sentinelhub Documentation

Release 3.1.0

Sinergise EO research team

Oct 19, 2020

Contents:

1 Installation **3**

2 Configuration **5**
- 2.1 Sentinel Hub Capabilities . . . 5
- 2.2 Amazon S3 Capabilities . . . 5
- 2.3 Other . . . 6

3 Examples **7**
- 3.1 Sentinel Hub Processing API services from within Python . . . 7
  - 3.1.1 Prerequisites . . . 8
  - 3.1.2 Example 1: True color (PNG) on a specific date . . . 11
  - 3.1.3 Example 2: True color mosaic of least cloudy acquisitions . . . 16
  - 3.1.4 Example 3: All Sentinel-2's raw band values . . . 18
  - 3.1.5 Example 4: Save downloaded data to disk and read it from disk . . . 23
  - 3.1.6 Example 5: Other Data Collections . . . 24
  - 3.1.7 Example 6 : Multi-response request type . . . 27
  - 3.1.8 Example 7 : Raw dictionary request . . . 30
  - 3.1.9 Example 8 : Multiple timestamps data . . . 32
- 3.2 Data collections . . . 35
  - 3.2.1 Use an existing data collection definition . . . 35
  - 3.2.2 Define a new data collection . . . 40
- 3.3 Sentinel Hub OGC web services from within Python . . . 45
  - 3.3.1 Web Map Service (WMS) and Web Coverage Service (WCS) . . . 45
- 3.4 Large area utilities . . . 73
  - 3.4.1 Prerequisites . . . 73
  - 3.4.2 Area Splitting . . . 75
- 3.5 Sentinel Hub Batch Processing . . . 86
  - 3.5.1 1. Create a batch request . . . 87
  - 3.5.2 2. Analyse a batch request . . . 93
  - 3.5.3 3. Run a batch request job . . . 94
- 3.6 Sentinel Hub Feature Info Service (FIS) . . . 95
  - 3.6.1 Exploring basic statistics . . . 96
  - 3.6.2 Comparing histograms . . . 101
- 3.7 Accessing satellite data from AWS with Python . . . 103
  - 3.7.1 Searching for available data . . . 103
  - 3.7.2 Download data . . . 105
- 3.8 Downloading satellite data from AWS with command line . . . 108

3.8.1 Sentinel-2 products . . . 108
3.8.2 Sentinel-2 tiles . . . 109
3.8.3 .SAFE format details . . . 109

4 Package content **111**
4.1 Modules . . . 111
4.1.1 areas . . . 111
4.1.2 aws . . . 113
4.1.3 aws_safe . . . 118
4.1.4 config . . . 121
4.1.5 constants . . . 123
4.1.6 data_collections . . . 126
4.1.7 data_request . . . 128
4.1.8 download.aws_client . . . 140
4.1.9 download.client . . . 140
4.1.10 download.request . . . 142
4.1.11 download.sentinelhub_client . . . 144
4.1.12 fis . . . 144
4.1.13 geo_utils . . . 144
4.1.14 geometry . . . 147
4.1.15 geopedia . . . 151
4.1.16 io_utils . . . 154
4.1.17 ogc . . . 157
4.1.18 opensearch . . . 159
4.1.19 os_utils . . . 161
4.1.20 sentinelhub_batch . . . 162
4.1.21 sentinelhub_rate_limit . . . 166
4.1.22 sentinelhub_request . . . 167
4.1.23 sentinelhub_session . . . 169
4.1.24 testing_utils . . . 170
4.1.25 time_utils . . . 171

5 Indices and tables **173**

Python Module Index **175**

Index **177**

Documentation for Python utility packages developed by the EO research team at Sinergise.

This code allows one to reproduce some of the functionalities of the EO service Sentinel Hub. The code is available on GitHub.

CHAPTER 1

Installation

This package requires Python >=3.6 and can be installed with PyPI package manager:

```
$ pip install sentinelhub --upgrade
```

Alternatively, the package can be installed with Conda from *conda-forge* channel:

```
$ conda install -c conda-forge sentinelhub
```

In order to install the latest (development) version clone the GitHub repository and install:

```
$ pip install -e . --upgrade
```

or manually:

```
$ python setup.py build
$ python setup.py install
```

Before installing `sentinelhub-py` on **Windows** it is recommended to install package `shapely` from Unofficial Windows wheels repository (link).

