Author: **Antonio Vecoli**

Date: **06/06/2017**

Tech For Space www.techforspace.com (https://www.techforspace.com/)

For any technical or Python support please refer to our **Project Page** (https://www.techforspace.com/project/sentinels-earth-observation/)

Handling a Sentinel-2 product with SNAP in Python (Tutorial 2)

After the basic elements of SNAP explained in the first two tutorials, it is now possible to introduce a set of more advanced operations that allow to modify a Sentinel-2 data product and make it available for specific scientific processings. The Sentinel-2 product used in this case will be the same of Tutorial 1 and it can be downloaded(with personal account) at the following link :

https://scihub.copernicus.eu/dhus/odata/v1/Products('c94ebae2-3b0d-4472-96a0-324bb54d7bdf')/$value (https://scihub.copernicus.eu/dhus/odata/v1/Products('c94ebae2-3b0d-4472-96a0-324bb54d7bdf')/$value)

Resampling a Sentinel-2 data product

In the first tutorial a single band image has been extracted from a complete Sentinel-2 product without implementing any scientific analysis on it, because the aim was to show how it is possible to read an S-2 product in Python, also suggesting some simple image processing technique for a better visualization. But in general, when working with multispectral data, several techniques of scientific analysis need to consider more than one band at the same time. In this case, for all the selected bands, their rasters should be available with the same spatial resolution, so that all the images and data arrays will have the same size in term of number of pixels and matrix dimensions. Let's consider a simple comparison between two different bands of the current S-2 product:

In [3]:

```
import snappy
from snappy import ProductIO
file_path = 'C:\Program Files\snap\S2A_MSIL1C_20170202T090201_N0204_R007_T35SNA_20170202T09
product = ProductIO.readProduct(file_path)
list(product.getBandNames())
```

Out[3]:

```
['B1',
 'B2',
 'B3',
 'B4',
 'B5',
 'B6',
 'B7',
 'B8',
 'B8A',
 'B9',
 'B10',
 'B11',
 'B12',
 'view_zenith_mean',
 'view_azimuth_mean',
 'sun_zenith',
 'sun_azimuth',
 'view_zenith_B1',
 'view_azimuth_B1',
 'view_zenith_B2',
 'view_azimuth_B2',
 'view_zenith_B3',
 'view_azimuth_B3',
 'view_zenith_B4',
 'view_azimuth_B4',
 'view_zenith_B5',
 'view_azimuth_B5',
 'view_zenith_B6',
 'view_azimuth_B6',
 'view_zenith_B7',
 'view_azimuth_B7',
 'view_zenith_B8',
 'view_azimuth_B8',
 'view_zenith_B8A',
 'view_azimuth_B8A',
 'view_zenith_B9',
 'view_azimuth_B9',
 'view_zenith_B10',
 'view_azimuth_B10',
 'view_zenith_B11',
 'view_azimuth_B11',
 'view_zenith_B12',
 'view_azimuth_B12']
```

According to the S-2 data product specifics, band 4 and band 5 are represented with rasters of different sizes and that can be easily verified :

In [4]:

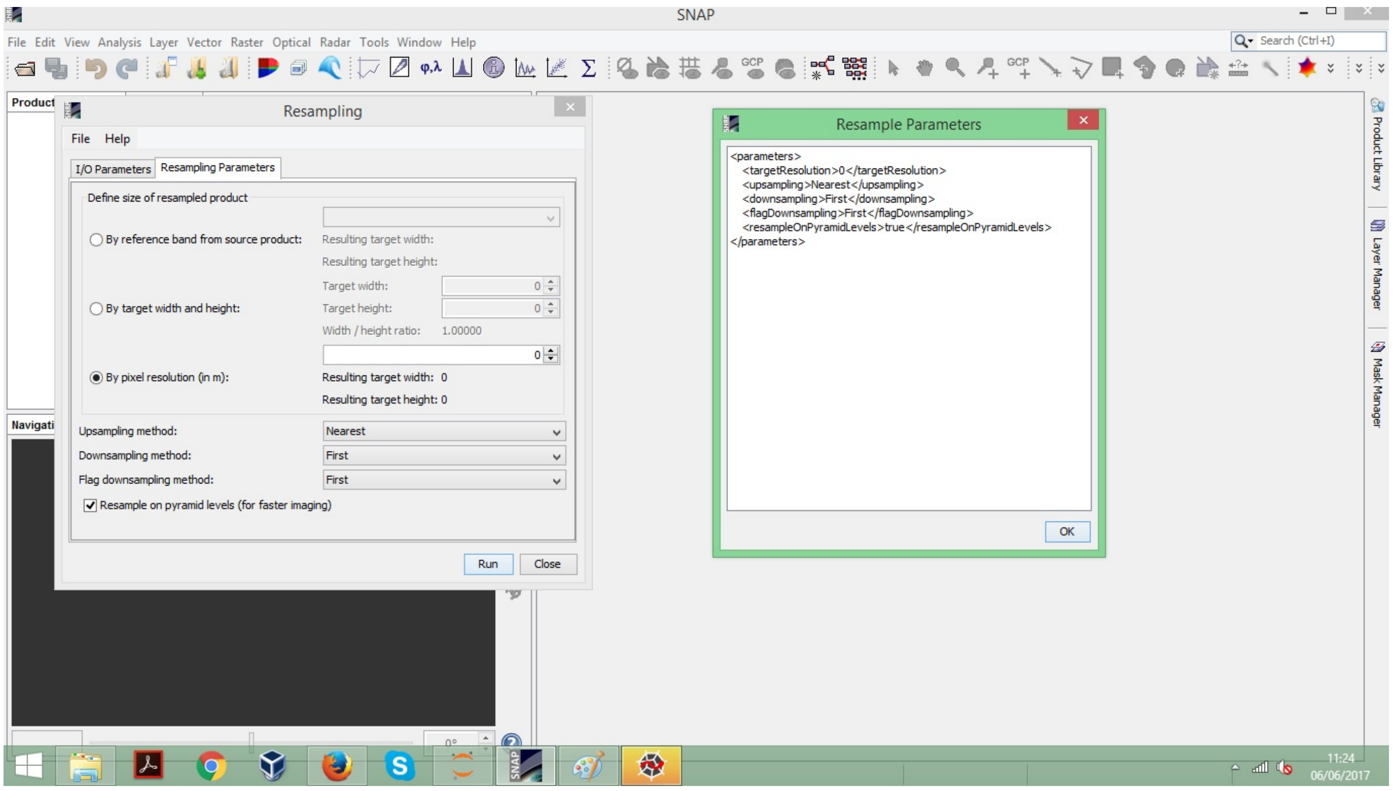
```
B4 = product.getBand('B4')
B5 = product.getBand('B5')

Width_4 = B4.getRasterWidth()
Height_4 = B4.getRasterHeight()
print("Band 4 Size: " + str(Width_4) +','+ str(Height_4))

Width_5 = B5.getRasterWidth()
Height_5 = B5.getRasterHeight()
print("Band 5 Size: " + str(Width_5) +','+ str(Height_5))
```

```
Band 4 Size: 10980,10980
Band 5 Size: 5490,5490
```

