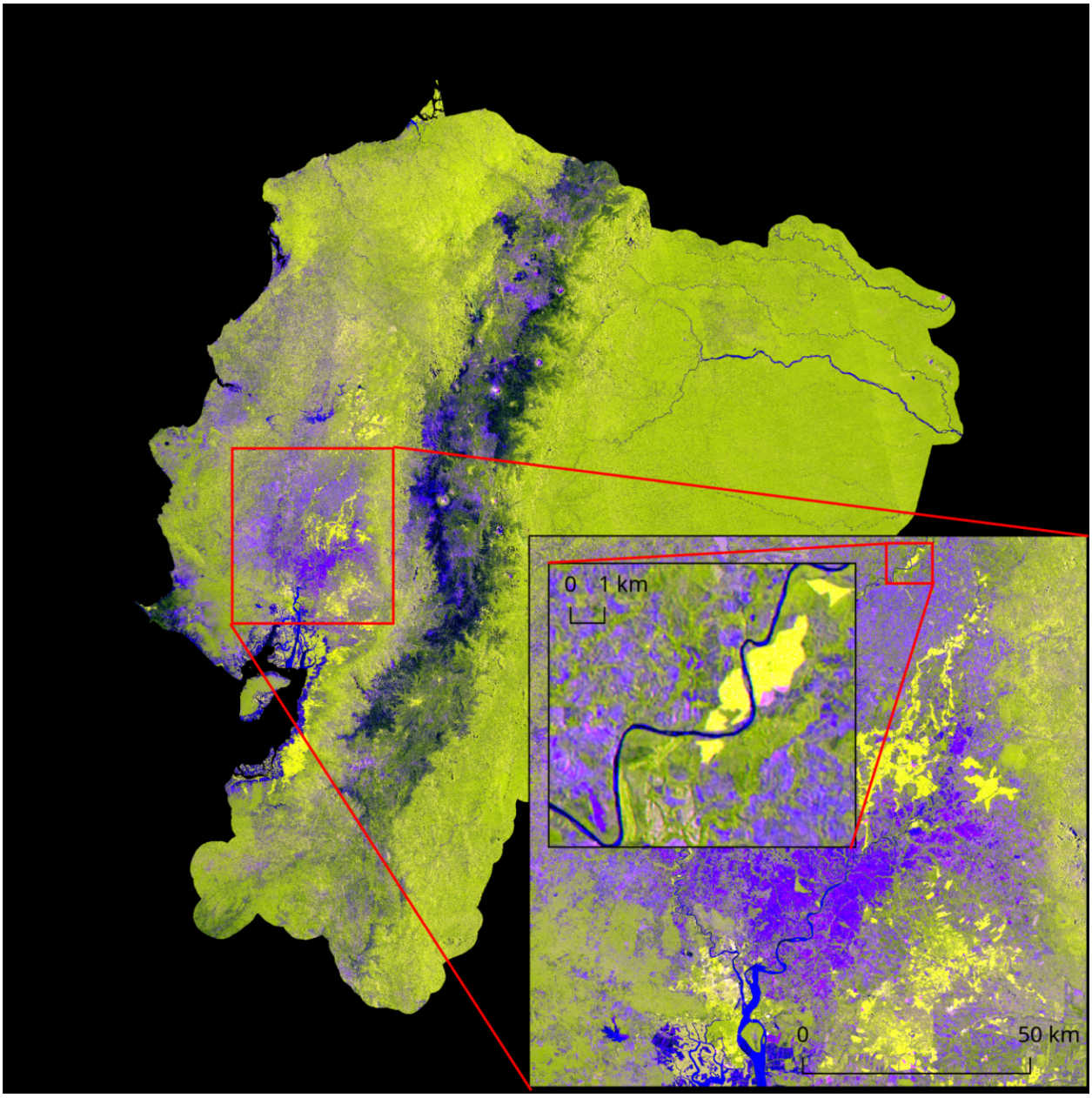
CHAPTER

THREE

EXAMPLES

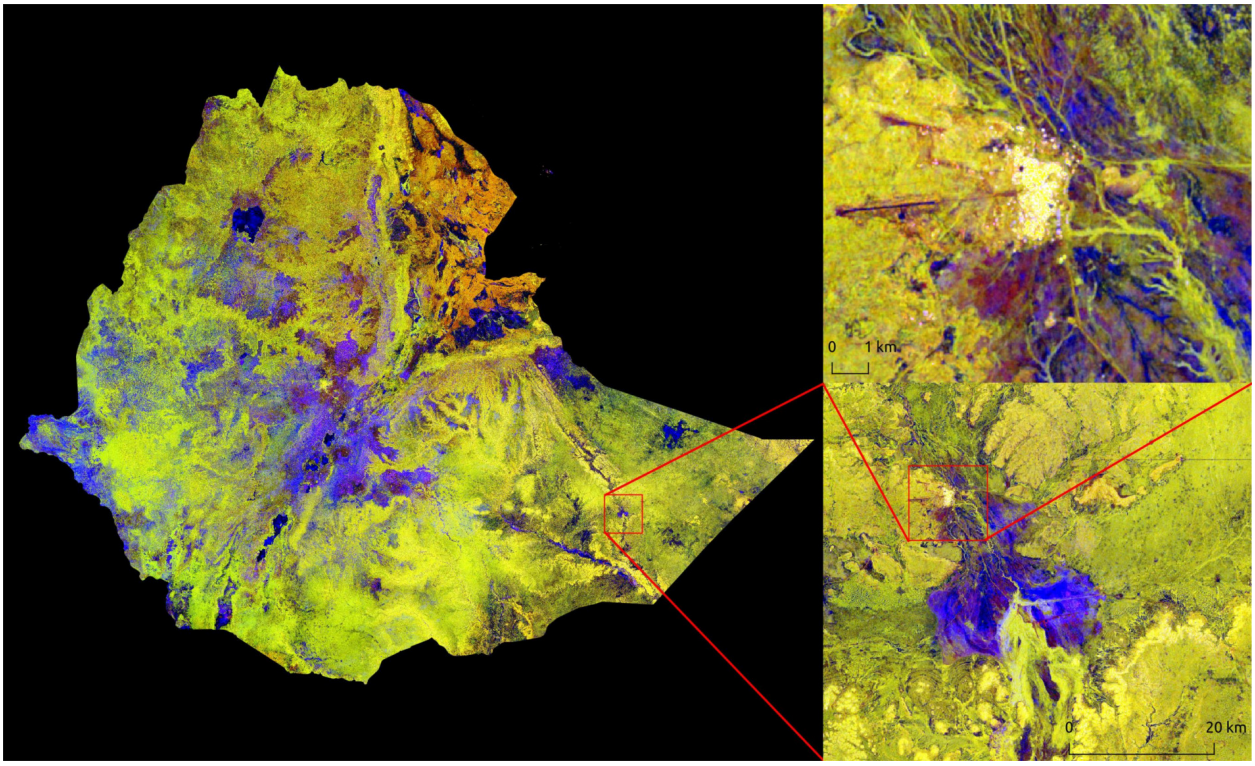
3.1 Ecuador VV-polarised Timescan Composite

- Year: 2016
- Sensor: Sentinel-1 C-Band SAR.
- Acquisitions: 6 acquisitions per swath (4 swaths)
- Output resolution: 30m
- RGB composite: - Red: VV-maximum - Green: VV-minimum - Blue: VV-Standard deviation



3.2 Ethiopia VV-VH polarised Timescan Composite

- Year: 2016-2017
- Sensor: Sentinel-1 C-Band SAR.
- Acquisitions: 7 acquisitions per swath (about 400 scenes over 8 swaths)
- Output resolution: 30m
- RGB composite: - Red: VV-minimum - Green: VH-minimum - Blue: VV-Standard deviation


0
1 km
0
20 km

CHAPTER

FOUR

ORIGIN OF THE PROJECT

Open SAR Toolkit was initially developed at the Food and Agriculture Organization of the United Nations under the SEPAL project between 2016-2018. It is still available there, but has been completely re-factored and transferred into a simpler and less-dependency rich **Python 3** version, which can be found on this page here. Instead of using R-Shiny as a GUI, the main interface are now Jupyter notebooks that are developed in parallel to this core package and should help to get started.

CHAPTER

FIVE

AUTHORS

meet our contributors.

5.1 Getting started

Warning: This documentation page is under construction.

Tip: For specific help, please open an issue on our repository by clicking on this link.

5.1.1 Installation

In this section the different ways of installing OST are presented.

Docker

Danger: Dockerhub is not permitting automatic builds. Therefore you need to build your own docker image using the DOCKERFIlE. The resulting docker image contains the full package, including ESA's Sentinel-1 Toolbox, Orfeo Toolbox, Jupyter Lab as well as the Open SAR Toolkit tutorial notebooks.

Docker installation is possible on various OS. Installation instructions can be found at https://docs.docker.com/install/

After docker is installed and running, launch the container with (adapt the path to the shared host folder and the name of the docke rimage at the very end):

```
docker run -it -p 8888:8888 -v /shared/folder/on/host:/home/ost/shared docker/image
```

The docker image automatically executes the jupyter lab and runs it on port 8888. You can find the address to the notebook on the command line where docker is running. Copy it into your favorites browser and replace 127.0.0.1 with localhost.

Manual installation

Dependencies

Sentinel Application Toolbox (SNAP)

OST bases mainly on the freely available SNAP toolbox for the SAR-specific processing routines. You can download SNAP from: http://step.esa.int/main/download/

If you install SNAP into the standard directory, OST should have no problems to find the SNAP command line executable. Otherwise you need to define the path to the gpt file on your own during processing.

Make sure to use SNAP 8 with the latest updates installed.

Orfeo Toolbox

If you want to create mosaics between different swaths, OST will rely on the `otbcli_Mosaic` command from The Orfeo Toolbox. You can download Orfeo from: https://www.orfeo-toolbox.org/download/

Make sure that the Orfeo bin folder is within your PATH variable to allow execution from command line.

Further dependencies (libs etc)

Ubuntu 18.04 and later:

```
sudo apt install python3-pip git libgdal-dev python3-gdal libspatialindex-dev nodejs npm␣
↪libgfortran5
```

Any Operating system using (mini)conda https://www.anaconda.com/:

```
conda install pip gdal jupyter jupyterlab git matplotlib numpy rasterio imageio rtree␣
↪geopandas fiona shapely matplotlib descartes tqdm scipy joblib retrying pytest pytest-
↪cov nodejs
```

OST installation

You can then use pip to install Open SAR Toolkit:

```
pip install opensartoolkit
```

5.1.2 Contribute

Introduction

First off, thank you for considering contributing to Active Admin. It's people like you that make Active Admin such a great tool. Following these guidelines helps to communicate that you respect the time of the developers managing and developing this open source project. In return, they should reciprocate that respect in addressing your issue, assessing changes, and helping you finalize your pull requests.

`OpenSarToolkit` is an open source project and we love to receive contributions from our community — you! There are many ways to contribute, from writing tutorials or blog posts, improving the documentation, submitting bug reports and feature requests or writing code which can be incorporated into the lib itself.

> **Warning:** Please, don't use the issue tracker for **support questions**. Instead, check if discussion channels can help with your issue.

Ground Rules

Responsibilities:

1. Ensure cross-platform compatibility for every change that's accepted. Windows, Mac, Debian & Ubuntu Linux.
2. Ensure that code that goes into core meets all requirements in the commitizen checklist
3. Create issues for any major changes and enhancements that you wish to make. Discuss things transparently and get community feedback.
4. Don't add any classes to the codebase unless absolutely needed. Err on the side of using functions.
5. Keep feature versions as small as possible, preferably one new feature per version.
6. Be welcoming to newcomers and encourage diverse new contributors from all backgrounds. See our Code of Conduct.

Your First Contribution

Working on your first Pull Request? You can learn how from this *free* series, How to Contribute to an Open Source Project on GitHub.

At this point, you're ready to make your changes! Feel free to ask for help; everyone is a beginner at first :smile_cat:! If a maintainer asks you to "rebase" your PR, they're saying that a lot of code has changed, and that you need to update your branch so it's easier to merge.

Getting started

Report a bug

> **Danger:** If you find a security vulnerability, do NOT open an issue. Email opensarkit@gmail.com instead.

When filing an issue, make sure to answer the questions predifined in the issue template, it will help us reproduce the bug and elp you debuging it.

If you find yourself wishing for a feature that doesn't exist in `OpenSarToolkit`, you are probably not alone. There are bound to be others out there with similar needs. Many of the features that `OpenSarToolkit` has today have been added because our users saw the need. Open an issue on our issues list on GitHub which describes the feature you would like to see, why you need it, and how it should work.

development env

To install the development environment of the `OpenSarToolkit` lib, create a new virtual environment:

```
$ cd OpenSarToolkit
$ python -m venv venv
(venv)$ source venv/bin/activate
```

Once in the venv, you can install `GDAL` (https://pypi.org/project/GDAL/) `SNAP` (http://step.esa.int/main/download/) and `ORFEO` (https://www.orfeo-toolbox.org/download/). then install the lib in development mode:

```
$ pip install -e .[dev]
```

This will install the `pre-commit` hooks that will be run each time you commit to the repository.

Note: You are not force to use en `venv` to run `ost` but make sure that your dependencies are compatible

pull request

For something that is bigger than a one or two line fix

1. Create your own fork of the code
2. Do the changes in your fork
3. If you like the change and think the project could use it:
   - Be sure you have followed the code style for the project.
   - run the test suit by running in the root folder of the lib:

     ```
     python -m pytest
     ```

   - Send a pull request using the provided template

Release

To publish an OST new version:

- Wait for the test to run and complete on `main`
- run the commitizen command locally

  ```
  cz bump
  ```

  You will see on your screen something like:

  ```
  bump: version 0.12.5 → 0.12.6
  tag to create: 0.12.6
  increment detected: PATCH
  ```

- Push to `main` (the commit is already created by the `cz bump` command)
- create a new release using the new tag name and the autogenerate report. It will trigger the publication on pipy.

Happy contribuing !

5.1.3 Contributors

Thanks goes to these wonderful people (emoji key):

This project follows the all-contributors specification. Contributions of any kind are welcome!

5.1.4 Contributor Covenant Code of Conduct

Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

Attribution

This Code of Conduct is adapted from the Contributor Covenant homepage, version 2.0.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

5.2 Content

Modules

5.2.1 ost.generic

Modules

ost.generic.ard_to_ts

Functions

ard_to_ts	
gd_ard_to_ts	

ost.generic.ard_to_ts.**ard_to_ts**(*list_of_files*, *burst*, *product*, *pol*, *config_file*)

ost.generic.ard_to_ts.**gd_ard_to_ts**(*list_of_args*, *config_file*)

ost.generic.common_wrappers

Functions

create_stack	**param file_list**
linear_to_db	Wrapper function around SNAP's linear to db routine
ls_mask	Wrapper function of a Snap graph for Layover/Shadow mask creation
mt_speckle_filter	**param in_stack**
speckle_filter	Wrapper function around SNAP's Speckle Filter function
terrain_correction	Wrapper function around Snap's terrain or ellipsoid correction
terrain_flattening	Wrapper function to Snap's Terrain Flattening routine

ost.generic.common_wrappers.**create_stack**(*file_list*, *out_stack*, *logfile*, *config_dict*, *polarisation=None*, *pattern=None*)

Parameters

- **file_list** –
- **out_stack** –
- **logfile** –
- **config_dict** –
- **polarisation** –
- **pattern** –

Returns

ost.generic.common_wrappers.**linear_to_db**(*infile*, *outfile*, *logfile*, *config_dict*)

Wrapper function around SNAP's linear to db routine

This function takes an OST calibrated Sentinel-1 product and converts it to dB.

