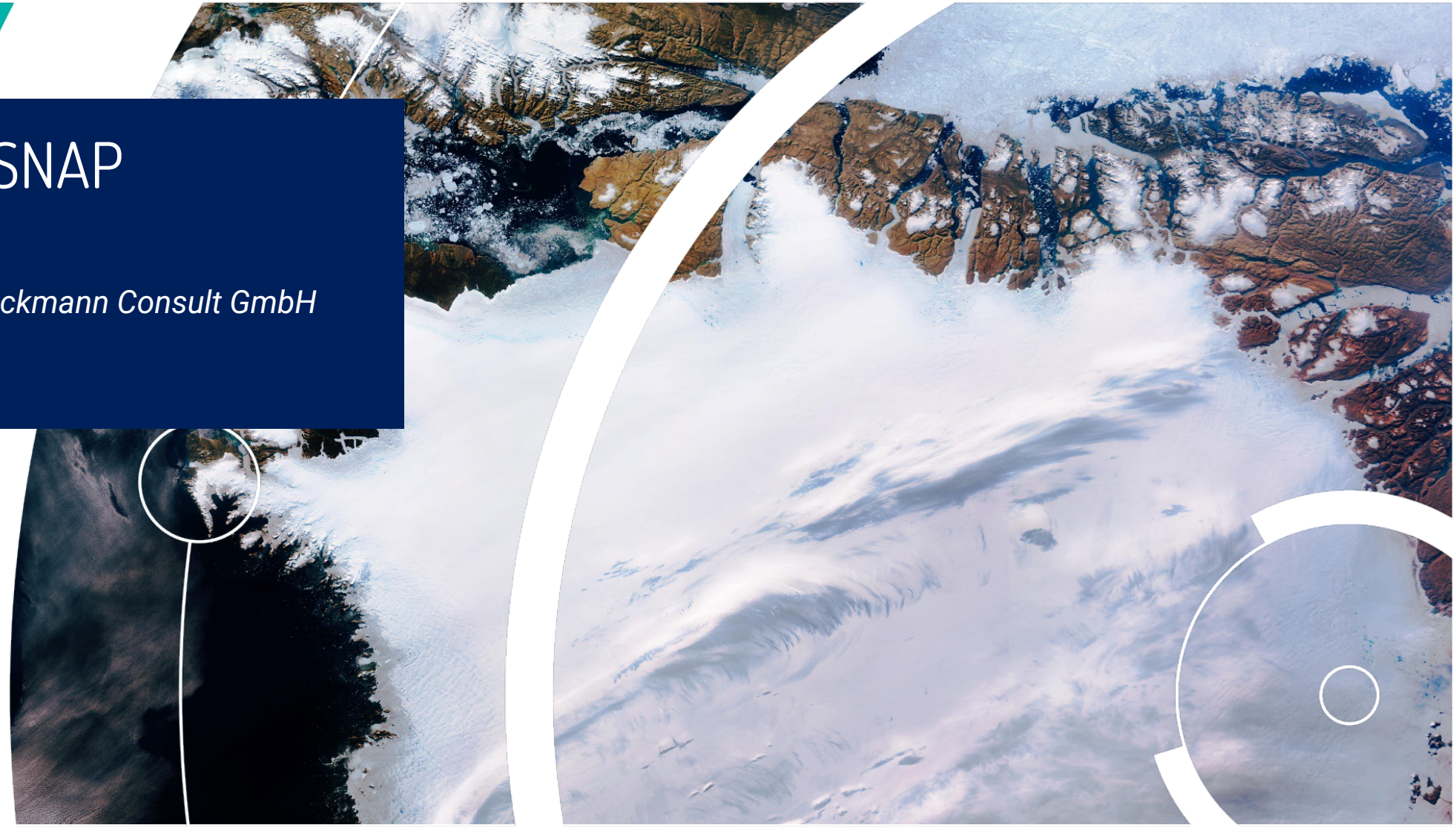
Introduction to SNAP

Ana B. Ruescas
Remote Sensing Expert, Brockmann Consult GmbH

OSOW 2023



SNAP

Earth Observation Made Easy

SNAP is

- an ecosystem to analyse, process and communicate Earth Observation data
- an Open-Source Project - github.com/senbox-org
- scalable to run on notebooks up to large production clusters
- used for scientific analysis, operational production and training
- easy to use

SNAP can

- access many satellite-based Earth Observation data products as well as generic raster formats directly in the cloud
- visualise the data in many ways
- analyse data using statistical functions, mathematical operations, correlation, comparison with point and vector data
- process satellite data with instrument specific as well as generic raster data operations
- save sessions and export results in various raster and non-raster formats
- be extended using Java and Python API

SNAP has

- comprehensive documentation - step.esa.int
- > 1 Million users and active community, > 10 000 forum users – forum.step.esa.int
- a long-term commitment of the European Space Agency

SNAP Team and Heritage

- Development of SNAP and the Sentinel Toolboxes started in 2014
- BEAM and NEST, the predecessors, started in 2000
- More than 20 years available
- ESA is planning with SNAP beyond 2027

- Development consortium
  - Brockmann Consult, Germany
  - Skywatch, Canada
  - CS Romania
  - CS France

BROCKMANN CONSULT GMBH

SKYWATCH

CS GROUP

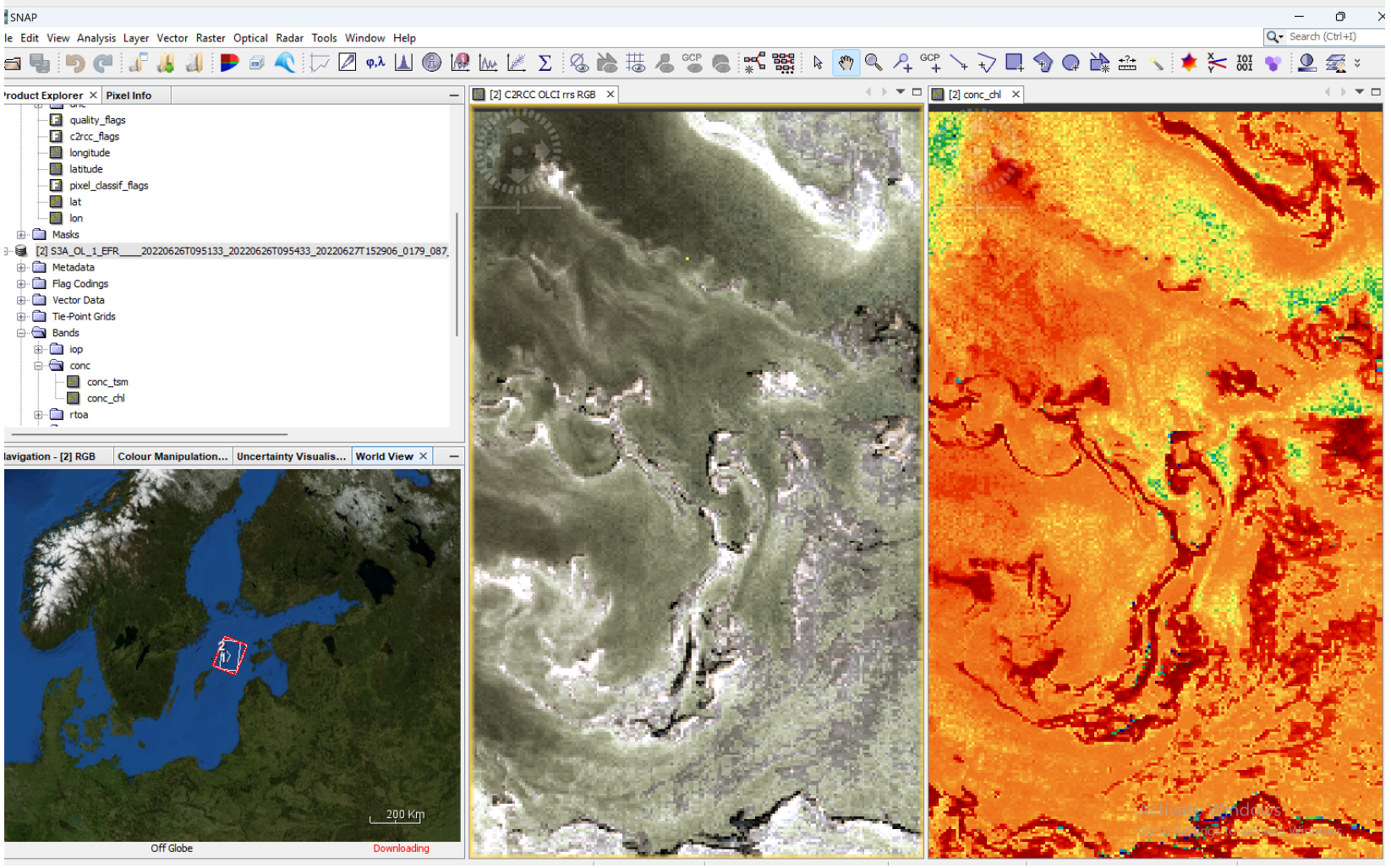
CS GROUP

Data Visualisation

SNAP Desktop is the GUI application which allows access to a large number of EO and generic raster data.