CHAPTER 2

Configuration

The package contains a configuration file `config.json`. After the package is installed you can check the initial configuration parameters in command line:

```
$ sentinelhub.config --show
```

2.1 Sentinel Hub Capabilities

By default parameters `instance_id`, `sh_client_id` and `sh_client_secret` will be empty. In case you would like to use any capabilities of the package that interact with Sentinel Hub services you can set any of these parameters with:

```
$ sentinelhub.config --instance_id <your instance id>
$ sentinelhub.config --sh_client_id <your client id> --sh_client_secret <your client␣
↪secret>
```

By doing so the package will use these parameters to interact with Sentinel Hub unless you purposely specify an instance of `sentinelhub.SHConfig` object containing different parameters.

2.2 Amazon S3 Capabilities

The package enables downloading Sentinel-2 L1C and L2A data from Amazon S3 storage buckets. The data is contained in Requester Pays buckets therefore AWS credentials are required to use these capabilities. The credentials can be set in package's configuration file with parameters `aws_access_key_id` and `aws_secret_access_key`. This can be configured from command line:

```
$ sentinelhub.config --aws_access_key_id <your access key> --aws_secret_access_key
↪<your secret access key>
```

In case the credentials are not set, the package will instead automatically try to use **locally stored AWS credentials**, if they were configured according to AWS configuration instructions. Any other configuration parameters (e.g. region) will also be collected the same way.

The AWS credentials also have to have correct permissions to be able to download data from S3 buckets. That can be configured in AWS IAM console. There are many ways how to configure sufficient permission, one of them is setting them to *AmazonS3ReadOnlyAccess*.

Important: Because satellite data at S3 is contained in Requester Pays buckets Amazon will charge users for download according to Amazon S3 Pricing. In this case users are charged for amount of data downloaded and the number of requests. The *sentinelhub* package will make at most one GET request for each file downloaded. Files *metadata.xml*, *tileInfo.json* and *productInfo.json* will be obtained without any charge from Sentinel Hub public repository.

2.3 Other

For more configuration options check:

```
$ sentinelhub.config --help
```

CHAPTER 3

Examples

3.1 Sentinel Hub Processing API services from within Python

In this example notebook we show how to use the Processing API provided by Sentinel Hub to download satellite imagery. We describe how to use various parameters and configurations to obtain either processed products or raw band data.

Table of Contents

Prerequisites

Sentinel Hub account

Credentials

Imports

Setting area of interest

Example 1: True color (PNG) on a specific date

Example 1.1 Adding cloud mask data

Example 2: True color mosaic of least cloudy acquisitions

Example 3: All Sentinel-2's raw band values

Example 4: Save downloaded data to disk and read it from disk

Example 4.1: Save downloaded data directly to disk

Example 5: Other Data Collections

Example 6 : Multi-response request type

Example 7 : Raw dictionary request

Example 8 : Multiple timestamps data

3.1.1 Prerequisites

Sentinel Hub account

In order to use Sentinel Hub services you will need a Sentinel Hub account. If you do not have one yet, you can create a free trial account at Sentinel Hub webpage. If you are a researcher you can even apply for a free non-commercial account at ESA OSEO page.

Credentials

In the Sentinel Hub dashboard under "User settings" there is the "OAuth clients" where we create a new OAuth client and use its CLIENT_ID and CLIENT_SECRET to create an instance of `sentinelhub.SHConfig`.

Note:

Instead of providing credentials here we could also configure them beforehand according to configuration instructions.

---

[1]:
```
from sentinelhub import SHConfig

# In case you put the credentials into the configuration file you can leave this␣
↪unchanged

CLIENT_ID = ''
CLIENT_SECRET = ''
```

[2]:
```
config = SHConfig()

if CLIENT_ID and CLIENT_SECRET:
    config.sh_client_id = CLIENT_ID
    config.sh_client_secret = CLIENT_SECRET

if config.sh_client_id == '' or config.sh_client_secret == '':
    print("Warning! To use Sentinel Hub services, please provide the credentials␣
↪(client ID and client secret).")
```

Imports

[3]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

[4]:
```
import os
import datetime
import numpy as np
import matplotlib.pyplot as plt

from sentinelhub import MimeType, CRS, BBox, SentinelHubRequest,␣
↪SentinelHubDownloadClient, \
    DataCollection, bbox_to_dimensions, DownloadRequest

from utils import plot_image
```

Setting area of interest

We will download Sentinel-2 imagery of Betsiboka Estuary such as the one shown below (taken by Sentinel-2 on 2017-12-15):


SENTINEL Hub

The bounding box in `WGS84` coordinate system is `[46.16, -16.15, 46.51, -15.58]` (longitude and latitude coordinates of lower left and upper right corners). You can get the bbox for a different area at the bboxfinder website.

