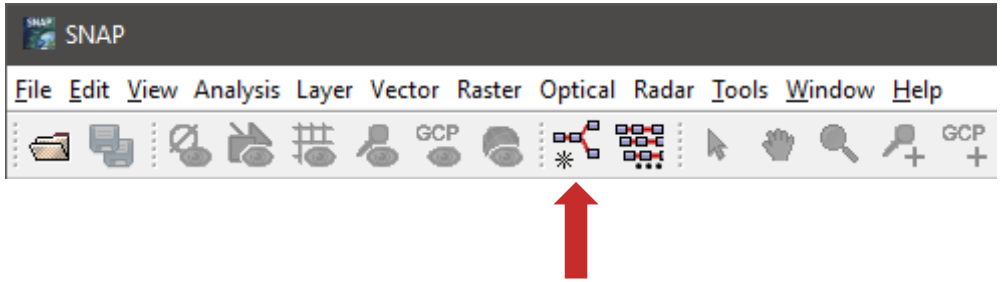
PYTHON SNAP GPF CONVERTER

Group Meeting
10.03.2022

S N A P

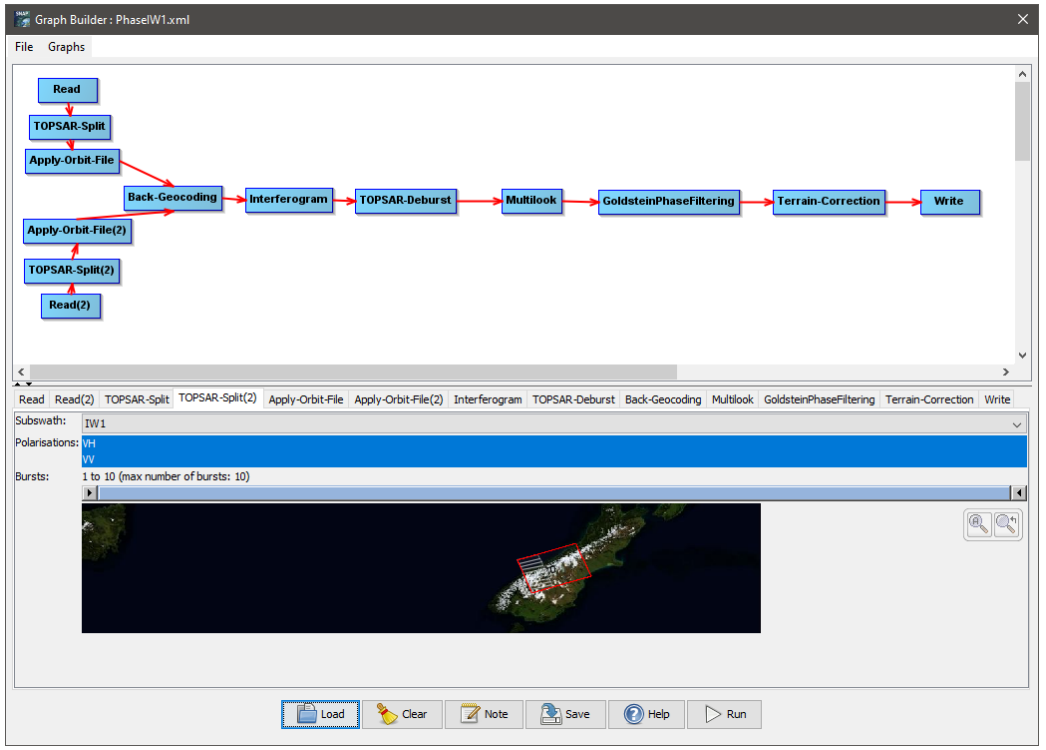
- Sentinel Application Platform
- SAR & Optical Processing
- No license necessary
- Website

- Graph Builder:



GRAPH BUILDER

- Default processing workflows
  - DInSAR
  - Oil Spill Detection
  - Ship Detection
  - …
- Individual adaptations
- Saved as .xml file



S N A P P Y

- Python 2.7, 3.3 to 3.6
- Installation with SNAP
- [ESA SNAPPY Tutorial]

```
# 1. Imports

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

# 2. Fill parameter Hashmap

parameters = Hashmap()
parameters.put('targetResolution', '10')
parameters.put('referenceBand', 'B4')

# 3. Call Operator

operator_name = 'Resample'
target_product = GPF.createProduct(operator_name, parameters, p)

# 4. Write Product
# Write target Product with ProductIO:

write_format = 'BEAM-DIMAP' # in this case write as BEAM-DIMAP
ProductIO.writeProduct(target_product , <'your/out/directory'>, write_format)

# Alternative solution: Computations are faster when using GPF to write the product instead of ProductIO:
incremental = false # most writer don't support the incremental writing mode (update exsiting file), except BEAM-DIMAP.
GPF.writeProduct(target_product , File(<'your/out/directory'>), write_format, incremental, ProgressMonitor.NULL)
```