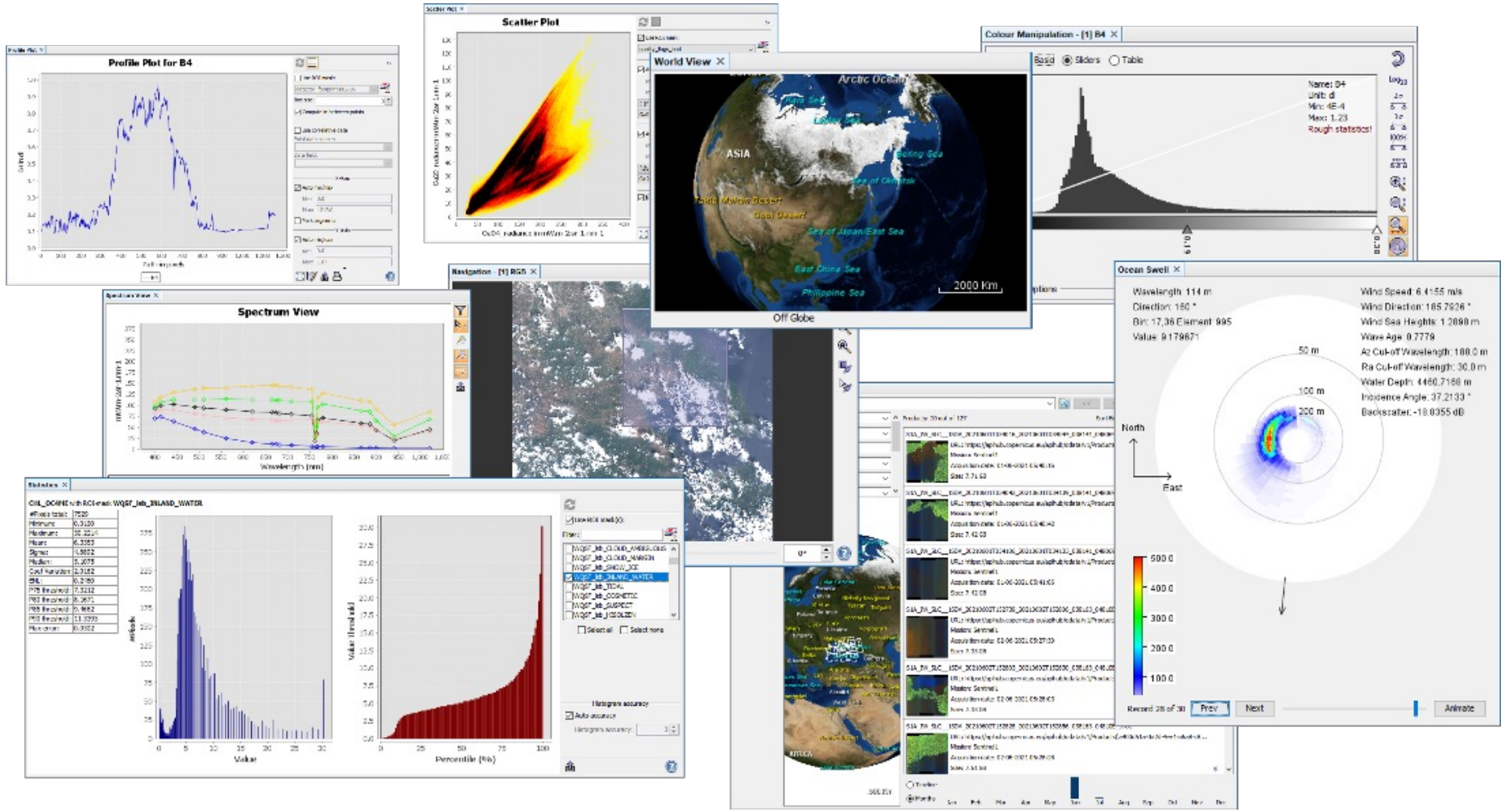
It provides various tools to display the data, and to visually analyse them.

The figure on the right shows an RGB of a Sentinel 3 product together with a visualisation of a water quality parameter.



Data Analysis

SNAP provides a rich suite of tools for data analysis, including profile and spectrum plots, statistical analysis, extraction of points through time series, and comparison with reference data (match-ups).

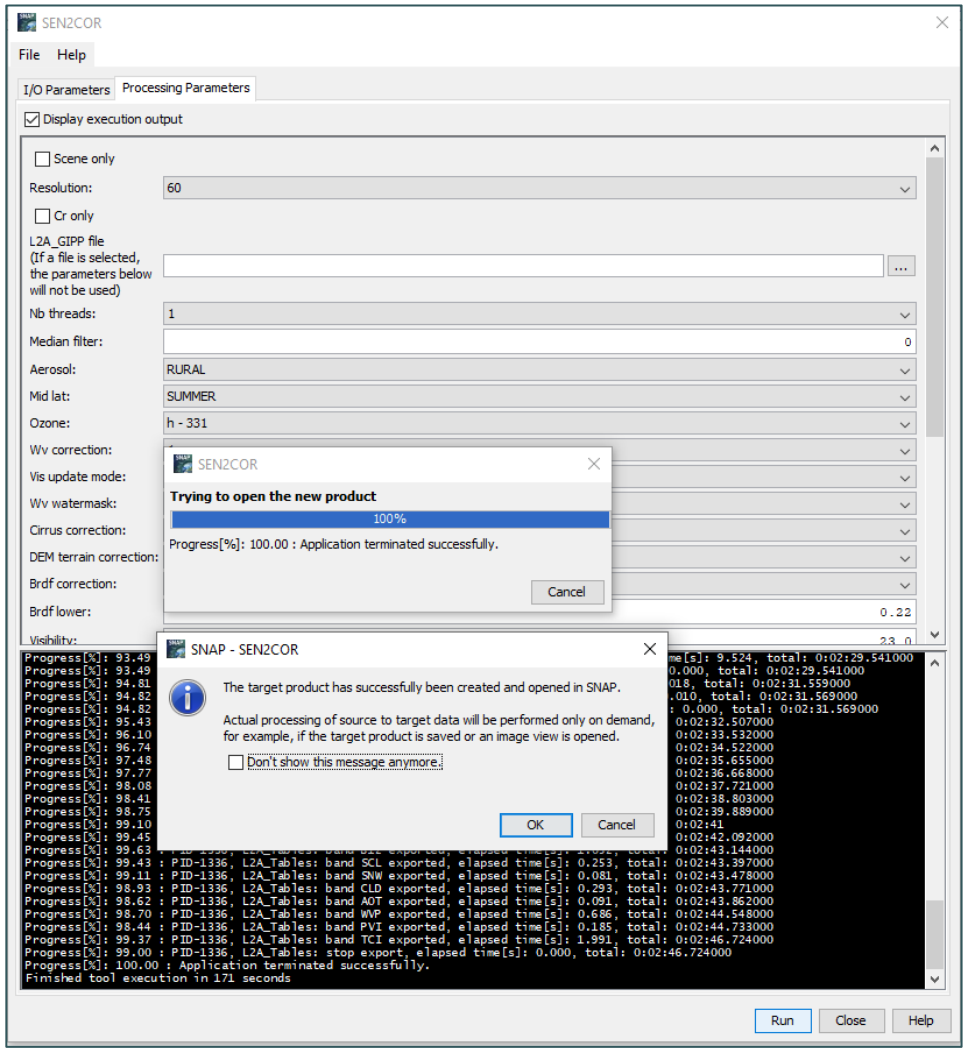


The figure shows some of the graphical analysis tools included in SNAP.

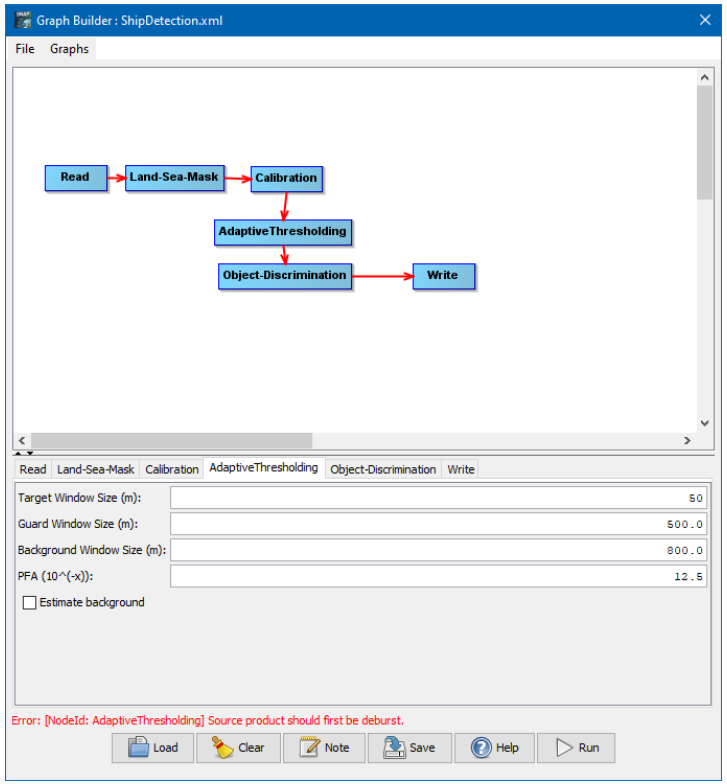
Data Processing

SNAP Data Processors analyse one or more input products and generate a new output product. Processors exist for generic operations such as band arithmetic, map projection or temporal aggregation. SNAP also provides a very large number of thematic processors, e.g. for atmospheric correction, biophysical indices calculation or retrieval of water quality. And SNAP supports special calibration of correction of satellite instruments with dedicated processors.

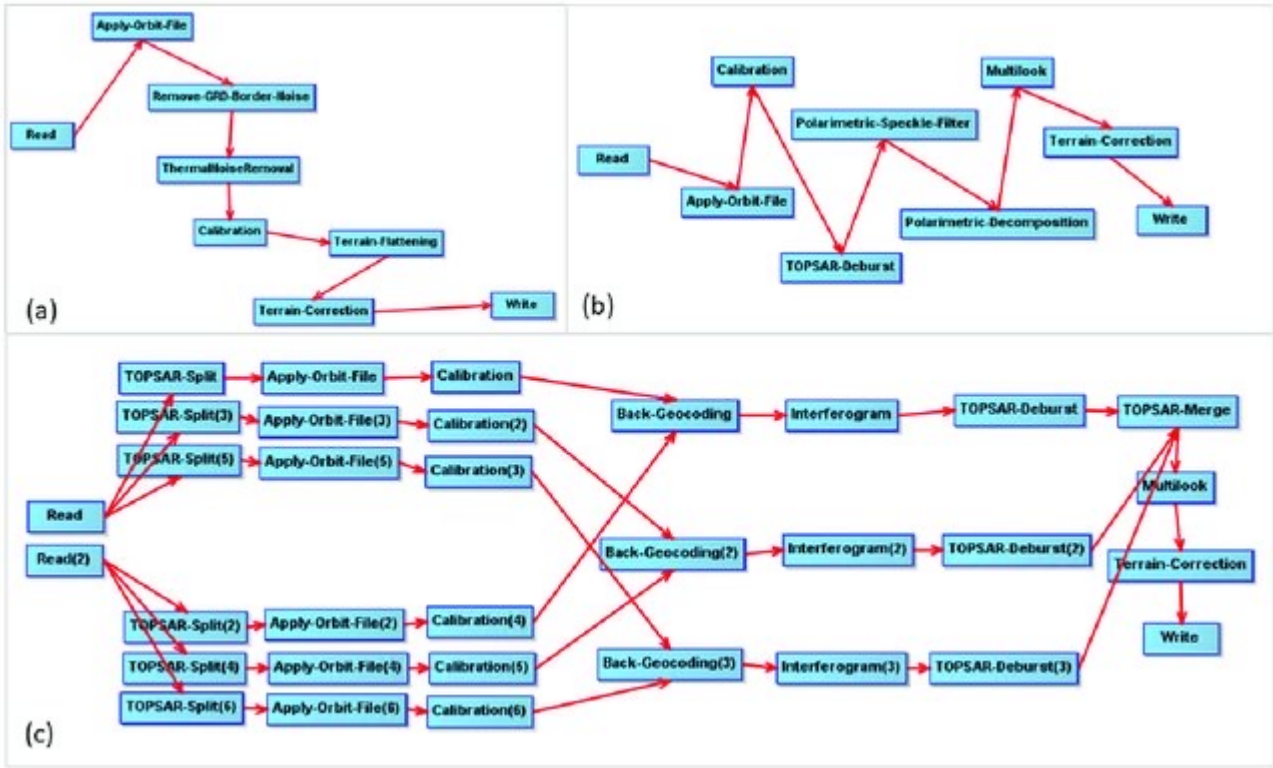
The figure shows the GUI for the Sentinel-2 Atmospheric Correction Processor sen2cor. The screenshot was taken when the processor was successfully executed. The black background shows the logging information during execution of the processor.