Parameters

- **infile** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.generic.common_wrappers.**ls_mask**(*infile*, *outfile*, *logfile*, *config_dict*)

Wrapper function of a Snap graph for Layover/Shadow mask creation

Parameters

- **infile** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.generic.common_wrappers.**mt_speckle_filter**(*in_stack*, *out_stack*, *logfile*, *config_dict*)

Parameters

- **in_stack** –
- **out_stack** –
- **logfile** –
- **config_dict** –

Returns

ost.generic.common_wrappers.**speckle_filter**(*infile*, *outfile*, *logfile*, *config_dict*)

Wrapper function around SNAP's Speckle Filter function

This function takes OST imported Sentinel-1 product and applies the Speckle Filter as defind within the config dictionary.

Parameters

- **infile** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.generic.common_wrappers.**terrain_correction**(*infile*, *outfile*, *logfile*, *config_dict*)

Wrapper function around Snap's terrain or ellipsoid correction

Based on the configuration parameters either the Range-Doppler terrain correction or an Ellisoid correction is applied for geocoding a calibrated Sentinel-1 product.

Parameters

- **infile** –

- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.generic.common_wrappers.**terrain_flattening**(*infile*, *outfile*, *logfile*, *config_dict*)
Wrapper function to Snap's Terrain Flattening routine

Parameters

- **infile** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.generic.mosaic

Functions

create_timeseries_mosaic_vrt
gd_mosaic
gd_mosaic_slc_acquisition
mosaic
mosaic_slc_acquisition

ost.generic.mosaic.**create_timeseries_mosaic_vrt**(*list_of_args*)

ost.generic.mosaic.**gd_mosaic**(*list_of_args*)

ost.generic.mosaic.**gd_mosaic_slc_acquisition**(*list_of_args*)

ost.generic.mosaic.**mosaic**(*filelist*, *outfile*, *config_file*, *cut_to_aoi=None*, *harm=None*)

ost.generic.mosaic.**mosaic_slc_acquisition**(*track*, *date*, *product*, *outfile*, *config_file*)

ost.generic.timescan

Functions

date_as_float
deseasonalize

continues on next page

Table 6 – continued from previous page

difference_in_years	
gd_mt_metrics	
mt_metrics	**param stack**
nan_percentile	
remove_outliers	

ost.generic.timescan.**date_as_float**(*date*)

ost.generic.timescan.**deseasonalize**(*stack*)

ost.generic.timescan.**difference_in_years**(*start*, *end*)

ost.generic.timescan.**gd_mt_metrics**(*list_of_args*)

ost.generic.timescan.**mt_metrics**(*stack*, *out_prefix*, *metrics*, *rescale_to_datatype*, *to_power*, *outlier_removal*, *datelist*)

Parameters

- **stack** –
- **out_prefix** –
- **metrics** –
- **rescale_to_datatype** –
- **to_power** –
- **outlier_removal** –
- **datelist** –

Returns

ost.generic.timescan.**nan_percentile**(*arr*, *q*)

ost.generic.timescan.**remove_outliers**(*arrayin*, *stddev=2*, *z_threshold=None*)

ost.generic.ts_extent

Functions

mt_extent	

ost.generic.ts_extent.**mt_extent**(*list_of_extents*, *config_file*)

ost.generic.ts_ls_mask

Functions

mt_layover	

ost.generic.ts_ls_mask.**mt_layover**(*list_of_ls*)

5.2.2 ost.helpers

Modules

ost.helpers.asf	Functions for connecting and downloading from Alaska Satellite Facility
ost.helpers.asf_wget	This module provides functions for connecting and downloading from Alaska satellite Faciltity's Vertex server
ost.helpers.db	This script allows for the search of Sentinel-1 data on scihub.
ost.helpers.errors	OST specific errors and warnings
ost.helpers.helpers	This script provides core helper functions for the OST package.
ost.helpers.onda	Functions for downloading from ONDA Dias mirror.
ost.helpers.peps	Functions for connecting and downloading from CNES Peps server
ost.helpers.raster	Helper functions for raster data
ost.helpers.scihub	Functions for searching and downloading from Copernicus scihub.
ost.helpers.settings	
ost.helpers.srtm	Functions for searching and downloading from Copernicus scihub.
ost.helpers.vector	

ost.helpers.asf

Functions for connecting and downloading from Alaska Satellite Facility

Functions

asf_download	This function will download S1 products from ASF mirror. :param url: the url to the file you want to download :param filename: the absolute path to where the downloaded file should be written to :param uname: ESA's scihub username :param pword: ESA's scihub password :return:.
asf_download_parallel	
ask_credentials	Interactive function asking the user for ASF credentials
batch_download	
check_connection	Check if a connection with scihub can be established

ost.helpers.asf.**asf_download**(*url*, *filename*, *uname*, *pword*)
This function will download S1 products from ASF mirror. :param url: the url to the file you want to download :param filename: the absolute path to where the downloaded file should

be written to

Parameters

- **uname** – ESA's scihub username
- **pword** – ESA's scihub password

Returns

ost.helpers.asf.**asf_download_parallel**(*argument_list*)

ost.helpers.asf.**ask_credentials**()
Interactive function asking the user for ASF credentials

Returns tuple of username and password

Return type tuple

ost.helpers.asf.**batch_download**(*inventory_df*, *download_dir*, *uname*, *pword*, *concurrent=10*)

ost.helpers.asf.**check_connection**(*uname*, *pword*)
Check if a connection with scihub can be established

Parameters

- **uname** –
- **pword** –

Returns

ost.helpers.asf_wget

This module provides functions for connecting and downloading from Alaska satellite Faciltity's Vertex server

Functions

batch_download	
check_connection	A helper function to check if a connection can be established
s1_download	This function will download S1 products from ASF mirror.

Classes

SessionWithHeaderRedirection	A class that helps connect to NASA's Earthdata

ost.helpers.asf_wget.SessionWithHeaderRedirection

class ost.helpers.asf_wget.**SessionWithHeaderRedirection**(*username*, *password*)
A class that helps connect to NASA's Earthdata

Attributes

—

Methods

rebuild_auth	When being redirected we may want to strip authentication from the request to avoid leaking credentials.

SessionWithHeaderRedirection.**rebuild_auth**(*prepared_request*, *response*)
When being redirected we may want to strip authentication from the request to avoid leaking credentials. This method intelligently removes and reapplies authentication where possible to avoid credential loss.

ost.helpers.asf_wget.**batch_download**(*inventory_df*, *download_dir*, *uname*, *pword*, *concurrent=10*)

ost.helpers.asf_wget.**check_connection**(*uname*, *pword*)
A helper function to check if a connection can be established

Args: uname: username of ASF Vertex server pword: password of ASF Vertex server

Returns int: status code of the get request

ost.helpers.asf_wget.**s1_download**(*argument_list*)
This function will download S1 products from ASF mirror.

Parameters

- **url** – the url to the file you want to download

- **filename** – the absolute path to where the downloaded file should be written to
- **uname** – ESA's scihub username
- **pword** – ESA's scihub password

Returns

ost.helpers.db

This script allows for the search of Sentinel-1 data on scihub.

Based on some search parameters the script will create a query on www.scihub.copernicus.eu and return the results either as shapefile, sqlite, or PostGreSQL database.

Functions

pgHandler	This function connects to an existing PostGreSQL database, with the access parameters stored in the dbConnectFile as follows:

Classes

pgConnect	

ost.helpers.db.pgConnect

class ost.helpers.db.**pgConnect**(*uname=None*, *pword=None*, *dbname='sat'*, *host='localhost'*, *port='5432'*)

Attributes

—

Methods

pgCreateS1	
pgGetUUID	
pgDrop	
pgInsert	This function inserts a table into the connected database object.
pgSQL	This is a wrapper for a sql input that does get all responses.

continues on next page

Table 18 – continued from previous page

pgSQLnoResp	This is a wrapper for a sql input that does not get any response.
shpGeom2pg	This function is a wrapper to import a shapefile geometry to a PostGreSQL database
pgDateline	This function splits the acquisition footprint into a geometry collection if it crosses the dateline

pgConnect.**pgCreateS1**(*tablename*)

pgConnect.**pgGetUUID**(*sceneID*, *tablename*)

pgConnect.**pgDrop**(*tablename*)

pgConnect.**pgInsert**(*tablename*, *values*)
This function inserts a table into the connected database object.

pgConnect.**pgSQL**(*sql*)
This is a wrapper for a sql input that does get all responses.

pgConnect.**pgSQLnoResp**(*sql*)
This is a wrapper for a sql input that does not get any response.

pgConnect.**shpGeom2pg**(*aoi*, *tablename*)
This function is a wrapper to import a shapefile geometry to a PostGreSQL database

pgConnect.**pgDateline**(*tablename*, *uuid*)
This function splits the acquisition footprint into a geometry collection if it crosses the dateline

ost.helpers.db.**pgHandler**(*dbConnectFile='/home/docs/.phiSAR/pgdb'*)
This function connects to an existing PostGreSQL database, with the access parameters stored in the dbConnectFile as follows:

"database name" "database user" "database password" "database host" "database port"

Parameters **dbConnectFile** – path to the connect file

Returns the psycopg2 database connection object

ost.helpers.errors

OST specific errors and warnings

Exceptions

DownloadError	Raised when a download goes wrong.
GPTRuntimeError	Raised when a GPT process returns wrong return code.
NotValidFileError	Raised when an output file did not pass the validation test.

ost.helpers.errors.**DownloadError**()
Raised when a download goes wrong.

ost.helpers.errors.**GPTRuntimeError**(*message*)
Raised when a GPT process returns wrong return code.

ost.helpers.errors.**NotValidFileError**(*message*)
Raised when an output file did not pass the validation test.

ost.helpers.helpers

This script provides core helper functions for the OST package.

Functions

check_out_dimap	
check_out_tiff	
check_zipfile	
delete_dimap	Removes both dim and data from a Snap dimap file
delete_shapefile	Removes the shapefile and all its associated files
move_dimap	Function to move dimap's data and dim another locations
remove_folder_content	A helper function that cleans the content of a folder
resolution_in_degree	Convert resolution in meters to degree based on Latitude
run_command	**param command**
timer	A helper function to print a time elapsed statement

ost.helpers.helpers.**check_out_dimap**(*dimap_prefix*, *test_stats=True*)

ost.helpers.helpers.**check_out_tiff**(*file*, *test_stats=True*)

ost.helpers.helpers.**check_zipfile**(*filename*)

ost.helpers.helpers.**delete_dimap**(*dimap_prefix*)
Removes both dim and data from a Snap dimap file

ost.helpers.helpers.**delete_shapefile**(*shapefile*)
Removes the shapefile and all its associated files

ost.helpers.helpers.**move_dimap**(*infile_prefix*, *outfile_prefix*, *to_tif*)
Function to move dimap's data and dim another locations

ost.helpers.helpers.**remove_folder_content**(*folder*)
A helper function that cleans the content of a folder

Parameters **folder** –

ost.helpers.helpers.**resolution_in_degree**(*latitude*, *meters*)
Convert resolution in meters to degree based on Latitude

Parameters

- **latitude** –
- **meters** –

Returns

ost.helpers.helpers.**run_command**(*command*, *logfile=None*, *elapsed=True*)

Parameters

- **command** –

- **logfile** –
- **elapsed** –

Returns

ost.helpers.helpers.**timer**(*start*)
A helper function to print a time elapsed statement

Parameters **start** –

Returns

Return type str

ost.helpers.onda

Functions for downloading from ONDA Dias mirror.

Functions

ask_credentials	Interactive function asking the user for ONDA Dias credentials
batch_download	
check_connection	Check if a connection with ONDA Dias can be established
connect	Generates an opener for the Copernicus apihub/dhus
onda_download	Single scene download function for Copernicus scihub/apihub

ost.helpers.onda.**ask_credentials**()
Interactive function asking the user for ONDA Dias credentials

Returns tuple of username and password

Return type tuple

ost.helpers.onda.**batch_download**(*inventory_df*, *download_dir*, *uname*, *pword*, *concurrent=2*)

ost.helpers.onda.**check_connection**(*uname*, *pword*)
Check if a connection with ONDA Dias can be established

Parameters

- **uname** –
- **pword** –

Returns

ost.helpers.onda.**connect**(*uname=None*, *pword=None*)
Generates an opener for the Copernicus apihub/dhus

Parameters

- **uname** (*str*) – username of ONDA Dias
- **pword** (*str*) – password of ONDA Dias

Returns an urllib opener instance for Copernicus' scihub

Return type opener object

ost.helpers.onda.**onda_download**(*argument_list*)
Single scene download function for Copernicus scihub/apihub

Parameters **argument_list** –

a list with 4 entries (this is used to enable parallel execution) argument_list[0]: product's uuid argument_list[1]: local path for the download argument_list[2]: username of ONDA Dias argument_list[3]: password of ONDA Dias

Returns

ost.helpers.peps

Functions for connecting and downloading from CNES Peps server

Functions

ask_credentials	Interactive function asking the user for CNES' Peps credentials
batch_download	
check_connection	Check if a connection with CNES Pepscan be established
connect	Generates an opener for the Copernicus apihub/dhus
peps_download	Single scene download function for Copernicus scihub/apihub

ost.helpers.peps.**ask_credentials**()
Interactive function asking the user for CNES' Peps credentials

Returns tuple of username and password

Return type tuple

ost.helpers.peps.**batch_download**(*inventory_df*, *download_dir*, *uname*, *pword*, *concurrent=10*)

ost.helpers.peps.**check_connection**(*uname*, *pword*)
Check if a connection with CNES Pepscan be established

Parameters

- **uname** –
- **pword** –

Returns

ost.helpers.peps.**connect**(*uname=None*, *pword=None*)
Generates an opener for the Copernicus apihub/dhus

Parameters

- **uname** (*str*) – username of ONDA Dias
- **pword** (*str*) – password of ONDA Dias

Returns an urllib opener instance for Copernicus' scihub

Return type opener object

ost.helpers.peps.**peps_download**(*argument_list*)
Single scene download function for Copernicus scihub/apihub

Parameters **argument_list** –

a list with 4 entries (this is used to enable parallel execution) argument_list[0]: product's url argument_list[1]: local path for the download argument_list[2]: username of Copernicus' scihub argument_list[3]: password of Copernicus' scihub

Returns

ost.helpers.raster

Helper functions for raster data

Functions

calc_max	
calc_min	
combine_timeseries	
convert_to_db	Convert array of SAR power to decibel
create_rgb_jpeg	**param filelist**
create_timeseries_animation	
create_tscan_vrt	
fill_internal_nans	Function that fills no-data values with interpolation
get_max	
get_min	
image_bounds	Function to create a polygon of image boundary
mask_by_shape	Mask a raster layer with a vector file (including data conversions)
norm	Normalize array by its min/max or 2- and 98 percentile
outline	Generates a vector file with the valid areas of a raster file
polygonize_bounds	Polygonize a raster mask based on a mask value
polygonize_ls	
rescale_to_float	Re-convert a previously converted integer array back to float
scale_to_int	Convert a float array to integer by linear scaling between min and max
stretch_to_8bit	

continues on next page

Table 23 – continued from previous page

visualise_rgb	**param filepath**

ost.helpers.raster.**calc_max**(*band*, *stretch='minmax'*)

ost.helpers.raster.**calc_min**(*band*, *stretch='minmax'*)

ost.helpers.raster.**combine_timeseries**(*processing_dir*, *config_dict*, *timescan=True*)

ost.helpers.raster.**convert_to_db**(*pow_array*)
Convert array of SAR power to decibel