All requests require bounding box to be given as an instance of `sentinelhub.geometry.BBox` with corresponding Coordinate Reference System (`sentinelhub.constants.CRS`). In our case it is in *WGS84* and we can use the predefined *WGS84* coordinate reference system from `sentinelhub.constants.CRS`.

[5]:
```
betsiboka_coords_wgs84 = [46.16, -16.15, 46.51, -15.58]
```

When the bounding box bounds have been defined, you can initialize the `BBox` of the area of interest. Using the `bbox_to_dimensions` utility function, you can provide the desired resolution parameter of the image in meters and obtain the output image shape.

[6]:
```
resolution = 60
betsiboka_bbox = BBox(bbox=betsiboka_coords_wgs84, crs=CRS.WGS84)
betsiboka_size = bbox_to_dimensions(betsiboka_bbox, resolution=resolution)

print(f'Image shape at {resolution} m resolution: {betsiboka_size} pixels')
```

```
Image shape at 60 m resolution: (631, 1047) pixels
```

3.1.2 Example 1: True color (PNG) on a specific date

We build the request according to the API Reference, using the `SentinelHubRequest` class. Each Processing API request also needs an evalscript.

The information that we specify in the `SentinelHubRequest` object is:

- an evalscript,
- a list of input data collections with time interval,
- a format of the response,
- a bounding box and it's size (size or resolution).