Data Processing



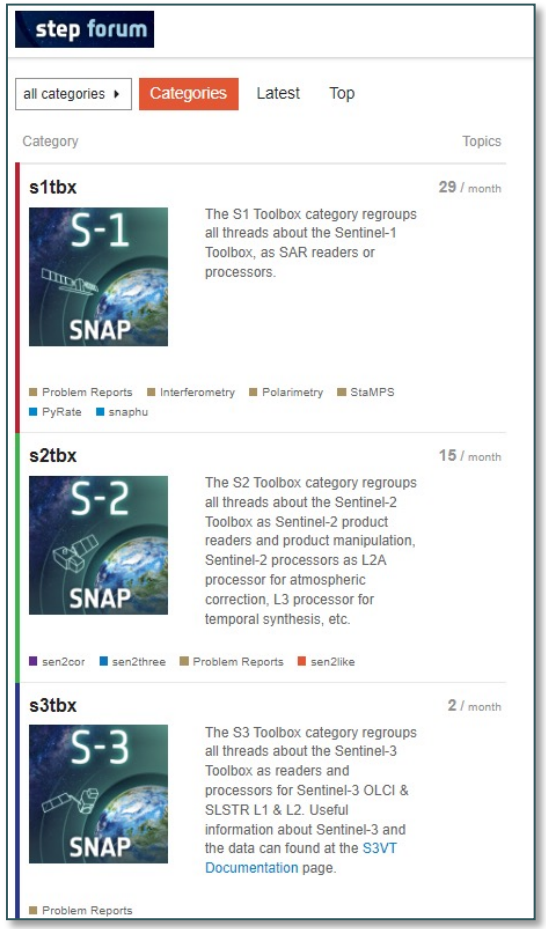
The SNAP graph builder allows to connect SNAP operators in processing graphs. These can be executed locally or in large clusters and cloud systems.



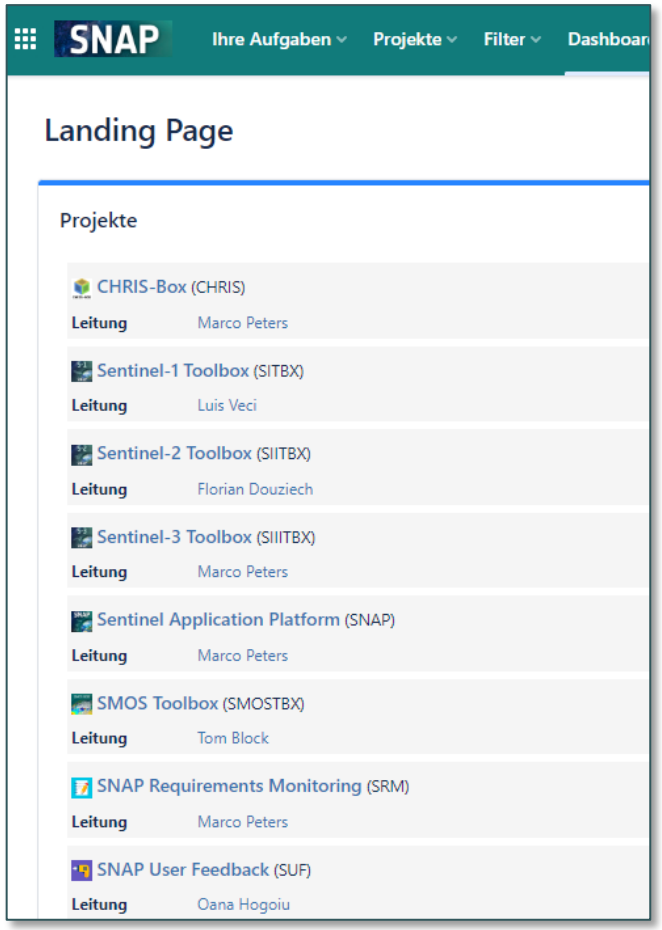
Example: Workflows in the SNAP graph builder tool for producing Synthetic Aperture Radar (SAR) analysis ready data (ARD) products. From Ticehurst, et al (2019). Building a SAR-Enabled Data Cube Capability in Australia Using SAR Analysis Ready Data. Data. 4. 100. 10.3390/data4030100.