Parameters **pow_array** –

Returns

ost.helpers.raster.**create_rgb_jpeg**(*filelist*, *outfile=None*, *shrink_factor=1*, *resampling_factor=5*, *plot=False*, *date=None*, *filetype=None*)

Parameters

- **filelist** –
- **outfile** –
- **shrink_factor** –
- **resampling_factor** – 5 is average
- **plot** –
- **date** –
- **filetype** –

Returns

ost.helpers.raster.**create_timeseries_animation**(*timeseries_folder*, *product_list*, *out_folder*, *shrink_factor=1*, *resampling_factor=5*, *duration=1*, *add_dates=False*, *prefix=False*)

ost.helpers.raster.**create_tscan_vrt**(*timescan_dir*, *config_file*)

ost.helpers.raster.**fill_internal_nans**(*array*)
Function that fills no-data values with interpolation

Parameters **array** –

Returns

ost.helpers.raster.**get_max**(*file*, *dtype='float32'*)

ost.helpers.raster.**get_min**(*file*, *dtype='float32'*)

ost.helpers.raster.**image_bounds**(*data_dir*)
Function to create a polygon of image boundary

This function for all files within a dimap data directory

Parameters **data_dir** –

Returns

ost.helpers.raster.**mask_by_shape**(*infile*, *outfile*, *vector*, *to_db=False*, *datatype='float32'*, *rescale=True*, *min_value=1e-06*, *max_value=1*, *ndv=None*, *description=True*)

Mask a raster layer with a vector file (including data conversions)

Parameters

- **infile** –
- **outfile** –
- **vector** –
- **to_db** –
- **datatype** –
- **rescale** –
- **min_value** –
- **max_value** –
- **ndv** –
- **description** –

Returns

ost.helpers.raster.**norm**(*array*, *percentile=False*)

Normalize array by its min/max or 2- and 98 percentile

Parameters

- **array** –
- **percentile** –

Returns

ost.helpers.raster.**outline**(*infile*, *outfile*, *ndv=0*, *less_then=False*, *driver='GeoJSON'*)

Generates a vector file with the valid areas of a raster file

Parameters

- **infile** – input raster file
- **outfile** – output shapefile
- **ndv** – no-data-value
- **less_then** –
- **driver** –

Returns

ost.helpers.raster.**polygonize_bounds**(*infile*, *outfile*, *mask_value=1*, *driver='GeoJSON'*)

Polygonize a raster mask based on a mask value

Parameters

- **infile** –
- **outfile** –
- **mask_value** (*int/float, optional*) –
- **driver** (*str, optional*) –

Returns

ost.helpers.raster.**polygonize_ls**(*infile*, *outfile*, *driver='GeoJSON'*)

ost.helpers.raster.**rescale_to_float**(*int_array*, *data_type*)
Re-convert a previously converted integer array back to float

Parameters

- **int_array** –
- **data_type** –

Returns

ost.helpers.raster.**scale_to_int**(*float_array*, *min_value*, *max_value*, *data_type*)
Convert a float array to integer by linear scaling between min and max

Parameters

- **float_array** –
- **min_value** –
- **max_value** –
- **data_type** –

Returns

ost.helpers.raster.**stretch_to_8bit**(*file*, *layer*, *dtype*, *aut_stretch=False*)

ost.helpers.raster.**visualise_rgb**(*filepath*, *shrink_factor=25*)

Parameters

- **filepath** –
- **shrink_factor** –

Returns

ost.helpers.scihub

Functions for searching and downloading from Copernicus scihub.

Functions

ask_credentials	Interactive function asking the user for scihub credentials
batch_download	Batch download Sentinel-1 on the basis of an OST inventory GeoDataFrame
check_connection	Check if a connection with scihub can be established
connect	Generates an opener for the Copernicus apihub/dhus
create_aoi_str	Convert WKT formatted AOI to scihub's search url footprint attribute
create_s1_product_specs	Convert Sentinel-1's product metadata to scihub's product attributes
create_satellite_string	Convert mission_id to scihub's search url platformname attribute

continues on next page

Table 24 – continued from previous page

create_toi_str	Convert start and end date to scihub's search url time period attribute
next_page	Gets link for next page for results from apihub/scihub
s1_download	Single scene download function for Copernicus scihub/apihub
s1_download_parallel	Helper function for parallel download from scihub

ost.helpers.scihub.**ask_credentials**()
Interactive function asking the user for scihub credentials

Returns tuple of username and password

Return type tuple

ost.helpers.scihub.**batch_download**(*inventory_df*, *download_dir*, *uname*, *pword*, *concurrent=2*, *base_url='https://apihub.copernicus.eu/apihub'*)
Batch download Sentinel-1 on the basis of an OST inventory GeoDataFrame

Parameters

- **inventory_df** –
- **download_dir** –
- **uname** –
- **pword** –
- **concurrent** –
- **base_url** –

Returns

ost.helpers.scihub.**check_connection**(*uname*, *pword*, *base_url='https://apihub.copernicus.eu/apihub'*)
Check if a connection with scihub can be established

Parameters

- **uname** –
- **pword** –
- **base_url** –

Returns

ost.helpers.scihub.**connect**(*uname=None*, *pword=None*, *base_url='https://apihub.copernicus.eu/apihub'*)
Generates an opener for the Copernicus apihub/dhus

Parameters

- **uname** (*str*) – username of Copernicus' scihub
- **pword** (*str*) – password of Copernicus' scihub
- **base_url** –

Returns an urllib opener instance for Copernicus' scihub

Return type opener object

ost.helpers.scihub.**create_aoi_str**(*aoi*)
Convert WKT formatted AOI to scihub's search url footprint attribute

Parameters **aoi** (*WKT string*) – WKT representation of the Area Of Interest

Returns Copernicus' scihub compliant AOI query string

Return type str

ost.helpers.scihub.**create_s1_product_specs**(*product_type='*'*, *polarisation='*'*, *beam='*'*)
Convert Sentinel-1's product metadata to scihub's product attributes

Default values for all product specifications is the wildcard '*' in order to check for all

Parameters

- **product_type** (*str*) – Sentinel-1 product type (RAW, SLC, GRD), defaults to '*'
- **polarisation** (*string*) – Sentinel-1 polarisation mode (VV; VV VH; HH; HH HV), defaults to '*'
- **beam** (*str*) – Sentinel-1 beam mode (IW; SM, EW), defaults to '*'

Returns Copernicus' scihub compliant product specifications query string

Return type str

ost.helpers.scihub.**create_satellite_string**(*mission_id*)
Convert mission_id to scihub's search url platformname attribute

Parameters **mission_id** – an OST scene mission_id attribute (e.g. S1)

Returns Copernicus' scihub compliant satellite query string

Return type str

ost.helpers.scihub.**create_toi_str**(*start='2014-10-01'*, *end='2022-01-05'*)
Convert start and end date to scihub's search url time period attribute

Parameters

- **start** (*string, YYYY-MM-DD date format*) – start date as a YYYY-MM-DD formatted string, defaults to '2014-10-01'
- **end** (*string, YYYY-MM-DD date format*) – end date as a YYYY-MM-DD formatted string, defaults to now

Returns Copernicus' scihub compliant TOI query string

Return type str

ost.helpers.scihub.**next_page**(*dom*)
Gets link for next page for results from apihub/scihub

Parameters **dom** (*xml.dom object*) – object coming back from a Copernicus' scihub search request

Returns Link ot the next page or None if we reached the end.

Return type str

ost.helpers.scihub.**s1_download**(*uuid*, *filename*, *uname*, *pword*, *base_url='https://apihub.copernicus.eu/apihub'*)
Single scene download function for Copernicus scihub/apihub

Parameters

- **uuid** – product's uuid
- **filename** – local path for the download
- **uname** – username of Copernicus' scihub
- **pword** – password of Copernicus' scihub

- **base_url** –

Returns

ost.helpers.scihub.**s1_download_parallel**(*argument_list*)
Helper function for parallel download from scihub

ost.helpers.settings

Functions

check_ard_parameters
check_value
exception_handler
generate_access_file
get_gpt
set_log_level
setup_logfile

ost.helpers.settings.**check_ard_parameters**(*ard_parameters*)

ost.helpers.settings.**check_value**(*key*, *value*, *expected_type*, *choices=None*)

ost.helpers.settings.**exception_handler**(*exception_type*, *exception*, *traceback*)

ost.helpers.settings.**generate_access_file**()

ost.helpers.settings.**get_gpt**()

ost.helpers.settings.**set_log_level**(*log_level=20*)

ost.helpers.settings.**setup_logfile**(*logfile*)

ost.helpers.srtm

Functions for searching and downloading from Copernicus scihub.

Functions

download_srtm
download_srtm_tile

ost.helpers.srtm.**download_srtm**(*aoi*)

ost.helpers.srtm.**download_srtm_tile**(*url*)

ost.helpers.vector

Functions

aoi_to_wkt	Helper function to transform various AOI formats into WKT
buffer_shape	
difference	
epsg_to_wkt_projection	**param epsg_code**
exterior	Creates an exterior vector of an input vector
gdf_to_json_geometry	Function to parse features from GeoDataFrame in such a manner that rasterio wants them
geodesic_point_buffer	**param lat**
get_epsg	**param prjfile**
get_proj4	Get the proj4 string from a projection file of a shapefile
latlon_to_shp	**param lon**
latlon_to_wkt	A helper function to create a WKT representation of Lat/Lon pair
plot_inventory	
reproject_geometry	Reproject a wkt geometry based on EPSG code
set_subset	
shp_to_wkt	A helper function to translate a shapefile into WKT
wkt_manipulations	**param wkt**
wkt_to_gdf	**param wkt**

ost.helpers.vector.**aoi_to_wkt**(*aoi*)

Helper function to transform various AOI formats into WKT

This function is used to import an AOI definition into an OST project. The AOIs definition can be from diffrent sources, i.e. an ISO3 country code (that calls GeoPandas low-resolution country boundaries), a WKT string,

Parameters **aoi** (*str/Path*) – AOI , which can be an ISO3 country code, a WKT String or a path to a shapefile, a GeoPackage or a GeoJSON file

Returns AOI as WKT string

Return type WKT string

ost.helpers.vector.**buffer_shape**(*infile*, *outfile*, *buffer=None*)

ost.helpers.vector.**difference**(*infile1*, *infile2*, *outfile*)

ost.helpers.vector.**epsg_to_wkt_projection**(*epsg_code*)

Parameters **epsg_code** –

Returns

ost.helpers.vector.**exterior**(*infile*, *outfile*, *buffer=None*)

Creates an exterior vector of an input vector

Parameters

- **infile** –
- **outfile** –
- **buffer** –

Returns

ost.helpers.vector.**gdf_to_json_geometry**(*gdf*)

Function to parse features from GeoDataFrame in such a manner that rasterio wants them

ost.helpers.vector.**geodesic_point_buffer**(*lon*, *lat*, *meters*, *envelope=False*)

Parameters

- **lat** –
- **lon** –
- **meters** –
- **envelope** –

Returns

ost.helpers.vector.**get_epsg**(*prjfile*)

Parameters **prjfile** –

Returns

ost.helpers.vector.**get_proj4**(*prjfile*)

Get the proj4 string from a projection file of a shapefile

Parameters **prjfile** –

Returns

ost.helpers.vector.**latlon_to_shp**(*lon*, *lat*, *shapefile*)

Parameters

- **lon** –
- **lat** –
- **shapefile** –

Returns

ost.helpers.vector.**plot_inventory**(*aoi*, *inventory_df*, *transparency=0.05*, *annotate=False*)

ost.helpers.vector.**reproject_geometry**(*geom*, *inproj4*, *out_epsg*)
Reproject a wkt geometry based on EPSG code

Parameters

- **geom** – an ogr geom object
- **inproj4** – a proj4 string
- **out_epsg** – the EPSG code to which the geometry should transformed

Returns the transformed geometry (ogr-geometry object)

ost.helpers.vector.**set_subset**(*aoi*, *inventory_df*)

ost.helpers.vector.**shp_to_wkt**(*shapefile*, *buffer=None*, *convex=False*, *envelope=False*)
A helper function to translate a shapefile into WKT

Parameters

- **shapefile** –
- **buffer** –
- **convex** –
- **envelope** –

Returns

ost.helpers.vector.**wkt_manipulations**(*wkt*, *buffer=None*, *convex=False*, *envelope=False*)

Parameters

- **wkt** –
- **buffer** –
- **convex** –
- **envelope** –

Returns

ost.helpers.vector.**wkt_to_gdf**(*wkt*)

Parameters **wkt** –

Returns

5.2.3 ost.s1

Modules

ost.s1.burst_batch	Batch processing routines for Sentinel-1 bursts
ost.s1.burst_inventory	
ost.s1.burst_to_ard	

continues on next page

Table 28 – continued from previous page

ost.s1.download	Module for batch download of Sentinel-1 based on OST inventory
ost.s1.grd_batch	Batch processing for GRD products
ost.s1.grd_to_ard	
ost.s1.grd_wrappers	
ost.s1.refine_inventory	
ost.s1.s1scene	Class for handling a single Sentinel-1 product
ost.s1.search	Based on a set of search parameters the script will create a query on www.scihub.copernicus.eu and return the results either as shapefile, sqlite, or write to a PostGreSQL database.
ost.s1.slc_wrappers	
ost.s1.ts	

ost.s1.burst_batch

Batch processing routines for Sentinel-1 bursts

This module handles all the batch processing routines involved in the full workflow from raw Sentinel-1 SLC imagery to large-scale time-series and timescan mosaics.

Functions

ards_to_timeseries	
bursts_to_ards	Batch processing from single bursts to ARD format
mosaic_timescan	**param burst_inventory**
mosaic_timescan_old	
mosaic_timeseries	
timeseries_to_timescan	Function to create a timescan out of a OST timeseries.

ost.s1.burst_batch.**ards_to_timeseries**(*burst_gdf*, *config_file*)

ost.s1.burst_batch.**bursts_to_ards**(*burst_gdf*, *config_file*)

Batch processing from single bursts to ARD format

This function handles the burst processing based on a OST burst inventory file and an OST config file that contains all necessary information about the project (e.g. project directory) and processing steps applied for the ARD generation based on the JSON ARD-type templates.