The evalscript in the example is used to select the appropriate bands. We return the RGB (B04, B03, B02) Sentinel-2 L1C bands.

The image from Jun 12th 2020 is downloaded. Without any additional parameters in the evalscript, the downloaded data will correspond to reflectance values in `UINT8` format (values in 0-255 range).

[7]:
```
evalscript_true_color = """
    //VERSION=3

    function setup() {
        return {
            input: [{
                bands: ["B02", "B03", "B04"]
            }],
            output: {
                bands: 3
            }
        };
    }

    function evaluatePixel(sample) {
        return [sample.B04, sample.B03, sample.B02];
    }
```

(continues on next page)

(continued from previous page)

```
"""

request_true_color = SentinelHubRequest(
    evalscript=evalscript_true_color,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L1C,
            time_interval=('2020-06-12', '2020-06-13'),
        )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.PNG)
    ],
    bbox=betsiboka_bbox,
    size=betsiboka_size,
    config=config
)
```

[8]:
```
true_color_imgs = request_true_color.get_data()
```

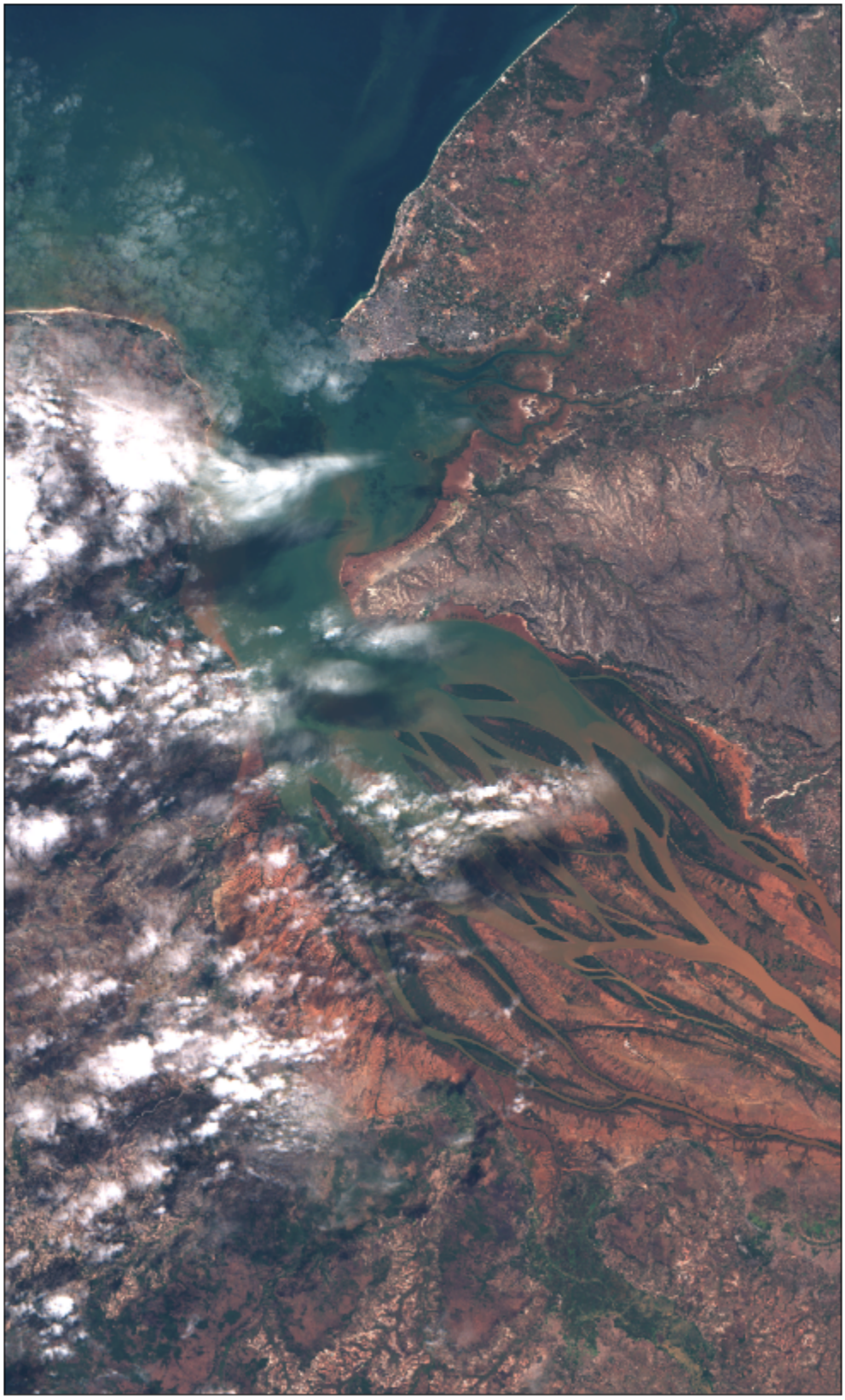
The method `get_data()` will always return a list of length 1 with the available image from the requested time interval in the form of numpy arrays.