User support

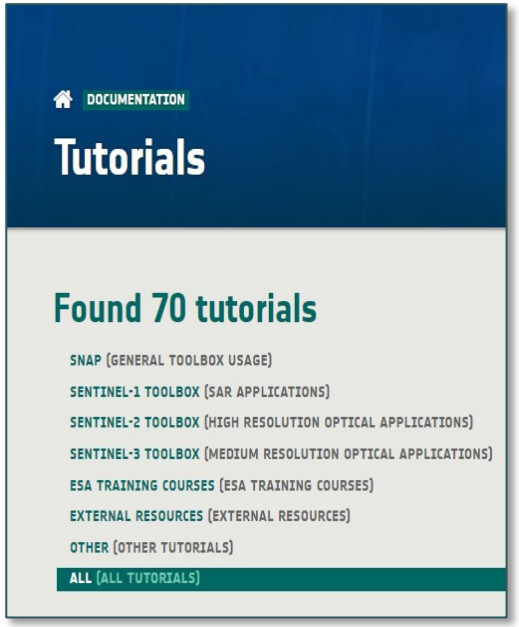
User Forum



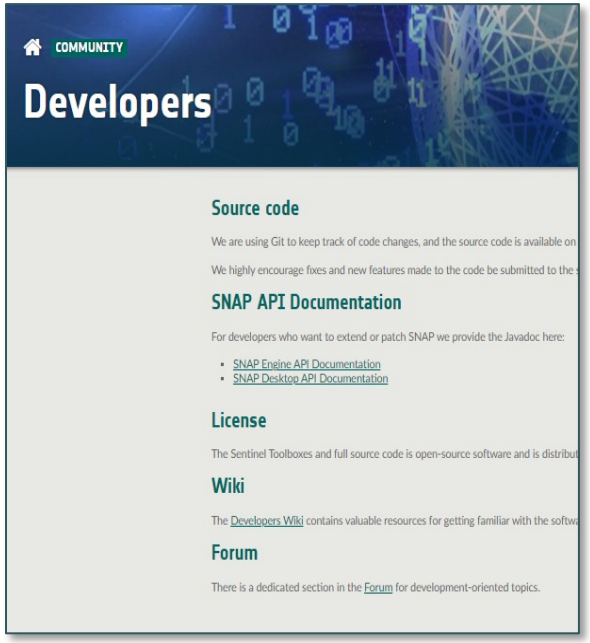
Issue Tracker



Tutorials

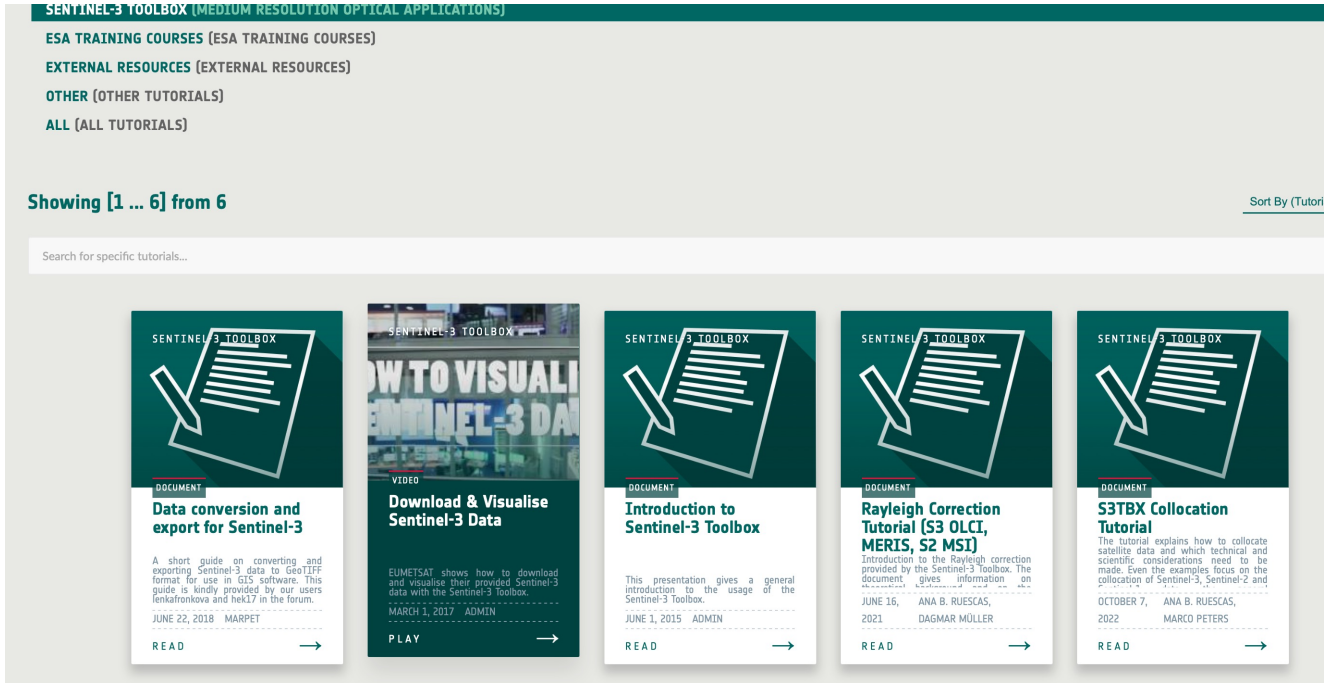


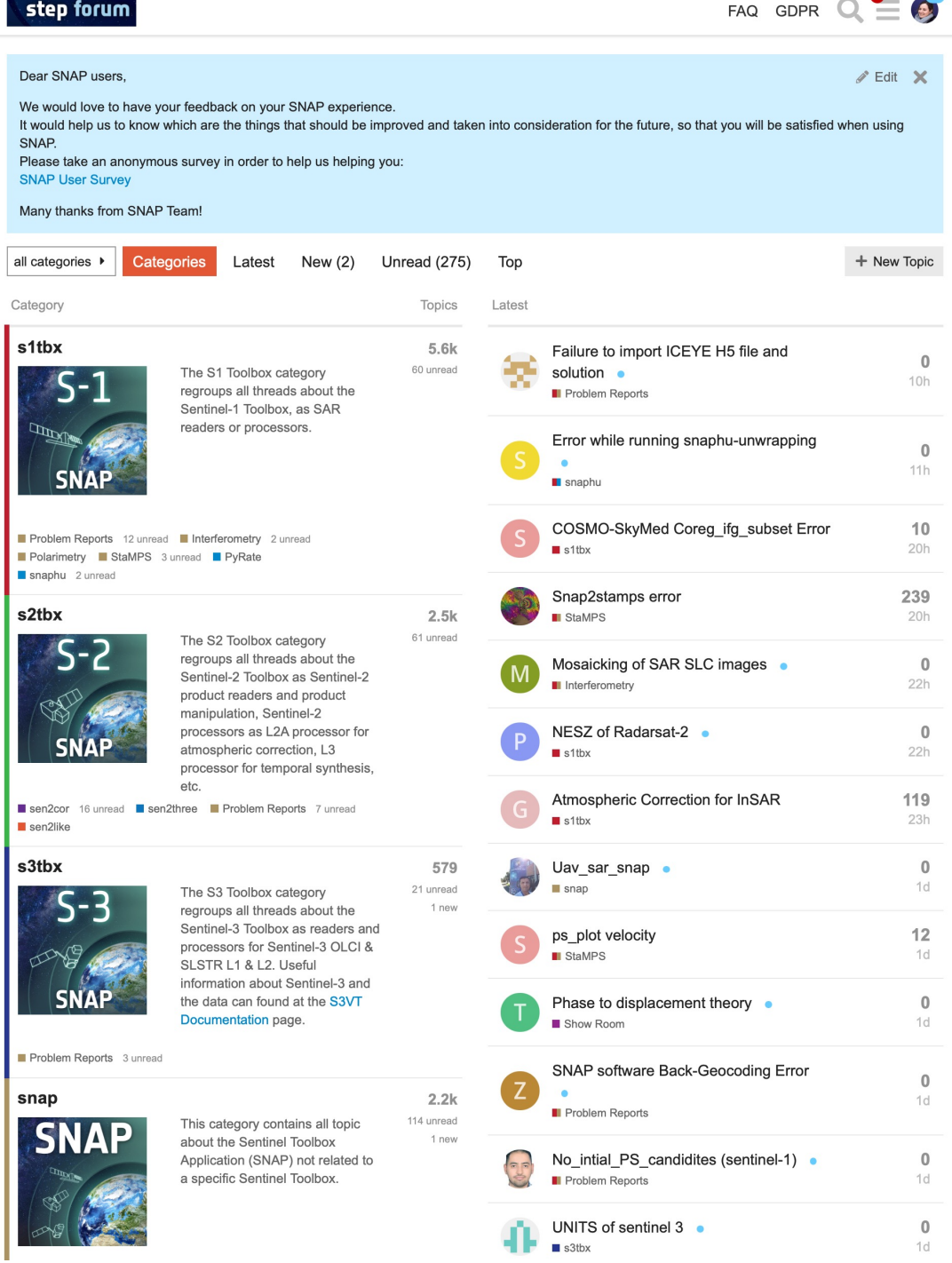
For Developers



SNAP Community & Tutorials

http://forum.step.esa.int/c/s3tbx
http://step.esa.int/main/doc/tutorials/
https://senbox.atlassian.net/wiki/spaces/SNAP/pages/1898053693/SNAP+FAQs





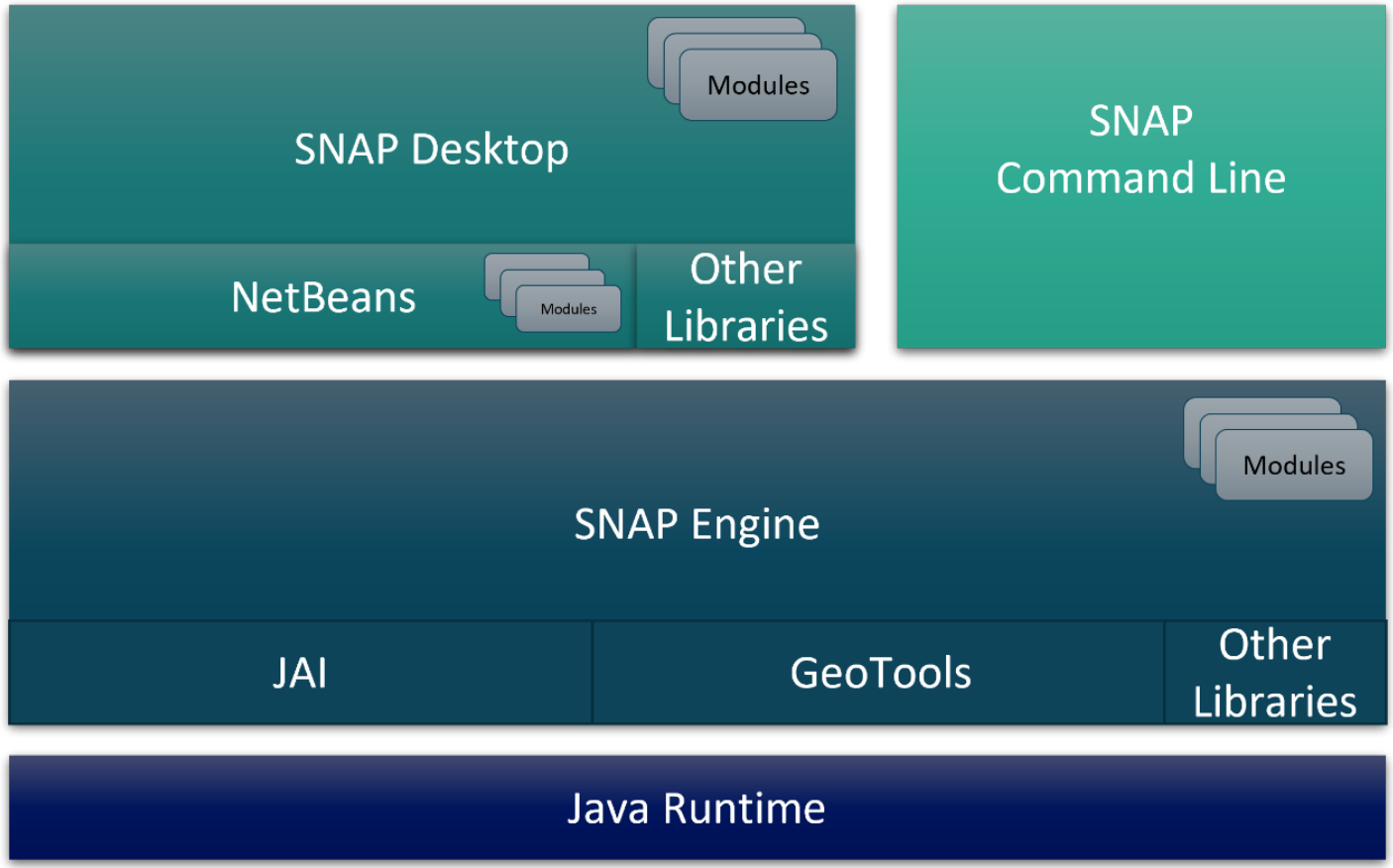
System Architecture

SNAP system architecture is structured into four layers.

At the bottom, the JDK provides the runtime environment and abstracts from the operating system.

On top of the JDK sits the SNAP engine. It provides the common functionality, the product data model, the processing framework, and the API for using and extending SNAP.

SNAP Desktop and Command Line interface are based on top of SNAP Engine and add the user interfaces.

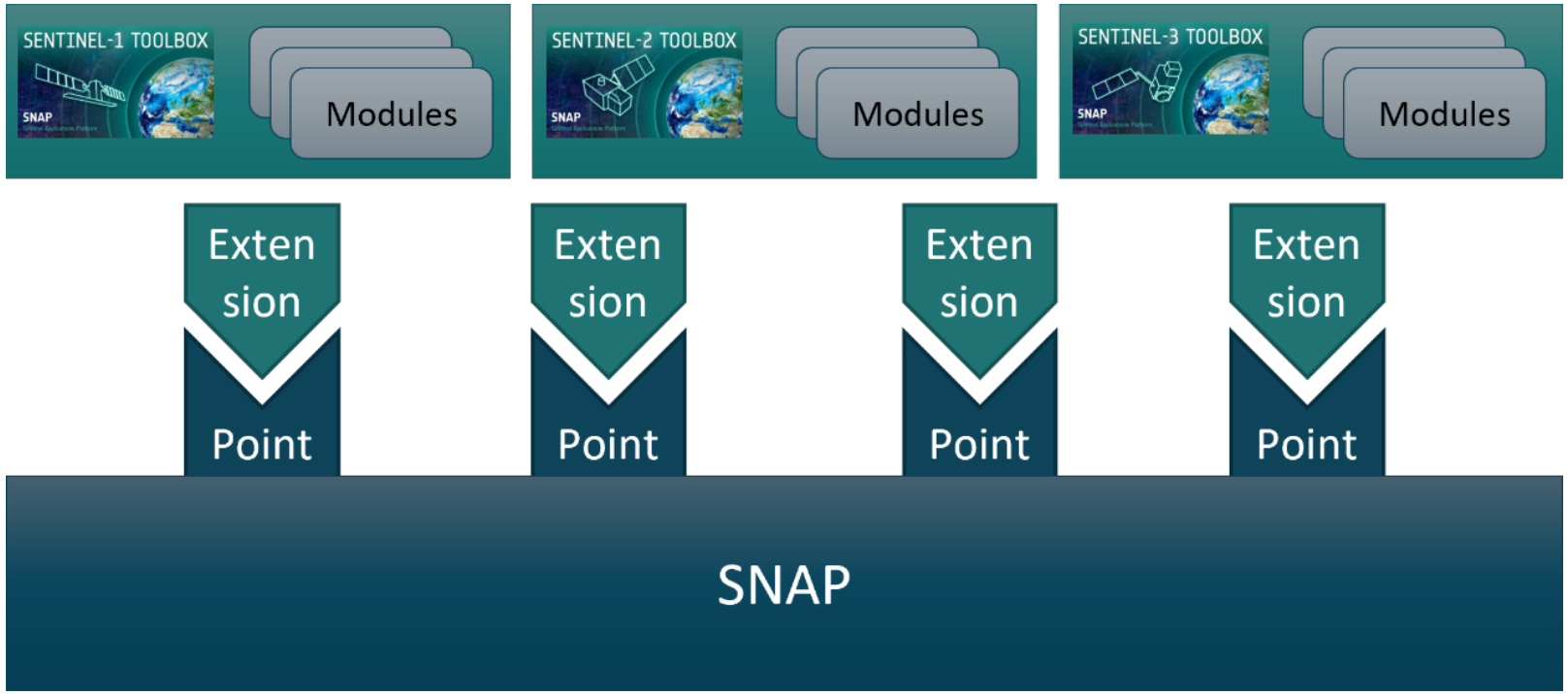


Toolbox Concept (Plug-ins)