The obtained results confirm that the two bands have been detected with different resolutions and if the user wanted to implement some processing that involves those bands, he should first operate a **RESAMPLING** of the S-2 product, according to a selected pixel resolution. The resampling operation can be directly executed in SNAP because it is included in the SNAP **Graph Processing Framework(GPF)**, a wide collection of data processors that can be applied to a Sentinel data product. Each data processor is called a **GPF operator** and it can be invoked in the desktop version of SNAP, in Python with the **snappy** module, or directly in the Windows/Linux command-line. The resampling operation is a typical example of a GPF operator because it is provided with a dedicated user interface that is available in the desktop version of SNAP. It is really important to look at the input parameters that must be set when the user wants to invoke a specific GPF operator. As for most of the GPF operators, also for the RESAMPLE operator the list of input parameters can be found in its user interface, as follows:



Define size of resampled product

By reference band from source product: Resulting target width: Resulting target height:

By target width and height: Target width: 0 Target height: 0 Width / height ratio: 1.00000

[✓ By pixel resolution (in m):] 0 Resulting target width: 0 Resulting target height: 0

Upsampling method: Nearest

Downsampling method: First

Flag downsampling method: First

✓ Resample on pyramid levels (for faster imaging)

```
<parameters>
  <targetResolution>0</targetResolution>
  <upsampling>Nearest</upsampling>
  <downsampling>First</downsampling>
  <flagDownsampling>First</flagDownsampling>
  <resampleOnPyramidLevels>true</resampleOnPyramidLevels>
</parameters>
```

The displayed list can change depending on the type of resampling the user wants to implement; the available options can be found in the desktop version of SNAP, looking into the user interface of the operator. In Python a GPF operator can be invoked only after the definition of the list of input parameters , using a Java

HashMap object (java.util.HashMap (https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html)). For this reason, whenever the user wants to work with a GPF operator he must always import that Java class in Python using the Jpy module:

In [7]:

```
from snappy import jpy
HashMap = snappy.jpy.get_type('java.util.HashMap')
```

It is then possible to construct an empty HasMap object and include the selected parameters with their values in it. In this simple case only the resolution parameter will be set.

In [8]:

```
parameters = HashMap()
parameters.put('targetResolution',20)
```

So the resolution in this specific case will be 20 meters per pixel. After the parameter definition it is possible to invoke the resampling operator using a syntax that is the same for all the GPF operators:

createProduct(String operatorName,Map(String,Object) parameters,Product sourceProduct)

and the Python implementation is given in the following line:

In [9]:

```
result = snappy.GPF.createProduct('Resample',parameters,product)
```

The output variable is a new data product and all its bands are now represented with the same resolution. As a test to confirm the successful operation it is possible to look again at band 4 and 5 to see what happened with them:

In [10]:

```
B4 = result.getBand('B4')
B5 = result.getBand('B5')

Width_4 = B4.getRasterWidth()
Height_4 = B4.getRasterHeight()
print("Band 4 Size: " + str(Width_4) +','+ str(Height_4))

Width_5 = B5.getRasterWidth()
Height_5 = B5.getRasterHeight()
print("Band 5 Size: " + str(Width_5) +','+ str(Height_5))
```

```
Band 4 Size: 5490,5490
Band 5 Size: 5490,5490
```

The user could try to visualize the image corresponding to a single band of the new product, explained in Tutorial 1, also verifying that all the band have definitely the same size. In many cases the resampling operation is just an initial step that can lead to various options. Infact the obtained product could be now written and saved as a new independent Sentinel product(with a specific format), or it could just be part of a more complex processing chain. Both of these options will be introduced in new tutorials that will widen the Python exploitation of the SNAP toolbox.