[9]:
```
print(f'Returned data is of type = {type(true_color_imgs)} and length {len(true_color_
↪imgs)}.')
print(f'Single element in the list is of type {type(true_color_imgs[-1])} and has␣
↪shape {true_color_imgs[-1].shape}')
```

```
Returned data is of type = <class 'list'> and length 1.
Single element in the list is of type <class 'numpy.ndarray'> and has shape (1047,␣
↪631, 3)
```

[10]:
```
image = true_color_imgs[0]
print(f'Image type: {image.dtype}')

# plot function
# factor 1/255 to scale between 0-1
# factor 3.5 to increase brightness
plot_image(image, factor=3.5/255, clip_range=(0,1))
```

```
Image type: uint8
```



Example 1.1 Adding cloud mask data

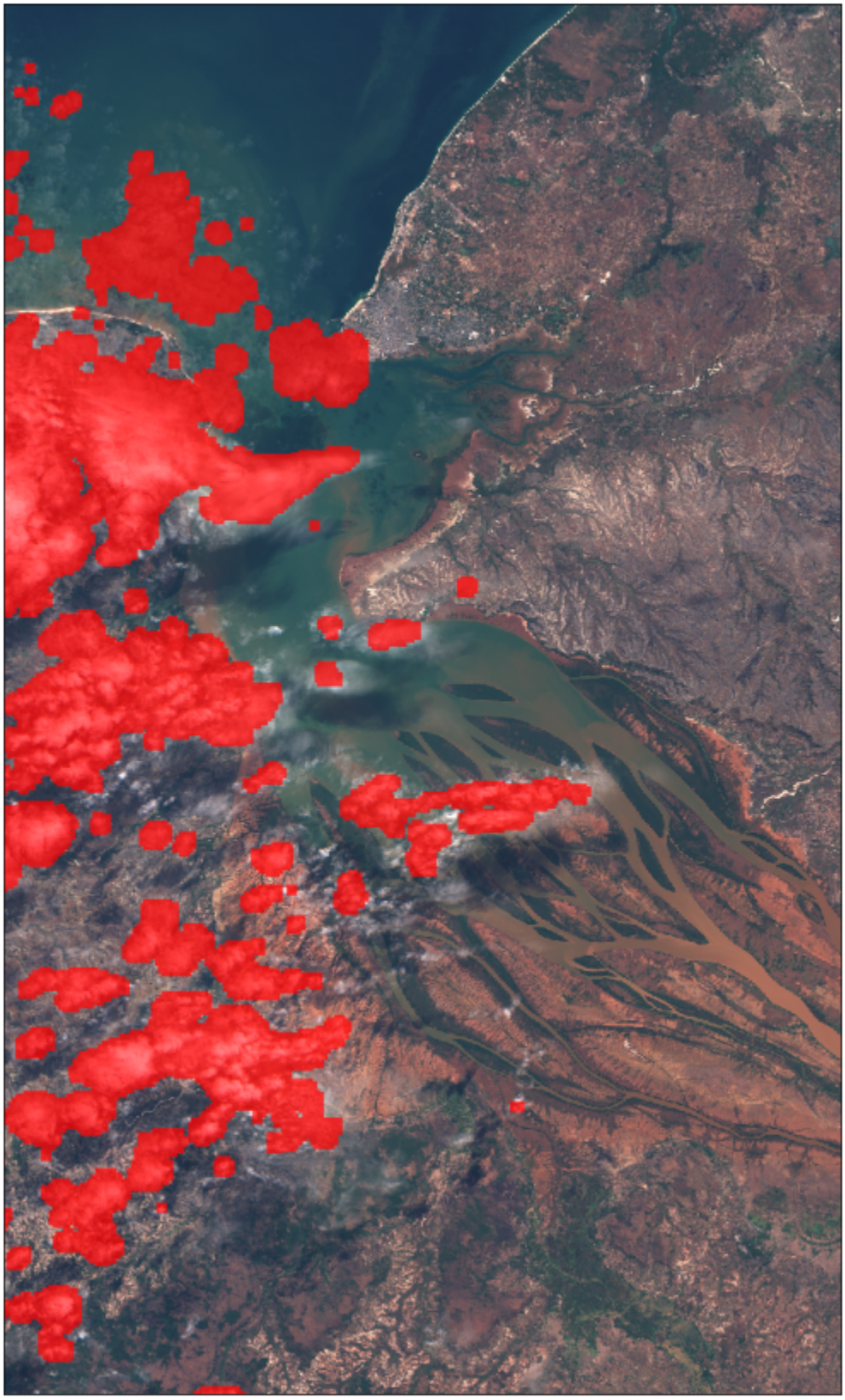
It is also possible to obtain cloud masks when requesting Sentinel-2 data by using the cloud mask band (`CLM`) or the cloud probabilities band (`CLP`). More info here.

The factor for increasing the image brightness can already be provided in the evalscript.

[11]:
```
evalscript_clm = """
//VERSION=3
function setup() {
  return {
    input: ["B02", "B03", "B04", "CLM"],
    output: { bands: 3 }
  }
}

function evaluatePixel(sample) {
  if (sample.CLM == 1) {
    return [0.75 + sample.B04, sample.B03, sample.B02]
  }
  return [3.5*sample.B04, 3.5*sample.B03, 3.5*sample.B02];
}
"""

request_true_color = SentinelHubRequest(
    evalscript=evalscript_clm,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L1C,
            time_interval=('2020-06-12', '2020-06-13'),
        )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.PNG)
    ],
    bbox=betsiboka_bbox,
    size=betsiboka_size,
    config=config
)
```