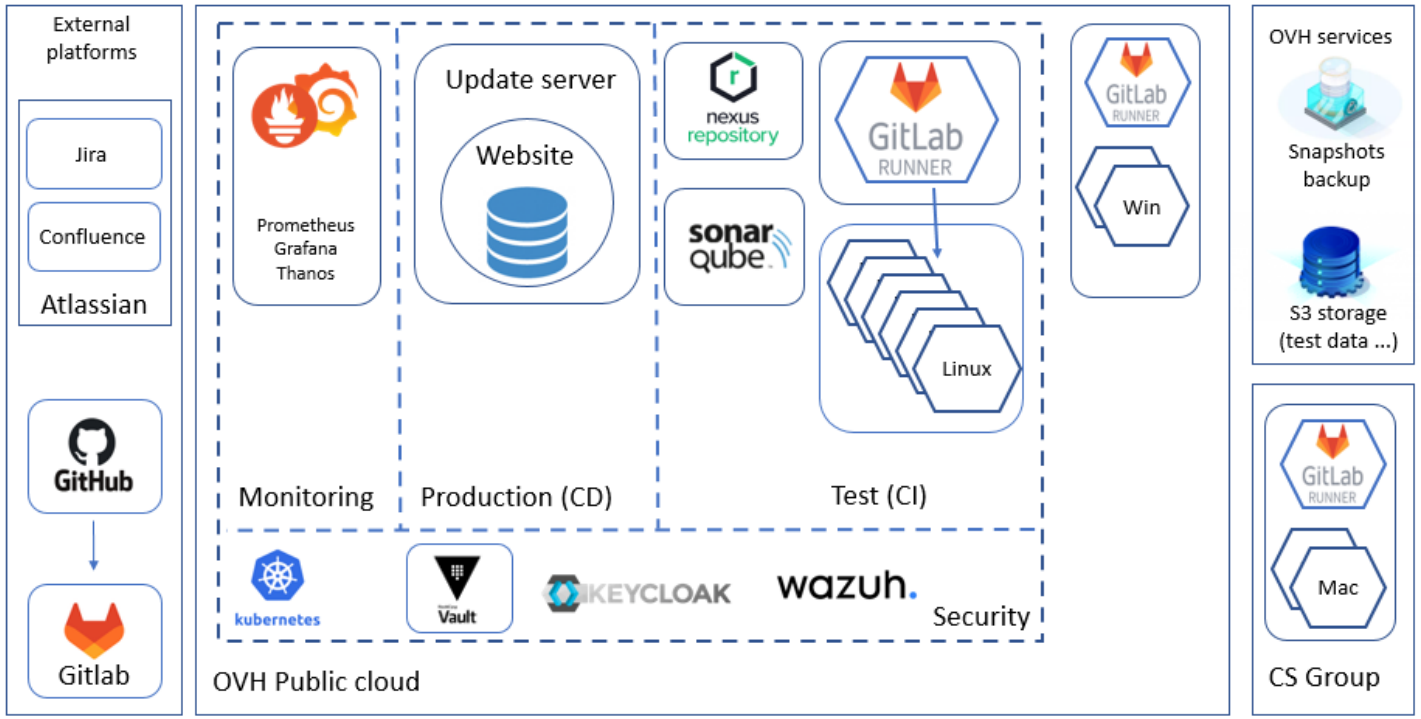
SNAP is made up by modules. The whole SNAP source code is split into numerous modules.

Open-Source Developers can extend SNAP by writing their own module and use plug-in points to link with SNAP.

Toolboxes are just a set of modules which are distributed and installed all together. This makes it convenient for users to install the Sentinel Toolboxes. Toolboxes docking with their extensions at the extension points



Continuous Integration and Quality Assurance

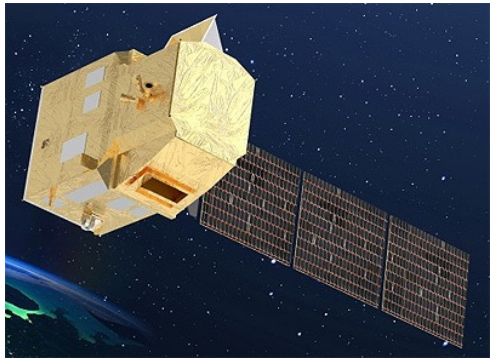


SNAP developers are using state-of-the art systems for continuous integration, fully automated tests with large test coverage, tools for system monitoring, and a security layer including secured storage of credentials, AntiVirus & AntiMalware. SNAP ecosystem is fully GDPR compliant.

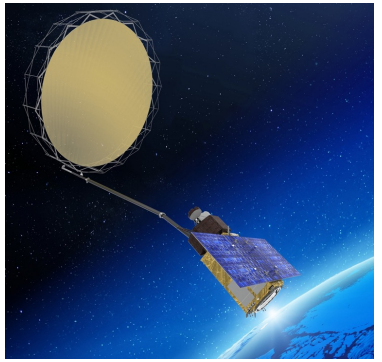
SNAP Roadmap

- **06/2023 – SNAP 10**
  - Optical and Microwave Toolboxes
  - Large software renovation
  - Making it technologically future proof
- **12/2023 – SNAP 11**
  - Product Groups
  - STAC support (Spatio-Temporal Asset Catalogue)
  - Preparing for hyperspectral CHIME and microwave CIMR
- **06/2024 – SNAP 12**
  - Change detection Toolbox
  - S1 ETAD improvements
  - Generic could detection operators
- **12/2024 – SNAP 13**
  - Support NISAR, BIOMASS
  - Time series tools
  - Optical and hyperspectral synergistic tools