GPF-XML TO PYTHON DICTIONARY

- All information available
  - Node (name)
  - Operator (SNAP function)
  - Parameters
  - Source product
    - Multiple possible

```
<graph id="Graph">
  <version>1.0</version>
  <node id="Read">
    <operator>Read</operator>
    <sources/>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <file>J:\_PhD\_Data\ASF_Download\The_Alpine_Gardens_landslide\Ascending\S1B_IW_SLC...
    </parameters>
  </node>
  <node id="Read(2)">
    <operator>Read</operator>
    <sources/>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <file>J:\_PhD\_Data\ASF_Download\The_Alpine_Gardens_landslide\Ascending\S1B_IW_SLC...
    </parameters>
  </node>
  <node id="TOPSAR-Split">
    <operator>TOPSAR-Split</operator>
    <sources>
      <sourceProduct refid="Read"/>
    </sources>
    <parameters class="com.bc.ceres.binding.dom.XppDomElement">
      <subswath>IW1</subswath>
      <selectedPolarisations>VH,VV</selectedPolarisations>
      <firstBurstIndex>1</firstBurstIndex>
      <lastBurstIndex>10</lastBurstIndex>
      <wktAoi/>
    </parameters>
  </node>
```

GPF-XML TO PYTHON DICTIONARY

- All information available
  - Node (name)
  - Operator (SNAP function)
  - Parameters
  - Source product
    - Multiple possible

- Python dictionary
  - Node + nr
  - Operator
  - Parameter dict()
  - Source product dict()
  - Next task in 2nd loop

```
for line in linesCorr:
    if '<node ' in line:
        seg     = line.split('"')[1]
        if '(' in seg:
            nr  = seg.split('(')[1]
            nr  = nr.replace(')','')
            seg = seg.split('(')[0]
        else:
            nr = '1'

        tasks[seg+nr]               = dict()
        tasks[seg+nr]['name']       = seg
        tasks[seg+nr]['nr']         = nr
        tasks[seg+nr]['operator']   = ''
        tasks[seg+nr]['parameters'] = dict()
        tasks[seg+nr]['sources']    = dict()
        tasks[seg+nr]['nextTask']   = dict()
        tasks[seg+nr]['status']     = 'pending'
        tasks[seg+nr]['image']      = ''

    elif '<operator>' in line:
        op                          = line.split('>')[1]
        op                          = op.split('<')[0]
        tasks[seg+nr]['operator']   = op

    elif '<parameters ' in line:
        par     = True

    elif '</parameters>' in line:
        par     = False

    elif par == True:
        l       = line.split('</')[0]
        l       = l.replace('<','')
        n,v     = l.split('>')[0:2]
        if n[-1] != '/' and '&quot' not in v:
            tasks[seg+nr]['parameters'][n]  = v

    elif '<sources>' in line:
        sor     = True
    elif '</sources>' in line:
        sor     = False
    elif sor == True:
        if 'sourceProduct' in line:
            segS     = line.split('"')[1]
            if '(' in segS:
                nrS  = segS.split('(')[1]
                nrS  = nrS.replace(')','')
                segS = segS.split('(')[0]
            else:
                nrS = '1'

            tasks[seg+nr]['sources'][segS+nrS] = {'name':segS,'nr':nrS}

    elif 'Presentation' in line:
        break
```

SNAPtask - Class

Setting individual tasks up

```
class SNAPtask:
    def __init__(self,name,nr,operator,parameters):
        self.name       = name
        self.nr         = nr
        self.parameters = parameters
        self.operator   = operator

        self.hashMap    = ''
        self.status     = 'pending'
        self.image      = ''

        self.nextTask   = list()
        self.sources    = list()
```

Creating parameter hashmaps

```
    def __createHashmap(self):
        self.hashMap = HashMap()

        for key in self.parameters.keys():
            self.hashMap.put(key,self.parameters[key])
```