[12]:
```
data_with_cloud_mask = request_true_color.get_data()
```

[13]:
```
plot_image(data_with_cloud_mask[0], factor=1/255)
```



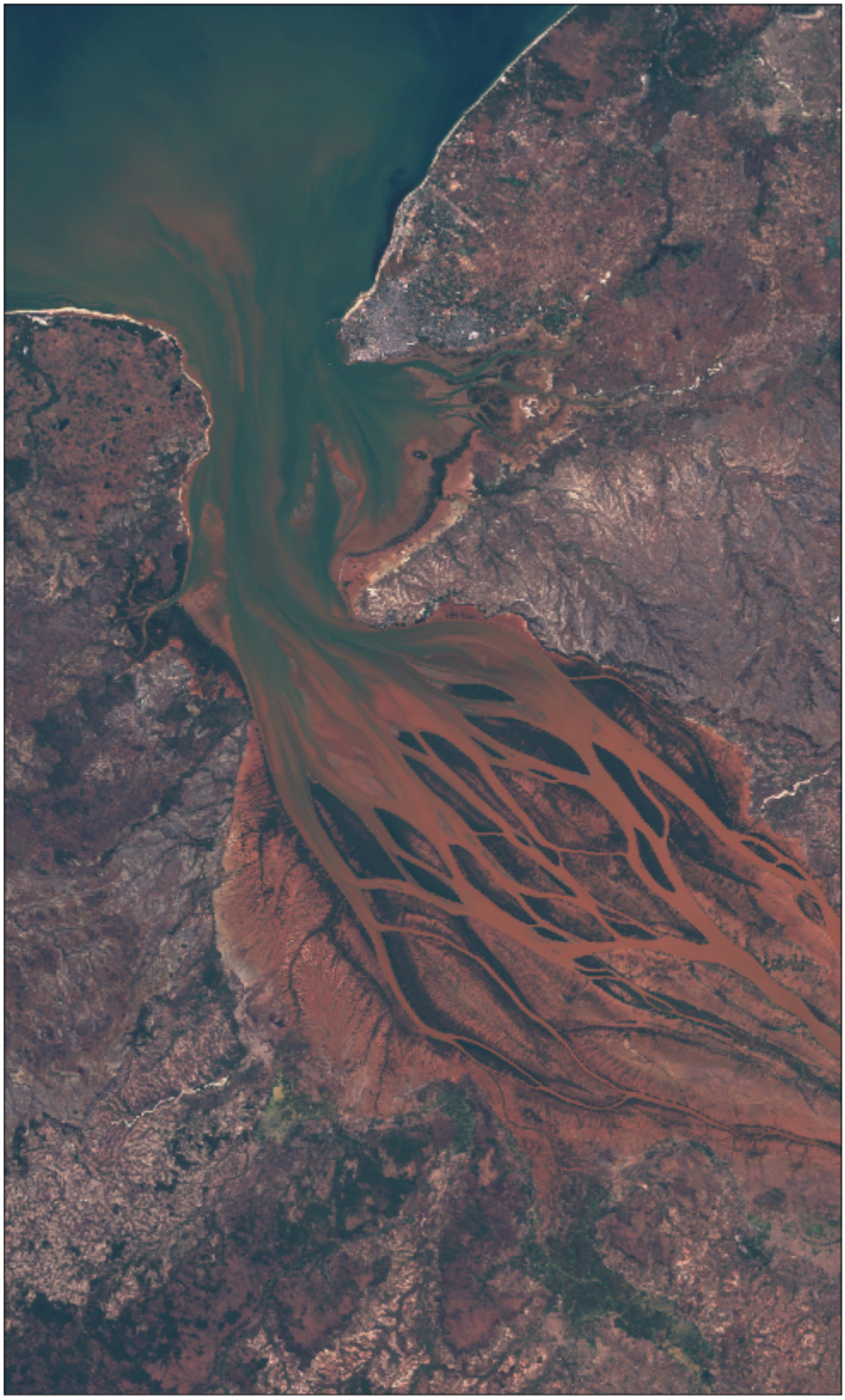
3.1.3 Example 2: True color mosaic of least cloudy acquisitions

The `SentinelHubRequest` automatically creates a mosaic from all available images in the given time interval. By default, the `mostRecent` mosaicking order is used. More information available here.

In this example we will provide a month long interval, order the images w.r.t. the cloud coverage on the tile level (`leastCC` parameter), and mosaic them in the specified order.

[14]:
```
request_true_color = SentinelHubRequest(
    evalscript=evalscript_true_color,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L1C,
            time_interval=('2020-06-01', '2020-06-30'),
            mosaicking_order='leastCC'
        )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.PNG)
    ],
    bbox=betsiboka_bbox,
    size=betsiboka_size,
    config=config
)
```

[15]:
```
plot_image(request_true_color.get_data()[0], factor=3.5/255, clip_range=(0,1))
```



3.1.4 Example 3: All Sentinel-2's raw band values

Now let's define an evalscript which will return all Sentinel-2 spectral bands with raw values.

In this example we are downloading already quite a big chunk of data, so optimization of the request is not out of the question. Downloading raw digital numbers in the `INT16` format instead of reflectances in the `FLOAT32` format means that much less data is downloaded, which results in a faster download and a smaller usage of SH processing units.

In order to achieve this, we have to set the input units in the evalscript to `DN` (digital numbers) and the output `sampleType` argument to `INT16`. Additionally, we can't pack all Sentinel-2's 13 bands into a `PNG` image, so we have to set the output image type to the `TIFF` format via `MimeType.TIFF` in the request.

The digital numbers are in the range from 0-10000, so we have to scale the downloaded data appropriately.