Optical Toolbox
with future CHIME support

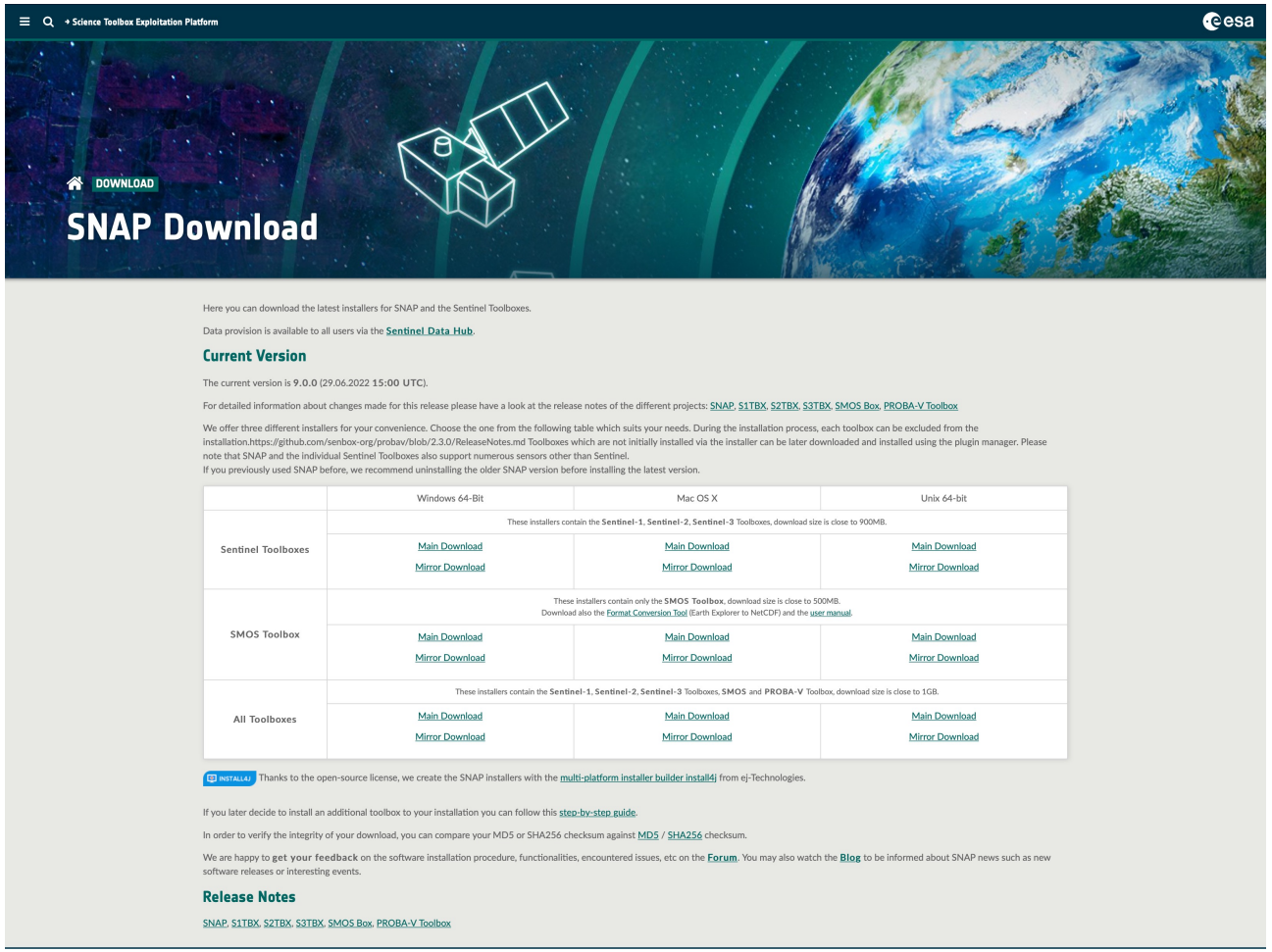


Microwave Toolbox
with future CIMR support

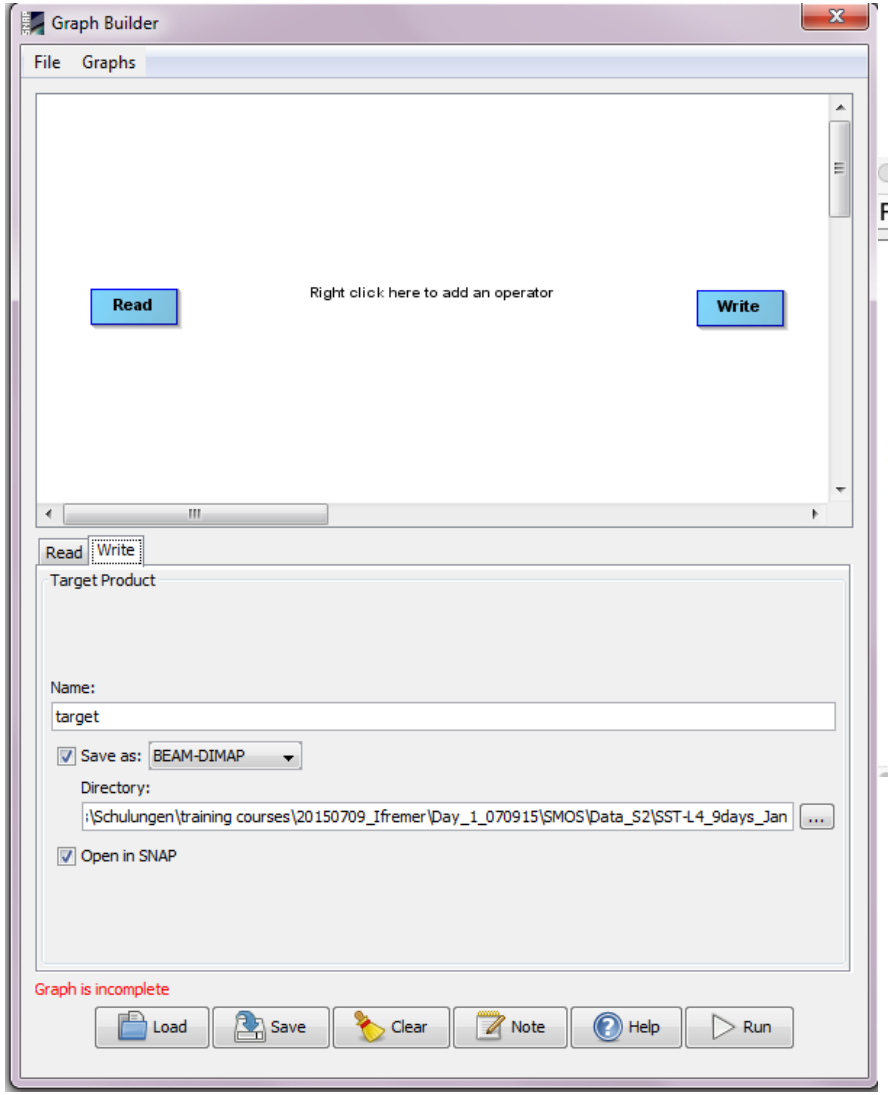


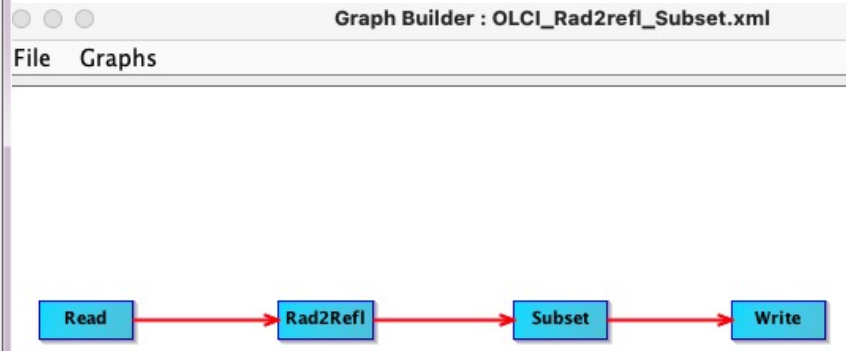
Download SNAP

https://step.esa.int/main/download/snap-download/



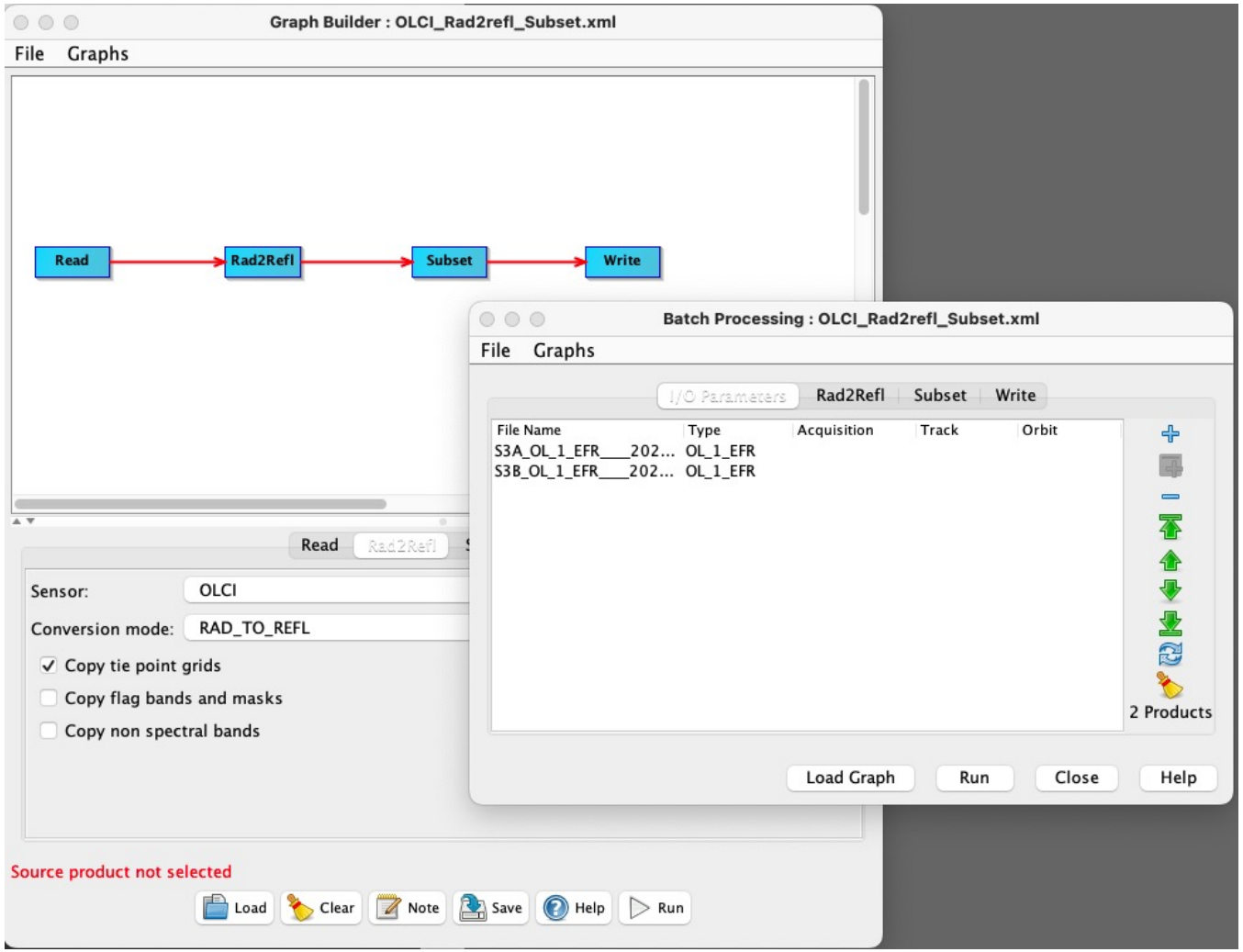
Batch processing: Graph Builder





Not all processors are available on Graph Builder

Batch processing: Graph Builder



- The Batch Processing tool available via the DAT allows you to execute a single reader/writer graph for a set of products.
- Select the Batch Processing tool from the Graphs menu and then press the "Load" button to browse for a previously saved graph.
- Next, add products in the IO tab by pressing the "Add" button or dragging and dropping a ProductSet or Products from the Project or Products views.
- Set the target folder where the output will be written to and then press "Run".

SNAP Graph Processing Tool (GPT): batch processing

- Use SNAP‘s command-line tools:
  - From a command-line shell (v1)
  - From shell scripts (v2)
  - From Python, IDL, MatLab…using dedicated systems (v3)
- Many advantages

✓ No intermediate files written, no I/O overhead

✓ Reusability of processing chains

✓ Simple and comprehensive operator configuration

✓ Reusability of operator configurations

SNAP GPT: Command line tools

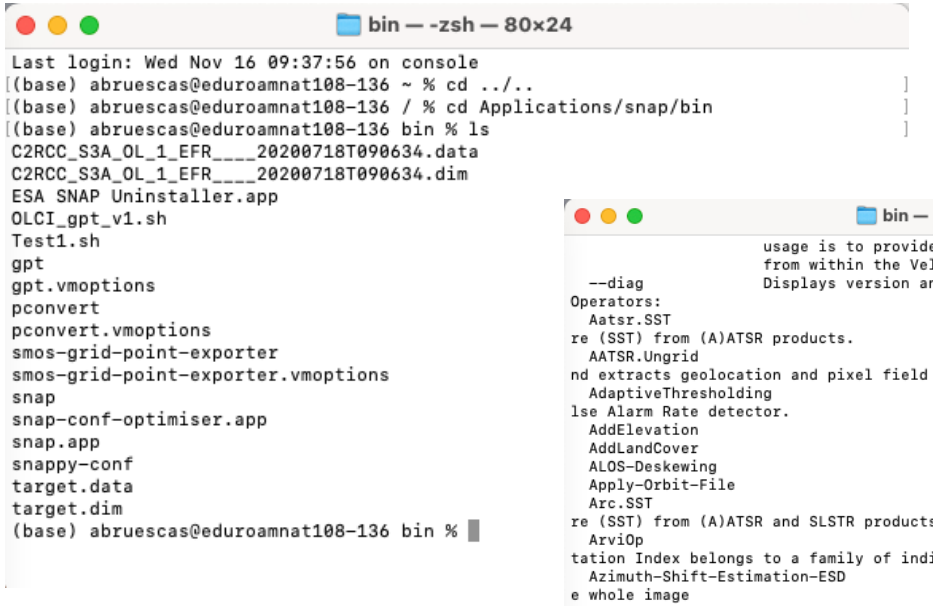
- Look into ${SNAP-HOME}/bin directory:

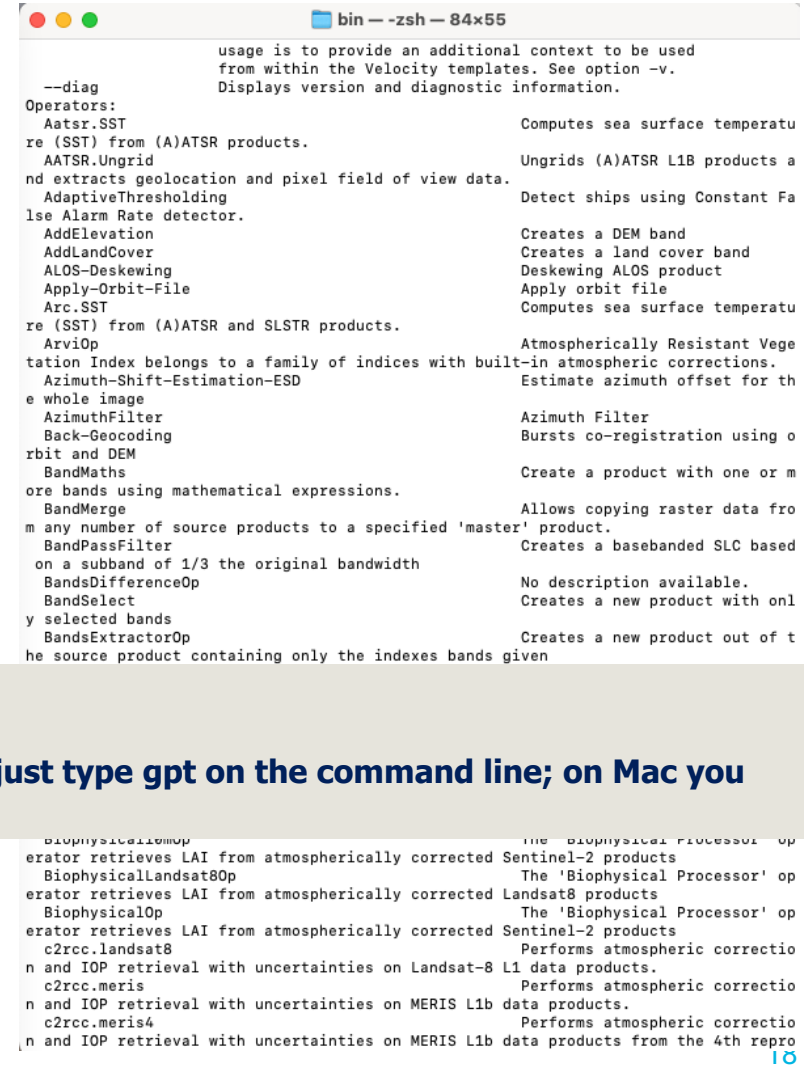
gpt (.exe in Windows)→used to execute various SNAP operators and chain of operators

- ESA SNAP/SNAP command line help>gpt –h

- pconvert→ used to convert product files into other data and image formats (quick-look generation)

- snappy-conf→ SNAP application configuration launcher for snap-py





Go to your SNAP folder and look
Call gpt
(if only one SNAP instance on your computer, just type gpt on the command line; on Mac you need to type the full path in your terminal)

SNAP GPT: Command line tools

- Most important SNAP batch-mode tool
- Usage:
  - *gpt <op>|<graph-file> [options] [<source-file-1> <source-file-2> ...]*
- Which operator are available?
  - *gpt –h or ./gpt -h*
- List of operators may vary depending on installed SNAP plug-ins

Call gpt from command line

S2Resampling_subset_c2rcc_msi.xml

Users > abruescas > Dropbox > Mac > Desktop > ESA_Academy_final > Hands_on > S2DATA > S2Resampling_subset_c2rcc_msi.xml

```
<!-- > ./gpt /Users/yourpathtoxml/S2Resampling_subset_c2rcc_msi.xml -Ssource=/Users/yourpathtoinputimage/S2B_MS
-->
<graph id="Graph">
  <version>1.0</version>
    <node id="S2Resampling">
      <operator>S2Resampling</operator>
      <sources>
        <sourceProduct>${source}</sourceProduct>
      </sources>
        <parameters>
    <resolution>60</resolution>
    <upsampling>Bilinear</upsampling>
    <downsampling>Mean</downsampling>
    <flagDownsampling>First</flagDownsampling>
    <resampleOnPyramidLevels>true</resampleOnPyramidLevels>
</parameters>
    </node>

  <node id="Subset">
    <operator>Subset</operator>
    <sources>
      <source>S2resampling</source>
```

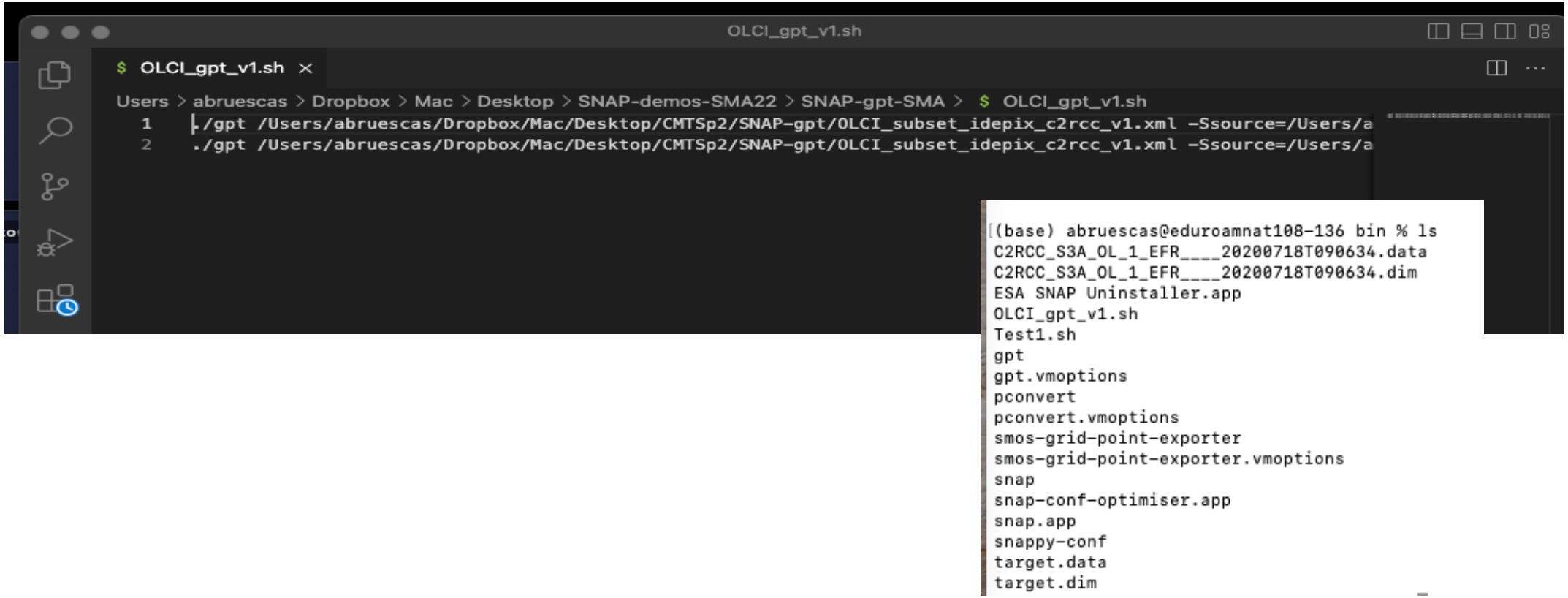
Mac/Linux

```
./gpt /Users/yourpathtoxml/S2Resampling_subset_c2rcc_msi.xml -
Ssource=/Users/yourpathtoinputimage/S2B_MSIL1C_20220818T095549_N0400_R122_T33UXS_20220818T104434.SAFE -t
/Users/Users/yourpathtooutputimage/S2B_MSIL1C_20220818T095549_N0400_R122_T33UXS_20220818T104434_res_sub_c2rcc.dim
```

```
      <fullSwath>false</fullSwath>
      <tiePointGridNames/>
      <copyMetadata>true</copyMetadata>
    </parameters>
  </node>

<node id="c2rcc">
      <operator>c2rcc.msi</operator>
      <sources>
        <sourceProduct>Subset</sourceProduct>
      </sources>
    <parameters>
    <validPixelExpression>B8 > 0</validPixelExpression>
    <salinity>35.0</salinity>
    <temperature>15.0</temperature>
```

Batch mode: Using gpt XML with shell (v1)

- Running on several images (batch mode): use Excel or other spreadsheet/editor for compiling a list with the names of all products to be processed. Then save as .sh (linux, mac) or .bat (windows).
- For running the script, open a terminal and give the order: OLCI_gpt_v1.sh



Batch mode: Using gpt XML with shell (v2)

OLCI_subset_idepix_c2rcc.xml

```
<graph id="Graph">
  <version>1.0</version>
<node id="Read">
      <operator>Read</operator>
      <sources/>
      <parameters>
        <file>/Users/abruescas/Dropbox/Mac/Desktop/CMTSp2/SNAP-gpt/OLCI_gpt_v2/S3A_OL_1_EFR____20200725T0925
      </parameters>
    </node>

  <node id="Subset">
    <operator>Subset</operator>
    <sources>
      <source>Read</source>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <sourceBands>Oa01_radiance,Oa02_radiance,Oa03_radiance,Oa04_radiance,Oa05_radiance,Oa06_radiance,Oa07_
      <geoRegion>POLYGON((15.836 54.278, 15.836 59.178, 21.103 59.178, 21.103 54.278, 15.836 54.278))</geoRe
      <subSamplingX>1</subSamplingX>
      <subSamplingY>1</subSamplingY>
      <fullSwath>false</fullSwath>
      <tiePointGridNames/>
      <copyMetadata>true</copyMetadata>
    </parameters>
  </node>


<node id="Idepix.Olci">
      <operator>Idepix.Olci</operator>
      <sources>
        <sourceProduct>Subset</sourceProduct>
      </sources>
<parameters>
    <outputSchillerNNValue>false</outputSchillerNNValue>
    <computeMountainShadow>true</computeMountainShadow>
    <mntShadowExtent>0.9</mntShadowExtent>
    <computeCloudShadow>true</computeCloudShadow>
    <outputCtp>false</outputCtp>
    <computeCloudBuffer>true</computeCloudBuffer>
    <cloudBufferWidth>2</cloudBufferWidth>
    <useSrtmLandWaterMask>false</useSrtmLandWaterMask>
</parameters>
</node>

<node id="c2rcc">
      <operator>c2rcc.olci</operator>
      <sources>
        <sourceProduct>Subset</sourceProduct>
      </sources>
    <parameters>
        <validPixelExpression>!quality_flags.invalid and (!quality_flags.land or quality_flags.fresh_inland_
```