Parameters

- **burst_gdf** (*GeoDataFrame*) – an OST burst inventory

- **config_file** – (str/Path) path to the project config file
- **executor_type** – executer type for parallel processing with godale, defaults to multiprocessing
- **max_workers** – number of parallel burst processing jobs to start

Returns

ost.s1.burst_batch.**mosaic_timescan**(*burst_inventory*, *config_file*)

Parameters

- **burst_inventory** –
- **config_file** –

Returns

ost.s1.burst_batch.**mosaic_timescan_old**(*config_file*)

ost.s1.burst_batch.**mosaic_timeseries**(*burst_inventory*, *config_file*)

ost.s1.burst_batch.**timeseries_to_timescan**(*burst_gdf*, *config_file*)
Function to create a timescan out of a OST timeseries.

ost.s1.burst_inventory

Functions

burst_extract	Extract all bursts from the Sentinel-1 annotation files
burst_inventory	Creates a Burst GeoDataFrame from an OST inventory file Args: Returns:
prepare_burst_inventory	
refine_burst_inventory	Creates a Burst GeoDataFrame from an OST inventory file Args: Returns:

ost.s1.burst_inventory.**burst_extract**(*scene_id*, *track*, *acq_date*, *et_root*)
Extract all bursts from the Sentinel-1 annotation files

Parameters

- **scene_id** –
- **track** –
- **acq_date** –
- **et_root** –

Returns

ost.s1.burst_inventory.**burst_inventory**(*inventory_df*, *outfile*, *download_dir='/home/docs'*, *data_mount=None*, *uname=None*, *pword=None*)
Creates a Burst GeoDataFrame from an OST inventory file Args: Returns:

ost.s1.burst_inventory.**prepare_burst_inventory**(*burst_gdf*, *config_file*)

ost.s1.burst_inventory.**refine_burst_inventory**(*aoi*, *burst_gdf*, *outfile*, *coverages=None*)
Creates a Burst GeoDataFrame from an OST inventory file Args: Returns:

ost.s1.burst_to_ard

Functions

burst_to_ard

create_backscatter_layers	Pipeline for backscatter processing
create_coherence_layers	Pipeline for Dual-polarimetric decomposition
create_polarimetric_layers	Pipeline for Dual-polarimetric decomposition

ost.s1.burst_to_ard.**burst_to_ard**(*burst*, *config_file*)

ost.s1.burst_to_ard.**create_backscatter_layers**(*import_file*, *out_dir*, *burst_prefix*, *config_dict*)

Pipeline for backscatter processing

Parameters

- **import_file** –
- **out_dir** –
- **burst_prefix** –
- **config_dict** –

Returns

ost.s1.burst_to_ard.**create_coherence_layers**(*master_import*, *slave_import*, *out_dir*, *master_prefix*, *config_dict*)

Pipeline for Dual-polarimetric decomposition

Parameters

- **master_import** –
- **slave_import** –
- **out_dir** –
- **master_prefix** –
- **config_dict** –

Returns

ost.s1.burst_to_ard.**create_polarimetric_layers**(*import_file*, *out_dir*, *burst_prefix*, *config_dict*)

Pipeline for Dual-polarimetric decomposition

Parameters

- **import_file** –
- **out_dir** –
- **burst_prefix** –
- **config_dict** –

Returns

ost.s1.download

Module for batch download of Sentinel-1 based on OST inventory

This module handles the download of Sentinel-1, offering download capabilities from different servers such as Copernicus Scihub, Alaska Satellite Facility's vertex as well as PEPS from CNES.

Functions

download_sentinel1	Function to download Sentinel-1 with choice of data repository
restore_download_dir	Create the OST download directory structure from downloaded files

ost.s1.download.**download_sentinel1**(*inventory_df*, *download_dir*, *mirror=None*, *concurrent=2*, *uname=None*, *pword=None*)

Function to download Sentinel-1 with choice of data repository

Parameters

- **inventory_df** (*GeoDataFrame*) – OST inventory dataframe
- **download_dir** (*Path*) – high-level download directory
- **mirror** (*int*) – number of data repository

  1 - Scihub

  2 - ASF

  3 - PEPS

  4 - ONDA
- **concurrent** (*int*) – number of parallel downloads
- **uname** (*str*) – username for respective data repository
- **pword** (*str*) – password for respective data repository

ost.s1.download.**restore_download_dir**(*input_directory*, *download_dir*)

Create the OST download directory structure from downloaded files

In case data is already downloaded to a single folder, this function can be used to create a OST compliant structure of the download directory.

Parameters

- **input_directory** (*str/Path*) – the directory, where the downloaded files are located
- **download_dir** (*str/Path*) – the high-level directory compliant with OST

ost.s1.grd_batch

Batch processing for GRD products

Functions

ards_to_timeseries	
create_processed_df	
grd_to_ard_batch	
mosaic_timescan	
mosaic_timeseries	
timeseries_to_timescan	

ost.s1.grd_batch.**ards_to_timeseries**(*inventory_df*, *config_file*)

ost.s1.grd_batch.**create_processed_df**(*inventory_df*, *list_of_scenes*, *outfile*, *out_ls*, *error*)

ost.s1.grd_batch.**grd_to_ard_batch**(*inventory_df*, *config_file*)

ost.s1.grd_batch.**mosaic_timescan**(*config_file*)

ost.s1.grd_batch.**mosaic_timeseries**(*inventory_df*, *config_file*)

ost.s1.grd_batch.**timeseries_to_timescan**(*inventory_df*, *config_file*)

ost.s1.grd_to_ard

Functions

ard_to_rgb	
grd_to_ard	Main function for the grd to ard generation

ost.s1.grd_to_ard.**ard_to_rgb**(*infile*, *outfile*, *driver='GTiff'*, *to_db=True*, *shrink_factor=1*)

ost.s1.grd_to_ard.**grd_to_ard**(*filelist*, *config_file*)

Main function for the grd to ard generation

This function represents the full workflow for the generation of an Analysis-Ready-Data product. The standard parameters reflect the CEOS ARD defintion for Sentinel-1 backcsatter products.

By changing the parameters, taking care of all parameters that can be given. The function can handle multiple inputs of the same acquisition, given that there are consecutive data takes.

Parameters

- **filelist** – must be a list with one or more absolute paths to GRD scene(s)
- **config_file** –

Returns

ost.s1.grd_wrappers

Functions

calibration	**param infile**
grd_frame_import	A wrapper of SNAP import of a single Sentinel-1 GRD product
grd_remove_border	OST function to remove GRD border noise from Sentinel-1 data
grd_subset_georegion	Wrapper function around SNAP's subset routine
multi_look	**param infile**
slice_assembly	Wrapper function around SNAP's slice assembly routine

ost.s1.grd_wrappers.**calibration**(*infile*, *outfile*, *logfile*, *config_dict*)

Parameters

- **infile** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.s1.grd_wrappers.**grd_frame_import**(*infile*, *outfile*, *logfile*, *config_dict*)

A wrapper of SNAP import of a single Sentinel-1 GRD product

This function takes an original Sentinel-1 scene (either zip or SAFE format), updates the orbit information (does not fail if not available), removes the thermal noise and stores it as a SNAP compatible BEAM-Dimap format.

Parameters

- **infile** (*str/Path*) – Sentinel-1 GRD product in zip or SAFE format
- **outfile** (*str/Path*) –
- **logfile** –
- **config_dict** (*dict*) – an OST configuration dictionary

Returns

ost.s1.grd_wrappers.**grd_remove_border**(*infile*)

OST function to remove GRD border noise from Sentinel-1 data

This is a custom routine to remove GRD border noise from Sentinel-1 GRD products. It works on the original intensity images.

NOTE: For the common dimap format, the infile needs to be the ENVI style file inside the *data folder.

The routine checks the outer 3000 columns for its mean value. If the mean value is below 100, all values will be set to 0, otherwise the routine will continue fpr another 150 columns setting the value to 0. All further columns towards the inner image are considered valid.

Parameters **infile** –

Returns

ost.s1.grd_wrappers.**grd_subset_georegion**(*infile*, *outfile*, *logfile*, *config_dict*)
Wrapper function around SNAP's subset routine

This function takes an OST imported/slice assembled frame and subsets it according to the coordinates given in the region

Parameters

- **infile** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.s1.grd_wrappers.**multi_look**(*infile*, *outfile*, *logfile*, *config_dict*)

Parameters

- **infile** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.s1.grd_wrappers.**slice_assembly**(*filelist*, *outfile*, *logfile*, *config_dict*)
Wrapper function around SNAP's slice assembly routine

Parameters

- **filelist** (*str*) – a string of a space separated list of OST imported Sentinel-1 GRD product frames to be assembled
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.s1.refine_inventory

Functions

search_refinement	**param aoi**

ost.s1.refine_inventory.**search_refinement**(*aoi*, *inventory_df*, *inventory_dir*, *exclude_marginal=True*, *full_aoi_crossing=True*, *mosaic_refine=True*, *area_reduce=0.05*, *complete_coverage=True*)

Parameters

- **aoi** –
- **inventory_df** –
- **inventory_dir** –
- **exclude_marginal** –
- **full_aoi_crossing** –
- **mosaic_refine** –
- **area_reduce** –
- **complete_coverage** –

Returns

'"A function to refine the Sentinel-1 search by certain criteria Args:

aoi (WKT str): inventory_df (GeoDataFrame): inventory_dir (str or path):

Returns: refined inventory (dictionary): coverages (dictionary):

ost.s1.s1scene

Class for handling a single Sentinel-1 product

This class, initialized by a valid Sentinel-1 scene identifier, extracts basic metadata information from the scene ID itself, as well as more detailed and allows for retrieving OST relevant paths.

For GRD products it is possible to pre-process the respective scene based on a ARD product type.

Classes

Sentinel1Scene	

ost.s1.s1scene.Sentinel1Scene

class ost.s1.s1scene.**Sentinel1Scene**(*scene_id*, *ard_type='OST_GTC'*, *log_level=20*)

Attributes

—

Methods

info	
info_dict	
download	
download_path	
get_path	
scihub_uuid	
scihub_url	
scihub_md5	
scihub_online_status	
scihub_trigger_production	
scihub_annotation_get	**param uname**
zip_annotation_get	**param download_dir**
safe_annotation_get	
ondadias_uuid	
asf_url	Constructor for ASF download URL
peps_uuid	Retrieval of the PEPS UUID from the Peps server
peps_online_status	Check if product is online at CNES' Peps server.
get_ard_parameters	
update_ard_parameters	**param ard_type**

continues on next page

Table 39 – continued from previous page

set_external_dem	**param dem_file**
create_ard	**param infile**
create_rgb	
create_rgb_thumbnail	
visualise_rgb	

Sentinel1Scene.**info**()

Sentinel1Scene.**info_dict**()

Sentinel1Scene.**download**(*download_dir*, *mirror=None*)

Sentinel1Scene.**download_path**(*download_dir*, *mkdir=False*)

Sentinel1Scene.**get_path**(*download_dir=None*, *data_mount=None*)

Sentinel1Scene.**scihub_uuid**(*opener*)

Sentinel1Scene.**scihub_url**(*opener*)

Sentinel1Scene.**scihub_md5**(*opener*)

Sentinel1Scene.**scihub_online_status**(*opener*)

Sentinel1Scene.**scihub_trigger_production**(*opener*)

Sentinel1Scene.**scihub_annotation_get**(*uname=None*, *pword=None*)

Parameters

- **uname** –
- **pword** –

Returns

Sentinel1Scene.**zip_annotation_get**(*download_dir*, *data_mount=None*)

Parameters

- **download_dir** –
- **data_mount** –

Returns

Sentinel1Scene.**safe_annotation_get**(*download_dir*, *data_mount=None*)

Sentinel1Scene.**ondadias_uuid**(*opener*)

Sentinel1Scene.**asf_url**()

Constructor for ASF download URL

Returns string of the ASF download url

Return type str

Sentinel1Scene.**peps_uuid**(*uname*, *pword*)
Retrieval of the PEPS UUID from the Peps server

Parameters

- **uname** – username for CNES' PEPS server
- **pword** – password for CNES' PEPS server

Returns

Sentinel1Scene.**peps_online_status**(*uname*, *pword*)
Check if product is online at CNES' Peps server.

Parameters

- **uname** – CNES' Peps username
- **pword** – CNES' Peps password

Returns

Sentinel1Scene.**get_ard_parameters**(*ard_type*)

Sentinel1Scene.**update_ard_parameters**(*ard_type=None*)

Parameters **ard_type** –

Returns

Sentinel1Scene.**set_external_dem**(*dem_file*, *ellipsoid_correction=True*)

Parameters

- **dem_file** –
- **ellipsoid_correction** –

Returns

Sentinel1Scene.**create_ard**(*infile*, *out_dir*, *subset=None*, *overwrite=False*)

Parameters

- **infile** –
- **out_dir** –
- **subset** –
- **overwrite** –

Returns

Sentinel1Scene.**create_rgb**(*outfile*, *driver='GTiff'*)

Sentinel1Scene.**create_rgb_thumbnail**(*outfile*, *shrink_factor=25*)

Sentinel1Scene.**visualise_rgb**(*shrink_factor=25*)

ost.s1.search

Based on a set of search parameters the script will create a query on www.scihub.copernicus.eu and return the results either as shapefile, sqlite, or write to a PostGreSQL database.

Functions:

gdfInv2Pg: writes the search result into a PostGreSQL/PostGIS Database

gdfInv2Sqlite: (tba) writes the search result into a SqLite/SpatiaLite Database

Main function

scihubSearch: handles the whole search process, i.e. login, query creation, search and write to desired output format

Contributors

Andreas Vollrath, ESA phi-lab

August 2018: Original implementation

Usage

python3 search.py -a /path/to/aoi-shapefile.shp -b 2018-01-01 -e 2018-31-12

-t GRD -m VV -b IW -o /path/to/search.shp

-a	defines ISO3 country code or path to an ESRI shapefile
-s	defines the satellite platform (Sentinel-1, Sentinel-2, etc.)
-b	defines start date*
-e	defines end date for search*
-t	defines the product type (i.e. RAW,SLC or GRD)*
-m	defines the polarisation mode (VV, VH, HH or HV)*
-b	defines the beammode (IW,EW or SM)*
-o	defines output that can be a shapefile (ending with .shp), a SQLite DB (ending with .sqlite) or a PostGreSQL DB (no suffix)
-u	the scihub username*
-p	the scihub secret password*

- optional, i.e will look for all available products as well as ask for username and password during script execution

Functions

check_availability	This function checks if the data is already downloaded or
scihub_catalogue	This is the main search function on scihub

ost.s1.search.**check_availability**(*inventory_gdf*, *download_dir*, *data_mount*)

This function checks if the data is already downloaded or available through a mount point on DIAS cloud

Parameters

- **inventory_gdf** –
- **download_dir** –
- **data_mount** –

Returns

ost.s1.search.**scihub_catalogue**(*query_string*, *output*, *append=False*, *uname=None*, *pword=None*, *base_url='https://apihub.copernicus.eu/apihub'*)

This is the main search function on scihub

Parameters

- **query_string** –
- **output** –
- **append** –
- **uname** –
- **pword** –

Returns

ost.s1.slc_wrappers

Functions

burst_import	A wrapper of SNAP import of a single Sentinel-1 SLC burst
calibration	A wrapper around SNAP's radiometric calibration
coherence	A wrapper around SNAP's coherence routine
coreg	A wrapper around SNAP's back-geocoding co-registration routine
coreg2	A wrapper around SNAP's back-geocoding co-registration routine
ha_alpha	A wrapper of SNAP H-A-alpha polarimetric decomposition

ost.s1.slc_wrappers.**burst_import**(*infile*, *outfile*, *logfile*, *swath*, *burst*, *config_dict*)

A wrapper of SNAP import of a single Sentinel-1 SLC burst

This function takes an original Sentinel-1 scene (either zip or SAFE format), updates the orbit information (does not fail if not available), and extracts a single burst based on the given input parameters.

Parameters

- **infile** –
- **outfile** –
- **logfile** –
- **swath** –
- **burst** –
- **config_dict** –

Returns

ost.s1.slc_wrappers.**calibration**(*infile*, *outfile*, *logfile*, *config_dict*)
A wrapper around SNAP's radiometric calibration

This function takes OST imported Sentinel-1 product and generates it to calibrated backscatter. 3 different calibration modes are supported.

- Radiometrically terrain corrected Gamma nought (RTC) NOTE: that the routine actually calibrates to bet0 and needs to be used together with _terrain_flattening routine
- ellipsoid based Gamma nought (GTCgamma)
- Sigma nought (GTCsigma).