[16]:
```
evalscript_all_bands = """
    //VERSION=3
    function setup() {
        return {
            input: [{
                bands: ["B01","B02","B03","B04","B05","B06","B07","B08","B8A","B09",
↪"B10","B11","B12"],
                units: "DN"
            }],
            output: {
                bands: 13,
                sampleType: "INT16"
            }
        };
    }

    function evaluatePixel(sample) {
        return [sample.B01,
                sample.B02,
                sample.B03,
                sample.B04,
                sample.B05,
                sample.B06,
                sample.B07,
                sample.B08,
                sample.B8A,
                sample.B09,
                sample.B10,
                sample.B11,
                sample.B12];
    }
"""

request_all_bands = SentinelHubRequest(
    evalscript=evalscript_all_bands,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L1C,
            time_interval=('2020-06-01', '2020-06-30'),
            mosaicking_order='leastCC'
    )],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.TIFF)
```

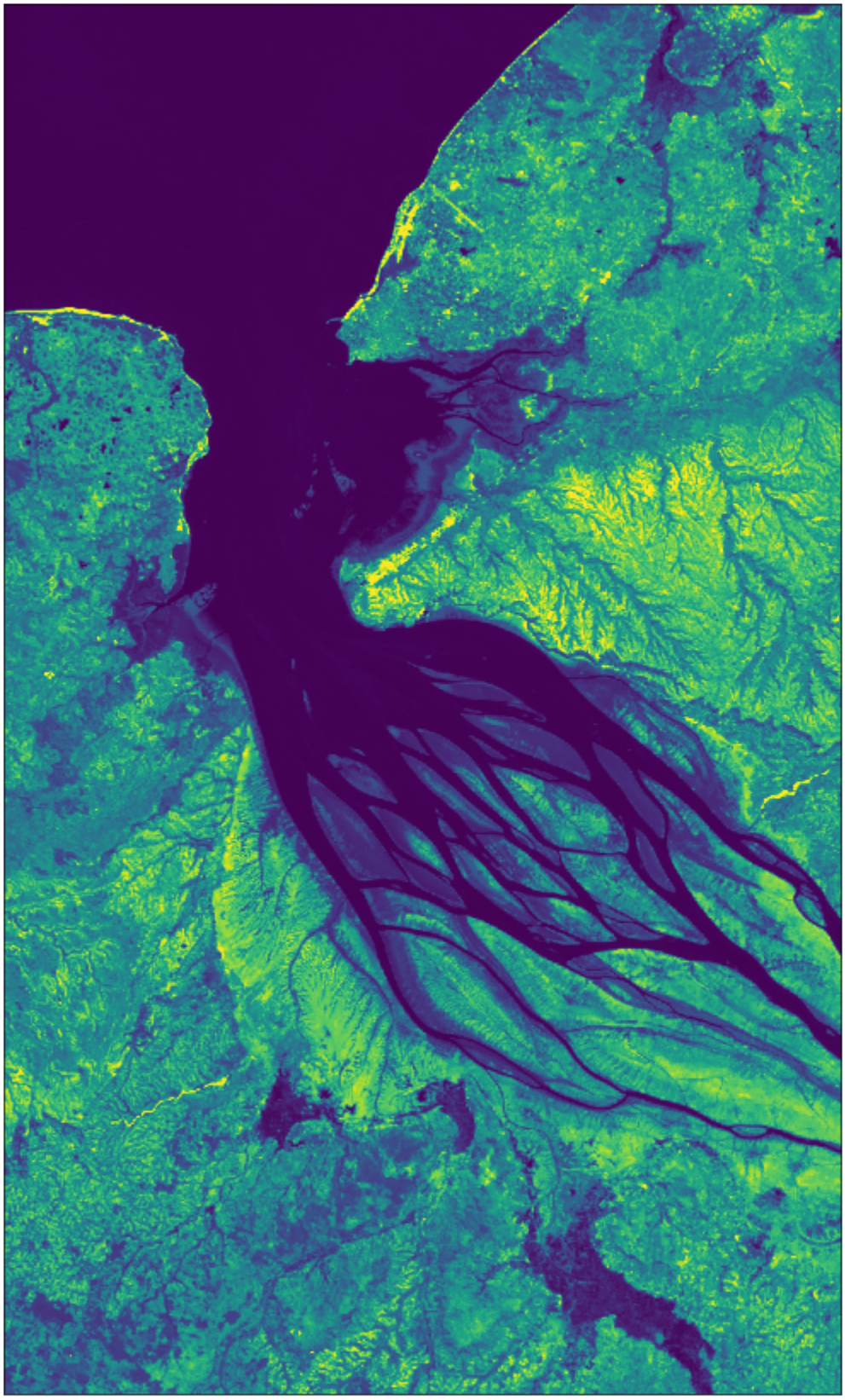
(continues on next page)

(continued from previous page)