Run individual task

```
    def runTask(self):
        self.__createHashmap()
        bands = ''

        print('     Checking: '+self.name+' - '+self.nr)
        while any([s.status == 'pending' for s in self.sources]):
            for s in self.sources:
                if s.status == 'pending':
                    s.runTask()

        if 'sourceBands' in self.parameters.keys():
            Names   = list(self.sources[0].image.getBandNames())
            bands   = []
            bands   += filter(lambda x: x.startswith('Amp'), Names)
            bands   += filter(lambda x: x.startswith('Phase'), Names)
            bands   += filter(lambda x: x.startswith('coh'), Names)
            bands   += filter(lambda x: x.startswith('Int'), Names)
            #bands   += filter(lambda x: x.startswith('i'), Names)
            #bands   += filter(lambda x: x.startswith('q'), Names)
            bands   += filter(lambda x: x.endswith('VV'), Names)
            bands   += filter(lambda x: x.endswith('VH'), Names)
            bands   += filter(lambda x: x.endswith('HH'), Names)
            bands   += filter(lambda x: x.endswith('DEM'), Names)
            bands   += filter(lambda x: x.endswith('elevation'), Names)
            bands   += filter(lambda x: x.startswith('lay'), Names)

            string  = ''
            for idx in range(len(bands)):
                if idx == len(bands)-1:
                    string  += str(bands[idx])
                else:
                    string  += str(bands[idx]) + ','
            print('SourceBands: ' + string)
            self.hashMap.put('sourceBands',string)


        if self.name != 'Read' and self.name != 'Write':
            print('Running: '+self.name+' - '+self.nr)
            self.image  = GPF.createProduct(self.operator,self.hashMap,[s.image for s in self.sources])
        elif self.name == 'Read':
            print('Reading: '+self.parameters['file'])
            self.image  = ProductIO.readProduct(self.parameters['file'])
        elif self.name == 'Write':
            print('Writing '+self.parameters['formatName']+' File: '+self.parameters['file'])
            ProductIO.writeProduct(self.sources[0].image, self.parameters['file'],self.parameters['formatName'])
        self.status     = 'finished'
```

Recursive running of source tasks

Filtering of source bands
to do: external interaction

3 types of SNAP function

Additional:

- Create individual task from dictionary
- Change individual parts of dictionary
  - Ideal for repetitive processing

```
def createTasksDict(tasks):
    tDict = dict()
    for task in tasks.keys():
        tDict[task] = SNAPtask(tasks[task]['name'],tasks[task]['nr'],tasks[task]['operator'],tasks[task]['parameters'])
    for t in tasks.keys():
        try:
            tDict[t].sources = [tDict[s] for s in tasks[t]['sources'].keys()]
        except:
            pass
        try:
            tDict[t].nextTask = [tDict[n] for n in tasks[t]['nextTask'].keys()]
        except:
            pass

    return tDict

def changeTasks(cmdIn,tasks):
    cmdIn = cmdIn[1:]
    if len(cmdIn) > 1:
        if (len(cmdIn[1:])%3)==0:
            for i in range(int(len(cmdIn[1:])/3)):
                print(cmdIn[i*3+1])
                if cmdIn[i*3+1][-1] in [str(i) for i in range(10)]:
                    try:
                        print(cmdIn[i*3+1]+' - '+cmdIn[i*3+2]+':\n'+ tasks[cmdIn[i*3+1]]['parameters'][cmdIn[i*3+2]])
                        print('Changed to:\n'+ cmdIn[i*3+3])
                        tasks[cmdIn[i*3+1]]['parameters'][cmdIn[i*3+2]] = cmdIn[i*3+3]
                    except:
                        print('Task - Parameter - Value not found.')
                else:
                    for key in tasks.keys():
                        if cmdIn[i*3+1] in key:
                            try:
                                print(key+' - '+cmdIn[i*3+2]+':\n'+ tasks[key]['parameters'][cmdIn[i*3+2]])
                                print('Changed to:\n'+ cmdIn[i*3+3])
                                tasks[key]['parameters'][cmdIn[i*3+2]] = cmdIn[i*3+3]
                            except:
                                print('Task - Parameter - Value not found.')

    return tasks
```

Running with command line:

- Former problems with accumulating memory

```
# ## Import:
# ----------------------------------------------------------------------
import os
from subprocess import PIPE, call, Popen



# ## Code Example:
xml     = 'J:\\_PhD\\_Processing\\_XML\\PhaseTest.xml'

read2   = "J:\_PhD\_Data\ASF_Download\Kaikoura2016\S1B_IW_SLC__1SSV_20161128T071352_20161128T071422_003155_0055E7_EDF9.zip"
read1   = "J:\_PhD\_Data\ASF_Download\Kaikoura2016\S1B_IW_SLC__1SDV_20161116T071350_20161116T071418_002980_005109_2DC8.zip"

for IW in ['IW1']:#,'IW2','IW3'
    output  = 'D:\\Documents\\Processing\\Kaikora\\Test'+IW+'after.dim'
    cmd     = ['D:\\Programme\\Python\\python.exe',
               'J:\\_PhD\\_Python\\_up2dateVersions\\SNAP\\xml2SNAP.py',
               xml,
               'Read1','file',read1,
               'Read2','file',read2,
               'TOPSAR-Split','subswath',IW,
               'TOPSAR-Split','selectedPolarisations','VV',
               'Write1','file',output]
    print(Popen(cmd, stdout=PIPE, stderr=PIPE).communicate()[0].decode('utf-8').strip())
```

GitLab:

- Gitlab Project ID: 31592
- https://gitlab.ethz.ch/luckm/snaptask-remotesensing
- luck@ifu.baug.ethz.ch, manuel.luck@gmail.com

You're very welcome to try it out and give feedback if you want something adapted!