Parameters

- **infile** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.s1.slc_wrappers.**coherence**(*infile*, *outfile*, *logfile*, *config_dict*)
A wrapper around SNAP's coherence routine

This function takes a co-registered stack of 2 Sentinel-1 SLC products and calculates the coherence.

Parameters

- **infile** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.s1.slc_wrappers.**coreg**(*master*, *slave*, *outfile*, *logfile*, *config_dict*)
A wrapper around SNAP's back-geocoding co-registration routine

This function takes 2 OST imported Sentinel-1 SLC products (master and slave) and co-registers them properly. This routine is sufficient for coherence estimation, but not for InSAR, since the ESD refinement is not applied.

Parameters

- **master** –
- **slave** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.s1.slc_wrappers.**coreg2**(*master*, *slave*, *outfile*, *logfile*, *config_dict*)
A wrapper around SNAP's back-geocoding co-registration routine

This function takes 2 OST imported Sentinel-1 SLC products (master and slave) and co-registers them properly. This routine is sufficient for coherence estimation, but not for InSAR, since the ESD refinement is not applied.

Parameters

- **master** –
- **slave** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.s1.slc_wrappers.**ha_alpha**(*infile*, *outfile*, *logfile*, *config_dict*)
A wrapper of SNAP H-A-alpha polarimetric decomposition

This function takes an OST imported Sentinel-1 scene/burst and calulates the polarimetric decomposition parameters for the H-A-alpha decomposition.

Parameters

- **infile** –
- **outfile** –
- **logfile** –
- **config_dict** –

Returns

ost.s1.ts

Functions

create_datelist	Create a text file of acquisition dates within your time-series
create_ts_animation	

ost.s1.ts.**create_datelist**(*path_to_timeseries*)
Create a text file of acquisition dates within your time-series

Args: path_to_timeseries (str): path to an OST time-series directory

ost.s1.ts.**create_ts_animation**(*ts_dir*, *temp_dir*, *outfile*, *shrink_factor*)

5.2.4 ost.Project

Classes

Generic	
Sentinel1	A Sentinel-1 specific subclass of the Generic OST class This subclass creates a Sentinel-1 specific
Sentinel1Batch	A Sentinel-1 specific subclass of the Generic OST class This subclass creates a Sentinel-1 specific

ost.Project.Generic

class ost.Project.**Generic**(*project_dir*, *aoi*, *start='1978-06-28'*, *end='2022-01-05'*, *data_mount=None*, *log_level=20*)

Attributes

—

ost.Project.Sentinel1

class ost.Project.**Sentinel1**(*project_dir*, *aoi*, *start='2014-10-01'*, *end='2022-01-05'*, *data_mount=None*, *product_type='*'*, *beam_mode='*'*, *polarisation='*'*, *log_level=20*)

A Sentinel-1 specific subclass of the Generic OST class This subclass creates a Sentinel-1 specific

Attributes

—

Methods

search	High Level search function
read_inventory	Read the Sentinel-1 data inventory from a OST inventory shapefile :param
download_size	Function to get the total size of all products when extracted in GB
refine_inventory	
download	
create_burst_inventory	

continues on next page

Table 46 – continued from previous page

read_burst_inventory	**param burst_file** a GeoPackage file created by OST holding a burst
plot_inventory	

Sentinel1.**search**(*outfile='full.inventory.gpkg'*, *append=False*, *base_url='https://apihub.copernicus.eu/apihub'*)
High Level search function

Parameters

- **outfile** –
- **append** –
- **base_url** –

Returns

Sentinel1.**read_inventory**()
Read the Sentinel-1 data inventory from a OST invetory shapefile :param

Sentinel1.**download_size**(*inventory_df=None*)
Function to get the total size of all products when extracted in GB

Parameters **inventory_df** –

Returns

Sentinel1.**refine_inventory**(*exclude_marginal=True*, *full_aoi_crossing=True*, *mosaic_refine=True*, *area_reduce=0.05*, *complete_coverage=True*)

Sentinel1.**download**(*inventory_df*, *mirror=None*, *concurrent=2*, *uname=None*, *pword=None*)

Sentinel1.**create_burst_inventory**(*inventory_df=None*, *refine=True*, *outfile=None*, *uname=None*, *pword=None*)

Sentinel1.**read_burst_inventory**(*burst_file=None*)

Parameters **burst_file** – a GeoPackage file created by OST holding a burst inventory

Returns geodataframe

Sentinel1.**plot_inventory**(*inventory_df=None*, *transparency=0.05*, *annotate=False*)

ost.Project.Sentinel1Batch

class ost.Project.**Sentinel1Batch**(*project_dir*, *aoi*, *start='2014-10-01'*, *end='2022-01-05'*, *data_mount=None*, *product_type='SLC'*, *beam_mode='IW'*, *polarisation='*'*, *ard_type='OST-GTC'*, *snap_cpu_parallelism=2*, *max_workers=1*, *log_level=20*)
A Sentinel-1 specific subclass of the Generic OST class This subclass creates a Sentinel-1 specific

Attributes

—

Methods

—

Sentinel1Batch.**get_ard_parameters**(*ard_type*)

Sentinel1Batch.**update_ard_parameters**(*ard_type=None*)

Sentinel1Batch.**set_external_dem**(*dem_file*, *ellipsoid_correction=True*)

Sentinel1Batch.**pre_download_srtm**()

Sentinel1Batch.**bursts_to_ards**(*timeseries=False*, *timescan=False*, *mosaic=False*, *overwrite=False*)
Batch processing function for full burst pre-processing workflow

This function allows for the generation of the

Parameters **timeseries** – if True, Time-series will be generated for

each burst id :type timeseries: bool, optional :param timescan: if True, Timescans will be generated for each burst id type: timescan: bool, optional :param mosaic: if True, Mosaics will be generated from the

Time-Series/Timescans of each burst id

Parameters **overwrite** – (if True, the processing folder will be

emptied :type overwrite: bool, optional :param max_workers: number of parallel burst :type max_workers: int, default=1 processing jobs :return:

static Sentinel1Batch.**create_timeseries_animation**(*timeseries_dir*, *product_list*, *outfile*, *shrink_factor=1*, *resampling_factor=5*, *duration=1*, *add_dates=False*, *prefix=False*)

Sentinel1Batch.**grds_to_ards**(*inventory_df*, *timeseries=False*, *timescan=False*, *mosaic=False*, *overwrite=False*, *max_workers=1*, *executor_type='billiard'*)

5.3 Examples

examples on how to run OST.

Open SAR Toolkit - Tutorial 1, version 1.3, July 2020. Andreas Vollrath, ESA/ESRIN phi-lab

5.3.1 OST Tutorial I

Pre-processing your first Sentinel-1 image with OST

Short description

This notebook introduces you to the *Sentinel1Scene* class of the Open SAR Toolkit and demonstrates how it can be used to download, extract metadata and pre-process a single Sentinel-1 scene.

Requirements

- a PC/Mac with at least 16GB of RAM
- about 4 GB of free disk space (or more if you want to process more scenes)
- **a NASA Earthdata account with signed EULA for use of https://search.asf.alaska.edu (just register directly there)**

NOTE: all cells that have an * after its number can be executed without changing any code.

0* - Install OST and dependencies

NOTE: Applies only if you haven't fully installed OST and its dependencies yet, e.g. on Google Colab, so uncomment the lines in this case.

[ ]:
```
# !apt-get -y install wget
# !wget https://raw.githubusercontent.com/ESA-PhiLab/OST_Notebooks/master/install_ost.sh
# !bash install_ost.sh
```

1* - Import the OST *Sentinel1Scene* class

[ ]:
```
# these imports we need to handle the folders, independent of the OS
from pathlib import Path
from pprint import pprint

# this is the Sentinel1Scene class, that basically handles all the workflow from␣
↪beginning to the end
from ost import Sentinel1Scene

# from ost.helpers.settings import set_log_level
# import logging
# set_log_level(logging.DEBUG)
```

2* - Create a folder for our outputs

By executing this cell, a new folder will be created and the path will be written to the *output_dir* variable

[ ]:
```
# get home folder
home = Path.home()

# create a processing directory
output_dir = home / "OST_Tutorials" / "Tutorial_1"
output_dir.mkdir(parents=True, exist_ok=True)
print(str(output_dir))
```

3* - Choose scene ID and display some metadata

In order to initialize an instance of the *Sentinel1Scene* class, all we need is a valid scene id of a Sentinel-1 product.

[ ]:
```
# create a S1Scene class instance based on the scene identifier of the first ever Dual-
↪Pol Sentinel-1 IW product

# ------------------------------------------------
# Some scenes to choose from

# very first IW (VV/VH) S1 image available over Istanbul/Turkey
# NOTE:only available via ASF data mirror
scene_id = "S1A_IW_GRDH_1SDV_20141003T040550_20141003T040619_002660_002F64_EC04"

# other scenes with different scene types to process (uncomment)
# IW scene (dual-polarised HH/HV) over Norway/Spitzbergen
# scene_id = 'S1B_IW_GRDH_1SDH_20200325T150411_20200325T150436_020850_02789D_2B85'

# IW scene (single-polarised VV) over Ecuadorian Amazon
# scene_id = 'S1A_IW_GRDH_1SSV_20150205T232009_20150205T232034_004494_00583A_1C80'

# EW scene (dual-polarised VV/VH) over Azores (needs a different DEM, see cell of ARD␣
↪parameters below)
# scene_id = 'S1B_EW_GRDM_1SDV_20200303T193150_20200303T193250_020532_026E82_5CE9'

# EW scene (dual-polarised HH/HV) over Greenland
# scene_id = 'S1B_EW_GRDM_1SDH_20200511T205319_20200511T205419_021539_028E4E_697E'

# Stripmap mode S5 scene (dual-polarised VV/VH) over Germany
# scene_id = 'S1B_S5_GRDH_1SDV_20170104T052519_20170104T052548_003694_006587_86AB'
# ------------------------------------------------

# create an S1Scene instance
s1 = Sentinel1Scene(scene_id)

# print summarising infos about the scene
s1.info()
```

4* Download scene

The first step is to download the selected scene. In our case we chose the first regular Sentinel-1 IW product acquired in the dual-polarisation VV/VH acquired the 3rd of October 2014. OST provides download from 3 different mirrors, ESA's scihub, CNES PEPS and Alaska Satellite Facility (NASA Earthdata). Since ESA's official scihub became a rolling archive, and so is PEPS, best is to download from the fantastic **Alaska Satellite Facility** mirror (selection 2), which holds the full Sentinel-1 archive online.

Note I: You can interrupt the download and restart it. The download will continue from where it stopped.

Note II: After the actual download, the file is checked for inconsistency. This assures that the download went fine and we can use it for processing. In addition, OST will magically remember that this file has been successfully downloaded (just run the cell again to see the behaviour).

```
[ ]: s1.download(output_dir)
```

5* - Define our ARD product

ARD stands for Analysis Ready Data and is interpreted differently by different persons. OST provides various pre-defined flavours which can be used instantly.

The following table shows the ARD types and corresponding processing steps applied for GRD data.

The default ARD type is *'OST-GTC'*, referring to a Geometrically Terrain Corrected product which is calibrated to the ellipsoid correceted γ^0 backscatter coefficient at 20m resolution. Other pre-defined ARD types are available, but it is also possible to customise single ARD parameters via a dictionary where all parameters are stored (as demonstrated in the cell below). Note how the resolution and the resampling of the image during terrain correction is changed at the bottom. In this way, actually all relevant parameters for processing are customisable.

```
[ ]: # Default ARD parameter

print(
    "-----------------------------------------------------------------------------------------
↪-----------------------"
)
print(
    "Our ARD parameters dictionary contains 4 keys. For the moment, only single_ARD is␣
↪relevant."
)
print(
    "-----------------------------------------------------------------------------------------
↪-----------------------"
)
pprint(s1.ard_parameters.keys())
print(
    "-----------------------------------------------------------------------------------------
↪-----------------------"
)
print("")

print(
    "-----------------------------------------------------------------------------------------
↪-----------------------"
)
```

(continues on next page)

(continued from previous page)

```
print("Dictionary of our default OST ARD parameters for single scene processing:")
print(
    "----------------------------------------------------------------------------------------------------
↪----------------------"
)
pprint(s1.ard_parameters["single_ARD"])
print(
    "----------------------------------------------------------------------------------------------------
↪----------------------"
)
print("")
```

[ ]:
```
# Template ARD parameters

# we change ARD type
# possible choices are:
# 'OST_GTC', 'OST-RTC', 'CEOS', 'Earth Engine'
s1.update_ard_parameters("Earth-Engine")
print(
    "----------------------------------------------------------------------------------------------------
↪----------------------"
)
print("Dictionary of Earth Engine ARD parameters:")
print(
    "----------------------------------------------------------------------------------------------------
↪----------------------"
)
pprint(s1.ard_parameters["single_ARD"])
print(
    "----------------------------------------------------------------------------------------------------
↪----------------------"
)
```

[ ]:
```
# Customised ARD parameters

# we cusomize the resolution and image resampling
s1.ard_parameters["single_ARD"]["resolution"] = 100  # set output resolution to 100m
s1.ard_parameters["single_ARD"]["remove_speckle"] = False  # apply a speckle filter
s1.ard_parameters["single_ARD"]["dem"][
    "image_resampling"
] = "BILINEAR_INTERPOLATION"  # BICUBIC_INTERPOLATION is default

# s1.ard_parameters['single_ARD']['product_type'] = 'RTC-gamma0'

# uncomment this for the Azores EW scene
# s1.ard_parameters['single_ARD']['dem']['dem_name'] = 'GETASSE30'
print(
    "----------------------------------------------------------------------------------------------------
↪----------------------"
)
print("Dictionary of our customised ARD parameters for the final scene processing:")
```

(continues on next page)

(continued from previous page)

```
print(
    "-----------------------------------------------------------------------------------------
↪-----------------------"
)
pprint(s1.ard_parameters["single_ARD"])
print(
    "-----------------------------------------------------------------------------------------
↪-----------------------"
)
```

6* - Create an ARD product

Our *Sentinel1Scene* class comes with the build-in method *create_ard* to produce a standardised ARD product based on the ARD dictionary above.

To run the command we just have to provide: - the path to the downloaded file. We can use the *get_path* method in conjunction with the download directory provided - a directory where the output files will be written to - a filename prefix (the output format will be the standard SNAP Dimap format, consisting of the dim-file and the data directory) - and a directory for storing temporary files (can not be the same as the output folder)

[ ]:
```
s1.create_ard(infile=s1.get_path(output_dir), out_dir=output_dir, overwrite=True)

print(" The path to our newly created ARD product can be obtained the following way:")
s1.ard_dimap
```

6* - Create a RGB color composite

Sentinel-1 scenes usually consist of two polarisation bands. In order to create a 3 channel RGB composite a ratio between the Co- (VV or HH) and the Cross-polarised (VH or HV) band is added. The *create_rgb* method takes the *ard_dimap* file and converts it to a 3-channel GeoTiff.

[ ]:
```
s1.create_rgb(outfile=output_dir / f"{s1.start_date}.tif")

print(" The path to our newly created RGB product can be obtained the following way:")
s1.ard_rgb
```

7* - Visualise the RGB composite

We can plot the newly created RGB image with the *visualise_rgb* method. A *shrink_factor* can be set, which reduces resolution in favour of memory requirements for plotting.

[ ]:
```
# -----------------------------------------------
# for plotting purposes we use this iPython magic
%matplotlib inline
%pylab inline
pylab.rcParams["figure.figsize"] = (23, 23)
# -----------------------------------------------
s1.visualise_rgb(shrink_factor=2)
```

8* - Create thumbnail image

For some it might be interesting to create a small thumbnail image in Jpeg format. The *create_rgb_thumbnail* method allows for this.