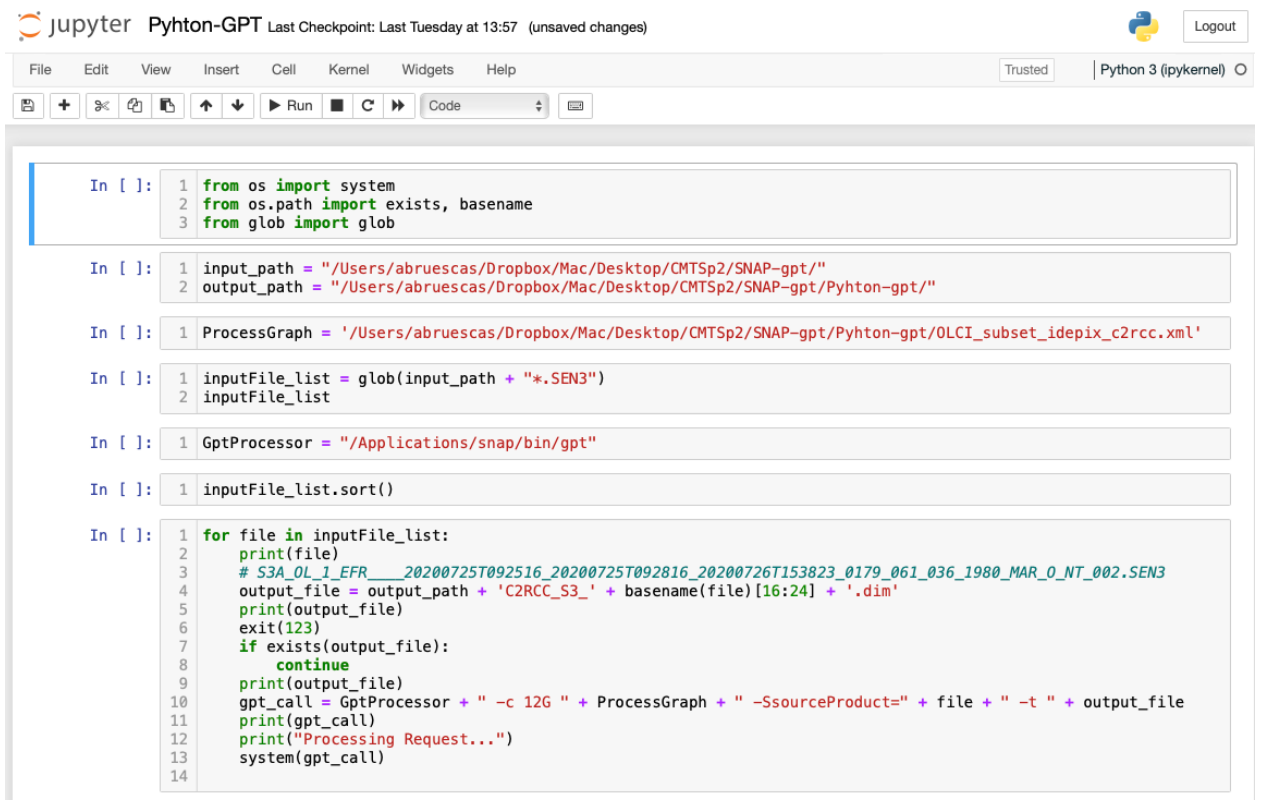
OLCI_gpt_v2.sh

```
/bash

folder=/yourdirectory/OLCI_gpt_v2/
rt=C2RCC_S3_
.dim

 $(ls -d -a /yourdirectory/*.SEN3)


product_basename=$(basename $i)
ac_date=$(echo $product_basename | cut -d '_' -f 8)
Output_pathname=$output_folder$nameStart$ac_date$nameEnd
/Applications/snap/bin/gpt OLCI_subset_idepix_c2rcc.xml -Source=$i -t $Output_pathname
```

SNAP GPT with Python (v3)

- OLCI_CMTS_processing.py
  or
- Python-GPT.ipynb

(OLCI_subset_idepix_c2rcc.xml)



jupyter Pyhton-GPT Last Checkpoint: Last Tuesday at 13:57 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [ ]:
```
from os import system
from os.path import exists, basename
from glob import glob
```

In [ ]:
```
input_path = "/Users/abruescas/Dropbox/Mac/Desktop/CMTSp2/SNAP-gpt/"
output_path = "/Users/abruescas/Dropbox/Mac/Desktop/CMTSp2/SNAP-gpt/Pyhton-gpt/"
```

In [ ]:
```
ProcessGraph = '/Users/abruescas/Dropbox/Mac/Desktop/CMTSp2/SNAP-gpt/Pyhton-gpt/OLCI_subset_idepix_c2rcc.xml'
```

In [ ]:
```
inputFile_list = glob(input_path + "*.SEN3")
inputFile_list
```

In [ ]:
```
GptProcessor = "/Applications/snap/bin/gpt"
```

In [ ]:
```
inputFile_list.sort()
```

In [ ]:
```
for file in inputFile_list:
    print(file)
    # S3A_OL_1_EFR____20200725T092516_20200725T092816_20200726T153823_0179_061_036_1980_MAR_O_NT_002.SEN3
    output_file = output_path + 'C2RCC_S3_' + basename(file)[16:24] + '.dim'
    print(output_file)
    exit(123)
    if exists(output_file):
        continue
    print(output_file)
    gpt_call = GptProcessor + " -c 12G " + ProcessGraph + " -SsourceProduct=" + file + " -t " + output_file
    print(gpt_call)
    print("Processing Request...")
    system(gpt_call)
```

https://senbox.atlassian.net/wiki/spaces/SNAP/pages/70503475/Bulk+Processing+with+GPT

SNAP / User Guide / Processing

Bulk Processing with GPT

Created by Marco Peters
Last updated: 2021-03-26 • 4 min read

Bulk Processing with GPT

This little tutorial gives an introduction on bulk processing with the command shell on Windows and Unix systems. The provided scripts try to stay very generic in order to serve multiple processing requirements. However, not every edge case can be covered. The intention is to cover at least the main use cases. The scripts can probably be improved at multiple points but they can give you a starting point to write your own scripts. If you know improvements to the scripts or have questions regarding the usage of the script you are kindly invited to the SNAP Forum.
A general introduction to GPT and graphs can be found at Creating a GPF Graph.

The four files mentioned below are attached for download.

- processDataset.bat (Windows)
- processDataset.bash (Unix)
- resample_s2.xml
- resample_20m.properties

Table of Contents

Add inline comme

SNAP with Python using SNAPPY

- Bridge between SNAP/Java and Python
- Allows invocation of SNAP functionality from Python and vice versa
- Data provided by SNAP can be used with e.g., numpy, matplotlib, etc.
- Processing operators can be implemented in Python
- GPF processing graphs can be created and executed with Python if preferred over command line

SNAPPY Examples


```
from snappy import ProductIO
import numpy as np
import matplotlib.pyplot as plt

p = ProductIO.readProduct('snappy/testdata/MER_FRS_L1B_SUBSET.dim')
rad13 = p.getBand('radiance_13')
w = rad13.getRasterWidth()
h = rad13.getRasterHeight()
rad13_data = np.zeros(w * h, np.float32)
rad13.readPixels(0, 0, w, h, rad13_data)
p.dispose()
rad13_data.shape = h, w
imgplot = plt.imshow(rad13_data)
imgplot.write_png('radiance_13.png')
```

1. Product is read using SNAP
2. Getting a band

3. Reading the data of the band
4. Dispose product (clean memory)
5. Re-shape the data array
6. Show image
7. Write as png

SNAPPY Examples


```
from snappy import ProductIO
from snappy import GPF
from snappy import Hashmap

p = ProductIO.readProduct('eo/S2A_MSIL1C_20201229<...>T144625.zip')
parameters = Hashmap()
parameters.put('resolution', '10')
parameters.put('upsampling', 'Nearest')
parameters.put('downsampling', 'Mean')

target_product = GPF.createProduct('S2Resampling', parameters, p)

ProductIO.writeProduct(target_product , <'your/out/directory'>, 'BEAM-DIMAP')
```

1. Read the input product
2. Provide processing parameters
3. Call the operator ‘Resample’
4. Write the processed product

SNAPISTA

- Based on SNAPPY
- Dedicated to create and run GPF graphs in a pythonic way
- Provided as conda package for Linux
- On Win and Mac a docker container needs to be used

- Currently still an external tool, but will be integrated into the SNAP ecosystem in the future

SNAPISTA Example


```
from snapista import Graph
from snapista import Operator, OperatorParams

g = Graph()

productPath = 'path to S1 file'

g.add_node(
    operator=Operator(
        "Read",
        formatName="SENTINEL-1",
        file=productPath,
    ),
    node_id="read",
)

g.add_node(
        operator=Operator("Apply-Orbit-File", continueOnFail="true"),
        node_id="apply-orbit-file",
        source="read",
)

g.add_node(
    operator=Operator(
        "Remove-GRD-Border-Noise", borderLimit="2000", trimThreshold="0.2"
    ),
    node_id="noise-removal",
    source="apply-orbit-file",
)
```

1. Add a read operation (node)

2. Add several operations more

SNAPISTA Example


```
g.add_node(
    operator=Operator("Calibration"),
    node_id="calibration",
    source="noise-removal",
)

g.add_node(
    operator=Operator("LinearToFromdB"),
    node_id="linear",
    source="calibration",
)

g.add_node(
    operator=Operator(
        "Terrain-Correction",
        pixelSpacingInMeter="20.0",
        demName="SRTM 3Sec",
        mapProjection="AUTO:42001",
        ),
        node_id="terrain-correction",
        source="linear",
)

g.add_node(
    operator=Operator("Write", file='result.tif', formatName="GeoTIFF-BigTIFF"),
    node_id="write",
    source="terrain-correction",
)

g.run()
```

3. Add a write operation
4. Finally execute the graph

Further Reading (snappy)

More examples and how to configure snappy

- Configure Python to use the snappy
  https://senbox.atlassian.net/wiki/x/BQAIAw
- How to use the SNAP API from Python
  https://senbox.atlassian.net/wiki/x/CoAmAQ
- What to consider when writing an Operator in Python
  https://senbox.atlassian.net/wiki/x/AoCBAg

More examples and how to configure SNAPISTA

- Installation Guide
  https://snap-contrib.github.io/snapista/installation.html
- Getting Started
  https://snap-contrib.github.io/snapista/gettingstarted.html
- More examples
  https://snap-contrib.github.io/snapista/examples/operator.html

Thank you!

Questions are welcome.