[ ]:
```
# define a filename for our thumbnail image
path_to_thumbnail = output_dir / f"{s1.start_date}.TN.jpg"

# create the thumbnail image
s1.create_rgb_thumbnail(outfile=str(path_to_thumbnail))
```

[ ]:
```
import imageio

img = imageio.imread(path_to_thumbnail)
!ls -sh {path_to_thumbnail}
plt.imshow(img)
```

9* - Play around

You can try out the different images and also check the difference in backscatter for RTC products (uncomment the line of product type in the customisable ARD parameters)

Open SAR Toolkit - Tutorial 2, version 1.1, July 2020. Andreas Vollrath, ESA/ESRIN phi-lab

5.3.2 OST Tutorial II

How to access and download Sentinel-1 data with OST

Short description

This notebook introduces you to OST's main class *Generic*, and its subclass *Sentinel1*. The *Generic* class handles the basic structure of any OST batch processing project, while the *Sentinel1* class provides methods to search, refine and download sets of acquisitions for the EU Copernicus Sentinel-1 mission.

This notebook is of interest for those users who like to only search and download Sentinel-1 data in an efficient way.

- **I:** Get to know the *Generic* main class for setting up a OST Project
- **II:** Get to know the *Sentinel1* subclass, that features functions for data search and access

Requirements

- a PC/Mac
- about 100MB of free disk space
- **a Copernicus Open Data Hub user account, valid for at least 7 days (https://scihub.copernicus.eu)**

NOTE: all cells that have an * after its number can be executed without changing any code.

0* - Install OST and dependencies

NOTE: Applies only if you haven't fully installed OST and its dependencies yet, e.g. on Google Colab, so uncomment the lines in this case.

```
[ ]: # !apt-get -y install wget
     # !wget https://raw.githubusercontent.com/ESA-PhiLab/OST_Notebooks/master/install_ost.sh
     # !bash install_ost.sh
```

I-1* - Import python libraries necessary for processing

```
[ ]: # this imports we need to handle the folders, independent of the OS
     from pathlib import Path
     from pprint import pprint

     # this is the Generic class, that basically handles all the workflow from beginning to␣
     ↪the end
     from ost import Generic
```

I-2 - Data selection parameters

In order to define your project you need to define 3 main attributes.

1 Area of Interest:

The Area of Interest can be defined in different ways:

1. One possibility is to use the low resolution layer of country boundaries from geopandas. To select a specific country you need to specify its ISO3 code. You can find a collection of all ISO3 codes here.
2. Another possibility is to provide a Well-Known Text formatted string, which is the format OST uses internally.
3. A third possibility is to provide a path to a valid vector file supported by OGR (e.g. GeoJSON, GeoPackage, KML, Esri Shapefile). Try to keep that as simple as possible. If your layer contains lots of different entries (e.g. crop fields), create a convex hull beforehand and use this.

2 Time of Interest:

The time of interest is defined by a *start* and *end* date. The date is defined by a string in the format 'YYYY-MM-DD'. If none of the two parameters are defined, both parameters will use default values, which is *2014-10-01* for *start*, and *today* for the end of the TOI.

3 Project directory

Here we set a high-level directory where all of the project-related data (i.e. inventory, download, processed files) will be stored or created.

[ ]:
```
# ----------------------------
# Area of interest
# ----------------------------

# Here we can either point to a shapefile, an ISO3 country code, or a WKT string
aoi = "AUT"  # AUT is the ISO3 country code of Austria

# ----------------------------
# Time of interest
# ----------------------------
# we set only the start date to today - 30 days
start = "2019-06-01"
end = "2019-08-31"

# ----------------------------
# Project folder
# ----------------------------

# get home folder
home = Path.home()

# create a processing directory
project_dir = home / "OST_Tutorials" / "Tutorial_2"

# ------------------------------
# Print out AOI and start date
# ------------------------------
print("AOI: ", aoi)
print("TOI start: ", start)
print("TOI end: ", end)
print("Project Directory: ", project_dir)
```

I-3* - Initialize the *Generic* class

The above defined variables are used to initialize the class with its main attributes.

[ ]:
```
# create an OST Generic class instance
ost_generic = Generic(project_dir=project_dir, aoi=aoi, start=start, end=end)

# Uncomment below to see the list of folders inside the project directory (UNIX only):
print("")
print(
    "We use the linux ls command for listing the directories inside our project folder:"
)
!ls {project_dir}
```

I-4* Customise project parameters

The initialisation of the class creates a config file, where all project attributes are stored. This includes for example the lcoation of the download or the processing folder. Those can be customised as shown below. Also note that independent of the input format of the AOI, it will be stored as Well Known Text string. The possible input formats for AOI defintion will be covered in later tutorials.

```
[ ]: # Default config as created by the class initialisation
print(" Before customisation")
print("------------------------------------------------------------------------")
pprint(ost_generic.config_dict)
print("------------------------------------------------------------------------")

# customisation
ost_generic.config_dict["download_dir"] = "/download"
ost_generic.config_dict["temp_dir"] = "/tmp"

print("")
print(" After customisation (note the change in download_dir and temp_dir)")
print("------------------------------------------------------------------------")
pprint(ost_generic.config_dict)
```

II-1* - The *Sentinel1* class

The *Sentinel1* class, as a subclass of the Generic class, inherts all the attributes and methods from the Generic class, and adds specific new ones for search and download of data.

```
[ ]: # the import of the Sentinel1 class
from ost import Sentinel1
```

II-2* Initialize the *Sentinel1* class

In addition to the AOI, TOI and project directory parameters needed for the initialization of the *Generic* class, three more Sentinel-1 specific attributes can be defined

1. product_type: this can be either RAW, SLC, GRD or OCN (default is '*' for all)
2. the beam mode: this can be either IW, SM, EW or WV (default is '*' for all)
3. polarisation: This can be either VV, VH, HV, HH or a combination, e.g. VV, VH or HH, HV (default is '*' for all)

Have a look at https://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/acquisition-modes for further information on Sentinel-1 acquisition modes and https://sentinel.esa.int/web/sentinel/missions/sentinel-1/observation-scenario for information of the observation scenario globally.

```
[ ]: # initialize the Sentinel1 class
ost_s1 = Sentinel1(
    project_dir=project_dir,
    aoi=aoi,
    start=start,
    end=end,
```

(continues on next page)

(continued from previous page)

```
    product_type="SLC",
    beam_mode="IW",
    polarisation="*",
)
```

II-3* Searching for data

The search method of our *Sentinel1* class instance will trigger a search query on the scihub catalogue and get the results back in 2 ways:

- write it into a shapefile (inside your inventory directory).
- store it as an instance attribute in form of a Geopandas GeoDataFrame that can be called by ost_s1.inventory

You will need a valid scihub account to do this step. In case you do not have a scihub account yet, please go here to register.

IMPORTANT OST, by default, queries the Copernicus Apihub (i.e. a different server than the one you access over your web browser), for which user credentials will be transfered only after a week of registration to the standard open scihub (more info here). In case this is an issue, use the commented line with the specified base_url and comment out the standard search command.

So you may need to wait a couple of days after first registration before it works.

[ ]:
```
# -----------------------------------------------------
# for plotting purposes we use this iPython magic
%matplotlib inline
%pylab inline
pylab.rcParams["figure.figsize"] = (13, 13)
# -----------------------------------------------------

# search command
ost_s1.search()

# uncomment in case you have issues with the registration procedure
# ost_s1.search(base_url='https://scihub.copernicus.eu/dhus')

# we plot the full Inventory on a map
ost_s1.plot_inventory(transparency=0.1)
```

II-4* The inventory attribute

The results of the search are stored in the *inventory* attribute of the class instance *ost_s1*. This is actually a Geopandas Dataframe that stores all the available metadata from the scihub catalogue. Therefore all (geo)pandas functionality can be applied for filtering, plotting and selection.

[ ]:
```
print(
    "-------------------------------------------------------------------------------
↪-----------------------"
)
print(
    " INFO: We found a total of {} products for our project definition".format(
```

(continues on next page)

(continued from previous page)

```
        len(ost_s1.inventory)
    )
)
print(
    "-----------------------------------------------------------------------------------------
↪-----------------------"
)
print("")
# combine OST class attribute with pandas head command to print out the first 5 rows of␣
↪the
print(
    "-----------------------------------------------------------------------------------------
↪-----------------------"
)
print("The columns of our inventory:")
print("")
print(ost_s1.inventory.columns)
print(
    "-----------------------------------------------------------------------------------------
↪-----------------------"
)

print("")
print(
    "-----------------------------------------------------------------------------------------
↪-----------------------"
)
print(" The last 5 rows of our inventory:")
print(ost_s1.inventory.tail(5))
```

II-5* Search Refinement

The results returned by the search algorithm on Copernicus scihub might not be 100% appropriate to what we are looking for. In this step we refine the results adressing possible issues and reduce later processing needs.

A first step **splits the data** by **orbit direction** (i.e. ascending and descending) and **polarization mode** (i.e. VV, VV/VH, HH, HH/HV). For each combination the routine then checks the coverage for the resulting combinations (e.g. descending VV/VH polarization). If one combination results in a non-full overlap to the AOI, all further steps are disregarded. In case a full coverage is possbile further refinement steps are taken:

1. Some of the acquisition frames might have been processed and/or stored **more than once** in the ESA ground segment. Therefore they appear twice, with the scene identifier that only changes for the last 4 digits. It is necessary to identify those scenes in order to avoid redundancy. We therefore take the ones with the latest ingestion date to assure the use of the latest processor.
2. Some of the scenes returned by the search query are actually **not overlapping the AOI**. This is because the search algorithm will actually check for data within a square defined by the outer bounds of the AOI geometry and not the AOI itself. The refinement only takes those frames overlapping with the AOI in order to reduce unnecessary processing later on.
3. In the case of **ascending tracks that cross the equator**, the **orbit number** of the frames will **increase by 1** even though they are practically from the same acquisition. During processing the frames need to be merged and the relative orbit numbers (i.e. tracks) should be the same. The metadata in the inventory is therefore updated in

order to normalize the relative orbit number for the project.

4. (optional) The tracks of Sentinel-1 overlap to a certain degree. The data inventory might return tracks that only **marginally cross the AOI**, but there AOI overlap is already covered by the adjacent track. Thus, if tracks do not contribute to the overall overlap of the AOI, they are disregarded.
5. (optional) Some acquisitions might **not cross the entire AOI**. For the subsequent time-series/timescan processing this becomes problematic, since the generation of the time-series will only consider the overlapping region for all acquisitions per track.
6. A similar issue appears when one track **crosses the AOI twice**. In other words some of the frames in the middle of the track are not overlapping the AOI and are already disregarded by step 2. The assembling of the non-subsequent frames during processing would result in a failure. The metadata in the inventory is consequently updated, where the first part of the relative orbit number will be renamed to XXX.1, the second part to XXX.2 and so on. During processing those acquistions will be handled as 2 different tracks, and only merged during the final mosaicking.
7. (optional) A last step is needed to assure that for one mosaic in time that consists of different tracks, is only covered once by each track.

```
[ ]: ost_s1.refine_inventory()
```

II-6* - Selecting the right data

The results of the refinement are stored in a new attribute called **refined_inventory_dict**. This is a python dictionary with the mosaic keys as dictionary keys, whereas the respective items are the refined Geodataframes.

```
[ ]: pylab.rcParams["figure.figsize"] = (19, 19)

key = "ASCENDING_VVVH"
ost_s1.refined_inventory_dict[key]
ost_s1.plot_inventory(ost_s1.refined_inventory_dict[key], 0.1)
```

II-7* Downloading the data

Now that we have a refined selection of the scenes we want to download, we have different data mirrors as options. By executing the following cell, OST will ask you from which data portal you want to download.

ESA's Scihub catalogue

The main entry point is the offcial scihub catalogue from ESA. It is however limited to 2 concurrent donwloads at the same time. Also note that it is a rolling archive, so for historical data, a special procedure has to applied to download this data (see Tips and Tricks notebook).

Alternative I - Alaska Satellite Facility:

A good alternative is the download mirror from the Alaska Satellite Facility, which provides the full archive of Sentinel-1 data. In order to get registered, go on their data portal and register. If you already have a NASA Earthdata account, make sure you signed the specific EULA needed to access the Copernicus data. A good practice is to try a download directly from the vertex data protal, to assure everything works.

Alternative II - PEPS server from CNES:

Another good alternative is the Peps server from the French Space Agency CNES. While it is also a rolling archive, copies of historic data are stored on tape and can be easily transferred to the online available storage. OST takes care of that automatically. You can register for an account here

Alternative III - ONDA DIAS by Serco:

Another good alternative is the free data access portal from the ONDA DIAS. This is especially well suited for SLC data for which it holds the full archive. GRD data is accessible by a rolling archive. You can register for an account here.

NOTE While for scihub there is a limit of 2 concurrent downloads, ASF, PEPS and ONDA do not have such strict limits. For ASF the limit is 10, and we can set this with the keyword concurrent.

```
[ ]: ost_s1.download(ost_s1.refined_inventory_dict[key], concurrent=10)
```

Open SAR Toolkit - Tutorial 3, version 1.2, July 2020. Andreas Vollrath, ESA/ESRIN phi-lab

5.3.3 OST Tutorial III

Process the latest Sentinel-1 GRD product for a given point

Short description

This notebook demonstrates the interaction between the *Sentinel1* class for data inventory and download, and the *Sentinel1Scene* class, together, for the generation of the latest Sentinel-1 product over a given point coordinate.

Requirements

- a PC/Mac with at least 16GB of RAM
- about 4GB of free disk space

- **a Copernicus Open Data Hub user account, ideally valid for at least 7 days (https://scihub.copernicus.eu)**

NOTE: all cells that have an * after its number can be executed without changing any code.

0* - Install OST and dependencies

NOTE: Applies only if you haven't fully installed OST and its dependencies yet, e.g. on Google Colab, so uncomment the lines in this case.

```
[ ]: # !apt-get -y install wget
     # !wget https://raw.githubusercontent.com/ESA-PhiLab/OST_Notebooks/master/install_ost.sh
     # !bash install_ost.sh
```

1* - Import python libraries necessary for processing

```
[ ]: # this is for the path handling and especially useful if you are on Windows
     from pathlib import Path
     from pprint import pprint

     # we will need this for our time of period definition
     from datetime import datetime, timedelta

     # this is the s1Project class, that basically handles all the workflow from beginning to␣
     ↪the end
     from ost import Sentinel1, Sentinel1Scene
```

2 - Data selection parameters

NOTE: In case you want to process a different area, all you want to change is the lat and lon values in line 6

As already covered in OST Tutorial 2, we need a minimum of 3 basic parameters to initialise the *Sentinel1* class.

1 Area of Interest:

In this case we only search for a *specific spot on earth, i.e. Rome, Italy*, that is defined by the *Latitude* and *Longitude*. We then create a Well-Known-Text formatted string.

2 Time of Interest:

In this example, the datetime class is used to set the start date to 30 days before today to assure we get any scene within our time of interest.

3 Project directory

Set this to anything you like if not happy with the default one.

```
[ ]: # ----------------------------
     # Area of interest
     # ----------------------------
```

(continues on next page)

(continued from previous page)

```
# Here we can either point to a shapefile or as well use
lat, lon = 41.8919, 12.5113
aoi = "POINT ({} {})".format(lon, lat)

# ----------------------------
# Time of interest
# ----------------------------
# we set only the start date to today - 30 days
start = (datetime.today() - timedelta(days=30)).strftime("%Y-%m-%d")

# ----------------------------
# Processing directory
# ----------------------------
# get home folder
home = Path.home()
# create a processing directory within the home folder
project_dir = home / "OST_Tutorials" / "Tutorial_3"

# ------------------------------
# Print out AOI and start date
# ------------------------------
print(
    "AOI: " + aoi,
)
print("TOI start: " + start)
print("Our project directory is located at: " + str(project_dir))
```

3* - Initialize the Sentinel1 project class

After initialisation of our class, where we explicitley add the GRD product type argument, we do a rough search over our AOI for the last 30 days. We print the first 5 entries and plot all images for visualization by using the *Sentinel1* class attribute *inventory* and method *plot_inventory*.

[ ]:
```
# -------------------------------------------------
# for plotting purposes we use this iPython magic
%matplotlib inline
%pylab inline
pylab.rcParams["figure.figsize"] = (19, 19)
# -------------------------------------------------

# create s1Project class instance
s1_project = Sentinel1(
    project_dir=project_dir, aoi=aoi, start=start, product_type="GRD"
)

# search command
s1_project.search()
# uncomment in case you have issues with the registration procedure
# ost_s1.search(base_url='https://scihub.copernicus.eu/dhus')
print("We found {} products.".format(len(s1_project.inventory.identifier.unique())))
```

(continues on next page)

(continued from previous page)

```
# combine OST class attribute with pandas head command to print out the first 5 rows of␣
↪the
print(s1_project.inventory.head(5))

# we plot the full Inventory on a map
s1_project.plot_inventory(transparency=0.1)
```

4* - Select the latest scene found during the search

Here we use some python-panda syntax on our rough data inventory to filter out the latest scene and create store it in a new dataframe.

```
[ ]: pylab.rcParams["figure.figsize"] = (13, 13)

# we give our inventory a shorter name iDf (for inventory Dataframe)
iDf = s1_project.inventory.copy()

# we select the latest scene based on the metadata entry endposition
latest_df = iDf[iDf.endposition == iDf.endposition.max()]

# we print out a little info on the date of the
print(" INFO: Latest scene found for {}".format(latest_df["acquisitiondate"].values[0]))

# we use the plotInventory method in combination with the newly
# created Geodataframe to see our scene footprint
s1_project.plot_inventory(latest_df, transparency=0.5)
```

7* Download scene

We use the build-in download method from the *Sentinel1* class. Note that you can pass any Geodataframe generated by OST, and filtered by you (e.g. sort out rows that you do not need). In our case we are only interested in the latest scene, so we pass the newly generated *latest_df* Geodataframe object.

NOTE that you should use ESA's scihub server in this case, since it is the place where the images arrive first. Other data mirrors might have slight delays, so that the scene found by the inventory might not be available.

```
[ ]: s1_project.download(latest_df)
```

8* - Display some metadata of the latest scene

After use of the *Sentinel1* class for finding and downloading the latest scene, we hand the scene identifier over to the *Sentinel1Scene* class for further processing as already demonstrated in OST Tutorial 1.

```
[ ]: # create a S1Scene class instance based on the scene identifier coming from our "latest␣
↪scene dataframe"
latest_scene = Sentinel1Scene(latest_df["identifier"].values[0])

# print summarising infos
latest_scene.info()
```

(continues on next page)

(continued from previous page)

```
# print file location
file_location = latest_scene.get_path(s1_project.download_dir)

print(" File is located at: ")
print(" " + str(file_location))
```

9* - Produce a subsetted ARD product

The creation of the ARD product follows the same logic as presented in OST Tutorial 1. However, for this case we introduce the subset argument to eh *create_ard* function. Subsetting is adviced if only a small portion of the whole image is of interest. It will speed up processing and uses less storage.

In our case we use some helper functions within the OST package to create a squared buffer area fo 10000 meter around our point of interest defined as AOI.

[ ]:
```
# 10 km buffer around AOI Point
from shapely.wkt import loads
from ost.helpers import vector as vec

# turn WKT into shapely geometry object
shp_aoi = loads(s1_project.aoi)

# use OST helper's function to create a quadrant buffer for subset
subset_area = vec.geodesic_point_buffer(shp_aoi.x, shp_aoi.y, 10000, envelope=True)

print("-----------------------------------------------------------------------------")
latest_scene.create_ard(
    # we see our download path can be automatically generated by providing the Project's␣
↪download directory
    infile=latest_scene.get_path(download_dir=s1_project.download_dir),
    # let's simply take our processing folder
    out_dir=s1_project.processing_dir,
    # define the subset
    subset=subset_area,
    # in case already processed, we will re-process
    overwrite=True,
)

print("-----------------------------------------------------------------------------")
print(" The path to our newly created ARD product can be obtained the following way:")
latest_scene.ard_dimap
```

10* - Create a RGB color composite

As already demonstrated in OST Tutorial 1, we create an RGB GeoTiff, and visualize it.

```
[ ]: latest_scene.create_rgb(
        outfile=s1_project.processing_dir / f"{latest_scene.start_date}.tif"
    )
    latest_scene.visualise_rgb(shrink_factor=1)
```

Open SAR Toolkit - Tutorial 4a, version 1.2, June 2020. Andreas Vollrath, ESA/ESRIN phi-lab

5.3.4 OST Tutorial IV-A

How to create near-daily timeseries over Vienna. Introduction to GRD Batch Processing part I.

Short description

This notebook provides an introduction to the batch processing of Sentinel-1 GRD data using OST's *Sentinel1Batch* class. This is a subclass of the *Sentinel1* class, and thus inherits all the functionalities of the *Generic* and the *Sentinel1* classes for the generation of a project as well as data search and refinement as presented in the OST Tutorial II notebook. The *Sentinel1Batch* class holds functions for the batch processing of single calibrated backscatter products. Furthermore, time-series processing and the generation of multi-temporal statistics, referred to as timescans, are introduced.

Within the given example, time-series for 4 different overlapping tracks are going to be produced over the city of Vienna, Austria. The notebook demonstrates:

1. the reduction of data processing by automatically subsetting the data,
2. time-series processing and the corresponding ARD types,
3. merging the track specific time-series into a unique time-series with almost daily coverage,
4. creation of a timeseries animation for outreach purposes.

Requirements

- a PC/Mac with at least 16GB of RAM
- about 25 GB of free disk space

- **a Copernicus Open Data Hub user account, ideally valid for at least 7 days (https://scihub.copernicus.eu)**

NOTE: all cells that have an * after its number can be executed without changing any code.

0* - Install OST and dependencies

NOTE: Applies only if you haven't fully installed OST yet, e.g. on Google Colab,

```
[ ]: # !apt-get -y install wget
     # !wget https://raw.githubusercontent.com/ESA-PhiLab/OST_Notebooks/master/install_ost.sh
     # !bash install_ost.sh
```

1* - Import of Libraries

In this step we import some standard python libraries for OS independent path handling as well as the *Sentinel1_GRDBatch* class thta handles the full workflow from search, download and processing of multiple GRD scenes. In addition, the OST helper module *vector* is loaded to create an AOI based on Point coordinates, and the *raster* module for creating a time-series animation.

```
[ ]: # this is for the path handling and especially useful if you are on Windows
     from pathlib import Path
     from pprint import pprint


     # this is the s1Project class, that basically handles all the workflow from beginning to␣
     ↪the end
     from ost import Sentinel1Batch
     from ost.helpers import vector, raster
```

2* - Set up the project

Here you going to initialize the *Sentinel1Batch* class by determining the project folder, the AOI and the start and end date. In addition we determine the image product type (i.e. GRD) and the ARD type that we use for processing. In this cas we choose the OST-RTC that produces Radiometrically Terrain Corrected products, i.e. the images are corrected for radiometric distortions along mountainous slopes. This type of ARD is advised when doing land cover and land use studiesover rugged terrain.

```
[ ]: # define the project directory
     project_dir = Path.home() / "OST_Tutorials" / "Tutorial_4a"

     # define aoi with a 2 point coordinates and create a buffer with 20km radius
     lat, lon = "48.25", "16.4"  # Vienna
     aoi = vector.latlon_to_wkt(lat, lon, buffer_meter=17500, envelope=True)

     # define the start and end date
     start = "2020-05-01"
     end = "2020-05-31"

     # initialize the class to s1_grd instance
     s1_grd = Sentinel1Batch(
```

(continues on next page)

(continued from previous page)

```
    project_dir=project_dir,
    aoi=aoi,
    start=start,
    end=end,
    product_type="GRD",
    ard_type="OST-RTC",
)

# do the search
if not s1_grd.inventory_file:
    s1_grd.search()
```

3* - Plot refined data inventory

The resultant dataframe from the search inventory is visualised. We do not do a refinement step here, since all images are fully overlapping the AOI. This allows us to create a combined, almost daily time-series of all images.

[ ]:
```
# -----------------------------------------------------
# for plotting purposes we use this iPython magic
%matplotlib inline
%pylab inline
pylab.rcParams["figure.figsize"] = (19, 19)
# -----------------------------------------------------

# plot the inventory
s1_grd.plot_inventory(s1_grd.inventory, transparency=0.1)
```

4* - Download of GRD scenes

As already shown in Tutorial II, you will download the scenes based on the refined inventory dataframe for the respective produckt key.

[ ]:
```
s1_grd.download(s1_grd.inventory, concurrent=10)
```

5* - Set ARD parameters

Similar to the *Sentinel1Scene* class (Tutorial I and III), the *Sentinel1Batch* class handles the defintion of ARD types in a hierarchical dictionary structure. You can use the same types and steps to customize as for the *Sentinel1Scene* class. For our example, we already intialised the class instance with the OST-RTC ARD type, in order to remove the backscatter distortion on mountainous slopes. This applies to all single image processing in the first step. The subsequent time-series processing will bring all the imagery to a common grid and apply a multitemporal speckle filter, that is much more efficient than speckle filtering applied on a single image. Since speckle filters a conceptionalised to work on the raw power data of SAR imagery, the conversion to the dB scale is only applied after the multi-temporal speckle filtering. In addition, all images are clipped to the very same extent, that is defined by the common data coverage of all images per track as well as the AOI.

Note that it is possible to change the datatype of the output to either unsigned 16 or 8-bit integer. The backscatter data is therefore linearly stretched between -30 to 5 dB. This has the advantage to reduce the necessary storage by a factor of 2 for 16-bit uint and a factor of 4 for 8-bit uint data.

The exact processing steps are as follows and depend on the ARD type:

```
[ ]: print(
         "-------------------------------------------------------------------------------------
     ↪-----------------------"
     )
     print("Time-series processing parameters hold in the configuration file:")
     print(
         "-------------------------------------------------------------------------------------
     ↪-----------------------"
     )
     print("")
     pprint(s1_grd.ard_parameters["time-series_ARD"])

     # custimze some single scene ARD parameters
     s1_grd.ard_parameters["single_ARD"][
         "resolution"
     ] = 50  # reduce for processing time and disk space
     s1_grd.ard_parameters["time-series_ARD"]["dtype_output"] = "uint8"
```

6* - Run the batch routine

To process all the data, including time-series and timescans is as easy as one command. All the complexity is handled by OST in the back, and you just have to wait, since processing can take quite a while. Note the keywords to aly higher level tim-series and timescan generation. Mosaicking refers toacross track mosaicking, which for this example is not the case. The *overwrite* argument tells OST if it should start processing from scratch (i.e. set to **True**), or process from where it stopped. The latter comes in handy when wokng on cloud instances that crash or automatically shutdown every once in a while.

Note that subsetting is set automatically to **True** when all tracks hold in the inventory fully overlap the AOI.

```
[ ]: s1_grd.grds_to_ards(
         inventory_df=s1_grd.inventory,
         timeseries=True,
         timescan=True,
         mosaic=False,
         overwrite=False,
     )
```

7* - Merge single per-track time-series to one single time-series

By using a little helper function from OST's raster module, combining the 4 time-series to a unique one is as easy the following command. Within your processing directory, a new foler called combined is created. If multi-temporal statistics for this new time-series should be created, set the timescan argument to **True**.

```
[ ]: raster.combine_timeseries(
         processing_dir=s1_grd.processing_dir, config_dict=s1_grd.config_dict, timescan=True
     )
```

8* - Create a time-series animation

Finally, a time-series animation is created. therefore we need to pass the time-series folder to the command. product_list expects a list of 1 to 3 elements. For GRD data this is either a single polarisation, or both bands. OST will calculate the power ratio of band 1 and 2 for a 3-band RGB composite. A shrink factor can be set to reduce image resolution and memory needs.

Note: This needs imagemagick installed, which is not a default requirement by OST. You can install it on e.g. Ubuntu by typing: sudo apt-get install magick

```
[ ]: # define Time-series folder
ts_folder = s1_grd.processing_dir / "combined" / "Timeseries"

# create the animation
raster.create_timeseries_animation(
    timeseries_folder=ts_folder,
    product_list=["bs.VV", "bs.VH"],
    out_folder=s1_grd.processing_dir,
    shrink_factor=3,
    add_dates=True,
)
```

Open SAR Toolkit - Tutorial 4b, version 1.2, June 2020. Andreas Vollrath, ESA/ESRIN phi-lab

5.3.5 OST Tutorial IV-B

How to create a timeseries animation of Iceberg A-68. Introduction to GRD Batch Processing part II.

Short description

This notebook continues to introduce you to the general workflow of OST for the batch processing of GRD data using the *Sentinel1Batch* class. In this example:

1. across-track mosaicking based on a refined inventory and
2. processing in polar regions is shown.

Requirements

- a PC/Mac with at least 16GB of RAM
- about 75 GB of free disk space

- **a Copernicus Open Data Hub user account, valid for at least 7 days (https://scihub.copernicus.eu)**

NOTE: all cells that have an * after its number can be executed without changing any code.

0* - Install OST and dependencies

NOTE: Applies only if you haven't fully installed OST yet, e.g. on Google Colab,

```
[ ]: # !apt-get -y install wget
     # !wget https://raw.githubusercontent.com/ESA-PhiLab/OST_Notebooks/master/install_ost.sh
     # !bash install_ost.sh
```

1* - Import of libraries

```
[ ]: from pathlib import Path
     from pprint import pprint

     from ost import Sentinel1Batch
     from ost.helpers import vector, raster

     # -----------------------------------------------------
     # for plotting purposes we use this iPython magic
     %matplotlib inline
     %pylab inline
     pylab.rcParams["figure.figsize"] = (19, 19)
     # -----------------------------------------------------
```

2 - Set up the project

This follows the logic of the prior OST Tutorial notebooks.

```
[ ]: # define a project directory
     home = Path.home()
     # create a processing directory
     project_dir = home / "OST_Tutorials" / "4b"

     # define aoi with helper function, i.e. get a buffered area around point coordinates
     lat, lon = "-67", "-61"
     aoi = vector.latlon_to_wkt(lat, lon, buffer_degree=1.5, envelope=True)

     # define the start and end date
     start = "2017-06-30"
     end = "2017-08-31"

     # initialize the class to s1_grd instance
```

(continues on next page)

(continued from previous page)

```
s1_grd = Sentinel1Batch(
    project_dir=project_dir, aoi=aoi, start=start, end=end, product_type="GRD"
)

# trigger the search
s1_grd.search()
s1_grd.plot_inventory()
```

3 - Search Refinement

In order to create a time-series of multiple tracks, a pre-condition is that all tracks feature the same amount of acquistions within our Time of Interest. Let's use some pandas syntax to see if this is the case:

[ ]:
```
df = s1_grd.inventory.pivot_table(
    index=["relativeorbit", "acquisitiondate"], aggfunc="size"
).reset_index()
df.pivot_table(index="relativeorbit", aggfunc="size").reset_index()
```

As in most cases, we do not fulfill this pre-condition. By considering all tracks, our time-series would need to be reduced to 5 acquisitions. However, images taken over track 9 are not necessary, since our AOI is fully covered by the other 2 tracks.

As already mentioned in OST Tutorial 2, the *refine_inventory* method takes care of those issues and prepares the inventory in a way that it is suitable for across-track mosaic time-series. This includes the splitting of images by orbit direction and polarzation mode in the first place. In addition, it checks if some tracks can be excluded because all the others fully overlap the AOI. In this way we reduce the amount of images to process, while optimising for our later time-series processing. See OST Tutorial 2 for full explanation and arguments.

[ ]:
```
# do the refinement
s1_grd.refine_inventory()
```

The output of the refinement procedure gives some infos, e.g. the exclusion of track 9. At the very end it summarises the information. Since in our case we only have imagery acquired in descending orbit and HH polarization, we see that 10 mosaics in time can be created. Another **important** infomation is the **key** defiend by orbit direction and polarisation, i.e. **DESCENDING_HH**. We will need this to select the refined inventory stored in the *refined_inventory_dict* attribute of our class instance as follows:

[ ]:
```
# select the key from output of refinement command
key = "DESCENDING_HH"

# we wrap the information of the length of our refined inventory in a print statement
print(
    f"The refined inventory holds {len(s1_grd.refined_inventory_dict[key])} acquisitions␣
↪to process."
)

# we plot the full Inventory on a map
s1_grd.plot_inventory(s1_grd.refined_inventory_dict[key], transparency=0.05)
```

3 - Data download

Here we download the data. It is best to use the ASF data mirror.

```
[ ]: s1_grd.download(s1_grd.refined_inventory_dict[key], concurrent=10)
```

5* - Customise ARD parameters

1. For data reduction we lower the resolution to 100 meters.
2. This will also reduce speckle, so we do not need it neither.
3. We do not care about Layover and Shadow for this example, since there is anyway no high-resolution DEM for Antarctica that could provide sufficient inforamtion.
4. Our time-series output will be scaled to dB scale for better stretching and visualisation
5. We further reduce the amount of data by converting from 32-bit float to unsigned 8-bit integer data type
6. Our AOI is only a rough seletion for the Area of Interest. We do not want to cut it to the boundaries to see the full data provided by the selected imagery.

```
[ ]: s1_grd.ard_parameters["single_ARD"]["resolution"] = 100
s1_grd.ard_parameters["single_ARD"]["remove_mt_speckle"] = False
s1_grd.ard_parameters["single_ARD"]["create_ls_mask"] = False
s1_grd.ard_parameters["single_ARD"]["dem"]["dem_name"] = "GETASSE30"
s1_grd.ard_parameters["time-series_ARD"]["to_db"] = True
s1_grd.ard_parameters["time-series_ARD"]["dtype_output"] = "uint8"
s1_grd.ard_parameters["mosaic"]["cut_to_aoi"] = False

pprint(s1_grd.ard_parameters)
```

6* - Produce Timeseries Mosaics

Now the creation of our mosaic time-series is just a single command away.

```
[ ]: s1_grd.grds_to_ards(
    inventory_df=s1_grd.refined_inventory_dict[key],
    timeseries=True,
    timescan=False,
    mosaic=True,
    overwrite=False,
)
```

7* - Creation of an animated GIF of a timeseries

Note: This needs imagemagick installed, which is not a default requirement by OST. You can install it on e.g. Ubuntu by typing: sudo apt-get install magick

[ ]:
```
from ost.helpers import raster

# define the timeseries folder for which the animation should be created
ts_folder = s1_grd.processing_dir / "Mosaic" / "Timeseries"

raster.create_timeseries_animation(
    ts_folder,
    ["bs.HH"],
    s1_grd.processing_dir,
    shrink_factor=15,
    add_dates=True,
    duration=0.25,
)
```

Open SAR Toolkit - Tutorial 4c, version 1.2, August 2020. Andreas Vollrath, ESA/ESRIN phi-lab

5.3.6 OST Tutorial IV - C

Create country-wide mosaic time-series and timescan. Introduction to GRD Batch processing part III.

Short description

This notebook is very similar to the Tutorial IVa, with the difference that you will process GRD data over a larger area and create time-series and timescan mosaics. It therefore represents the workflow for large-scale processing.

Requirements

- a PC/Mac with at least 16GB of RAM
- about 150GB of free disk space

- **a Copernicus Open Data Hub user account, valid for at least 7 days (https://scihub.copernicus.eu)**

NOTE: all cells that have an * after its number can be executed without changing any code.

0* - Install OST and dependencies

NOTE: Applies only if you haven't fully installed OST and its dependencies yet, e.g. on Google Colab, so uncomment the lines in this case.

[ ]:
```
# !apt-get -y install wget
# !wget https://raw.githubusercontent.com/ESA-PhiLab/OST_Notebooks/master/install_ost.sh
# !bash install_ost.sh
```

1* - Import of Libraries

In this step we import some standard python libraries for OS independent path handling as well as the *Sentinel1_GRDBatch* class thta handles the full workflow from search, download and processing of multiple GRD scenes. In addition, the OST helper module *vector* is loaded to create an AOI based on Point coordinates, and the *raster* module for creating a time-series animation.

[ ]:
```
from pathlib import Path
from pprint import pprint

from ost import Sentinel1Batch
from ost.helpers import vector, raster
```

2* - Set up the project

Here you going to initialize the *Sentinel1_GRDBatch* class by determining the project folder, the AOI and the start and end date. Since you should be already familiar with the *search* and *refine* functions, we execute them within the same cell.

[ ]:
```
# define a project directory
home = Path.home()
# create a processing directory
project_dir = home / "OST_Tutorials" / "Tutorial_4c"

# define aoi with helper function, i.e. get a buffered area around point coordinates
aoi = "IRL"

# define the start and end date
start = "2019-02-01"
end = "2019-04-30"

# initialize the class to s1_grd instance
s1_grd = Sentinel1Batch(
    project_dir=project_dir, aoi=aoi, start=start, end=end, product_type="GRD"
)

# trigger the search
```

(continues on next page)

(continued from previous page)

```
s1_grd.search()

# optional: once you did the search the first time, you can load
# the full inventory uncommenting the follwoing 2 lines
# and commenting the search command
# s1_grd.inventory_file = s1_grd.inventory_dir/"full.inventory.gpkg"
# s1_grd.read_inventory()

# do the refinement
s1_grd.refine_inventory()
```

3* - Plot refined data inventory

Here you will visualize the resultant dataframes from the refined search inventory based on the product key.

[ ]:
```
# ------------------------------------------------
# for plotting purposes we use this iPython magic
%matplotlib inline
%pylab inline
pylab.rcParams["figure.figsize"] = (19, 19)
# ------------------------------------------------

# search command
key = "ASCENDING_VVVH"
print(
    f"Our refined inventory holds {len(s1_grd.refined_inventory_dict[key])} frames to␣
↪process."
)
# we plot the full Inventory on a map
s1_grd.plot_inventory(s1_grd.refined_inventory_dict[key], transparency=0.1)
```

4* - Download of GRD scenes

As already shown in Tutorial II, you will download the scenes based on the refined inventory dataframe for the respective produckt key.

[ ]:
```
s1_grd.download(s1_grd.refined_inventory_dict[key])
```

5* - Set ARD parameters

Similar to the *Sentinel1-Scene* class (Tutorial I and III), the *Sentinel1-GRDBatch* class handles the defintion of ARD types in a hierarchical dictionary structure. You can use the same types and steps to customize as for the *Sentinel1-Scene* class.

[ ]:
```
# single scene ARD parameters
s1_grd.ard_parameters["single_ARD"]["resolution"] = 50
s1_grd.ard_parameters["single_ARD"]["product_type"] = "GTC-gamma0"
s1_grd.ard_parameters["single_ARD"]["create_ls_mask"] = False
```

(continues on next page)

(continued from previous page)

```
# time-series ARD
s1_grd.ard_parameters["time-series_ARD"]["remove_mt_speckle"] = False

# our borders for Ireland are quite rough. We therefore won't clip the final mosaics
s1_grd.ard_parameters["mosaic"]["cut_to_aoi"] = False

# s1_grd.config_dict['temp_dir'] = '/ram'
pprint(s1_grd.ard_parameters)
```

6* - Run the batch routine

To process all the data, including time-series and timescans is as easy as one command. All the complexity is handled behind, and you just have to wait, since processing can take quite a while.

[ ]:
```
s1_grd.grds_to_ards(
    inventory_df=s1_grd.refined_inventory_dict[key],
    timeseries=True,
    timescan=True,
    overwrite=False,
)
```

7* - Create a time-series animation

For interactive presentations it is nice to have animated "movies". The following command allows you to create animated time-series of oyur processed data.

[ ]:
```
# define Time-series folder
ts_folder = s1_grd.processing_dir / "23" / "Timeseries"

# create the animation
raster.create_timeseries_animation(
    timeseries_folder=ts_folder,
    product_list=["bs.VV", "bs.VH"],
    out_folder=s1_grd.processing_dir,
    shrink_factor=10,
    add_dates=True,
)
```

Open SAR Toolkit - Tips and Tricks, version 1.0, September 2019. Andreas Vollrath, ESA/ESRIN phi-lab

5.3.7 OST Tips and Tricks

This notebook provides code snippets that might be useful for specific OST usage.

Short description

This notebook shows some useful low level functionality of OST, as well as some tricks that can be helpful for certain projects.

- **1:** Create a squared AOI from point coordinates
- **2:** Create a OST confrom download directory from already downloaded files
- **3:** Create the Time of Interest using python's datatime class
- **4:** Load an already created inventory file into a OST class instance.
- **5:** How to download an offline scene from ESA scihub archive

- **6: Speed up processing by using a ram disk for temporary file storage**

1 - Create a squared AOI from Lat/Lon point coordinates

In case you do not have a shapefile of your Area Of Interest (AOI), but rather want to define it by Latitude and Longitude, considering a buffer, there is a helper function that let you do exactly this.

Note that there are 2 buffer options, in meter and in degree, respectively. The buffer in meter does the transform from Lat/Lon into meters based on a equidistant projection. This may result in non-sqaured rectangles towards the poles when plotting on Lat/Lon grid (see second cell)

[ ]:
```
# import of of vector helper functions of ost
from ost.helpers import vector

# define point by lat/lon coordinates
lat, lon = "78", "12"

# apply function with buffer in meters
wkt1 = vector.latlon_to_wkt(lat, lon, buffer_degree=0.5, envelope=True)
wkt2 = vector.latlon_to_wkt(lat, lon, buffer_meter=10000, envelope=True)
print(wkt1)
print(wkt2)
```

[ ]:
```
# we plot the wkt with geopandas and matplotlib
import geopandas as gpd
import matplotlib.pyplot as plt

# load world borders for background
world = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
# import aoi as gdf
aoi1 = vector.wkt_to_gdf(wkt1)
```

(continues on next page)

(continued from previous page)

```
aoi2 = vector.wkt_to_gdf(wkt2)
# get bounds of AOI
bounds = aoi1.geometry.bounds
# get world map as base
base = world.plot(color="lightgrey", edgecolor="white")
# plot aoi
aoi1.plot(ax=base, color="None", edgecolor="black")
aoi2.plot(ax=base, color="None", edgecolor="red")

# set bounds
plt.xlim([bounds.minx.min() - 2, bounds.maxx.max() + 2])
plt.ylim([bounds.miny.min() - 2, bounds.maxy.max() + 2])
plt.grid(color="grey", linestyle="-", linewidth=0.1)
```

2 Create a OST conform download directory

OST stores downloaded data in a certain directory hierarchy. In case you already downloaded your Sentinel-1 data by yourself, you can automatically create an OST conform directory, where all scenes found in the input directory, will be moved into its hierarchical structure.

[ ]:
```
from ost.s1 import download

input_directory = "/path/to/files/already/downloaded"
output_directory = "/path/to/OST/download/directory"
download.restore_download_dir(input_directory, output_directory)
```

3 Create the Time of Interest using python's datetime class

Sometimes it is wanted to create dynamic timestamps for the defintion of time of interest. Here we show python's datetime library is used in combination with the OST format needed for class instantion.

3.1. Last 30 days from today onwards.

Note, we do not need to set an end date, since this is by default today.

[ ]:
```
from datetime import datetime, timedelta

start_date = (datetime.today() - timedelta(days=30)).strftime("%Y-%m-%d")
print(start_date)
```

3.2. Target day (create date range around a certain date)

```
[ ]: # we set only the start date to today - 30 days
target_day = "2018-11-28"
delta_days = 60

# we set start and end 60 days before, repsectively after event
start = (
    datetime.strptime(target_day, "%Y-%m-%d") - timedelta(days=delta_days)
).strftime("%Y-%m-%d")
end = (datetime.strptime(target_day, "%Y-%m-%d") + timedelta(days=delta_days)).strftime(
    "%Y-%m-%d"
)

print(start, end)
```

4 Load an already created inventory file into a OST class instance.

Sometimes you need ot re-initialize the one of the batch processing classes. This results in an empty inventory atttriubte. In order ot skip the search on scihub you can load an already created inventory shapefile and set it as attribute in the following way:

```
[ ]: s1_instance = Sentinel1Batch(args)
s1_instance.inventory_file = s1_instance / "full_inventory.gpkg"
s1_instance.read_inventory()
```

5 How to download an offline scene from ESA scihub archive

ESA's scihub catalogue features an rolling archive of Sentinel-1 data meaning that older products are offline and need to be produced on demand. OST provides the functionality to do that in a programmatic way.

```
[ ]: from ost import Sentinel1_Scene
from ost.helpers.scihub import connect

# create instance
s1 = Sentinel1_Scene(
    "S1A_IW_GRDH_1SDV_20141004T230354_20141004T230423_002686_002FFD_062B"
)
# connection to Scihub
opener = connect()
# heck online status
print("The scene's online status is: ", s1.scihub_online_status(opener))
```

```
[ ]: import sys

# trigger production
s1.scihub_trigger_production(opener)

# let's run a loop until the scene is online
while status is False:
```

(continues on next page)

(continued from previous page)

```
    sys.sleep(60)  # just to not ask every millisecond, production can take a while
    status = s1.scihub_online_status(opener)
    print(status)

s1.download("/path/to/download")
```

6 Speed up processing by using a ram disk for temporary filestorage

On UNIX systems it is possible to mount part of your RAM as a hard disk. If you have enough RAM, you can use this as a directory for temporary file storage. Since the RAM is very fast in terms of read/write operations, processing can accelareted quite a bit.

Note that you need to run those commands on the command line and you will need administrative or superuser privilegues.

Here is an example for a 30 GB big ramdisk mounted at /ramdisk:

```
sudo mkdir /ramdisk
sudo mount -t tmpfs -o size=30G tmpfs /ramdisk
```

After that you can set your temp_dir to the mount point as in the following example (adjusting the other keywords for the class initialization of course)

[ ]:
```
from ost import Sentinel1Batch

project = Sentinel1Batch(
    project_dir="/your/project/dir",
    aoi="your/aoi",
    start="2019/01/01",
    end="2019-12-31",
)
project.config_dict["temp_dir"] = "/ramdisk"
```

PYTHON MODULE INDEX

O

INDEX

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Z