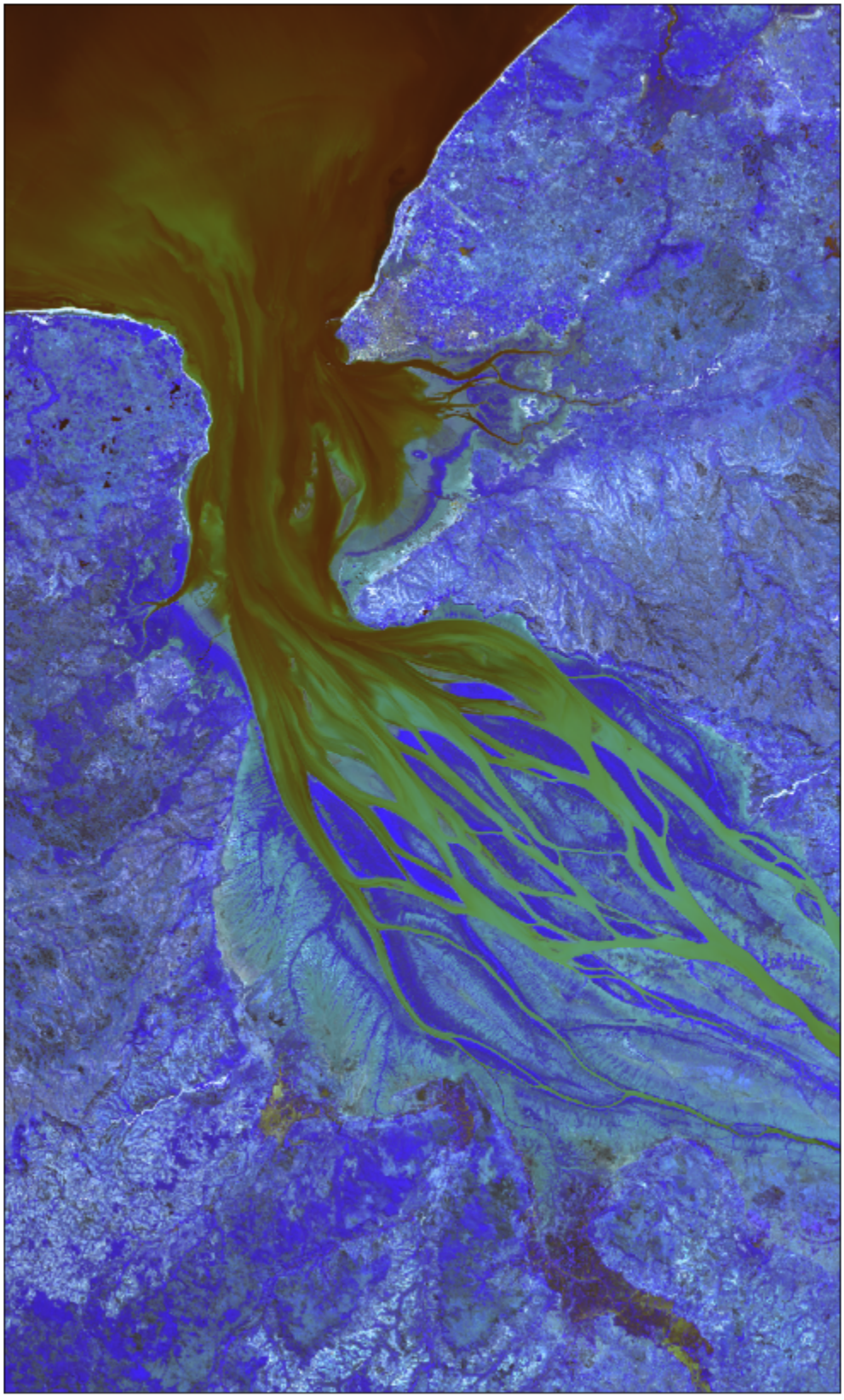
```
    ],
    bbox=betsiboka_bbox,
    size=betsiboka_size,
    config=config
)
```

[17]:
```
all_bands_response = request_all_bands.get_data()
```

[18]:
```
# Image showing the SWIR band B12
# Factor 1/1e4 due to the DN band values in the range 0-10000
# Factor 3.5 to increase the brightness
plot_image(all_bands_response[0][:, :, 12], factor=3.5/1e4, vmax=1)
```



[19]:
```
# From raw bands we can also construct a False-Color image
# False color image is (B03, B04, B08)
plot_image(all_bands_response[0][:, :, [2,3,7]], factor=3.5/1e4, clip_range=(0,1))
```



3.1.5 Example 4: Save downloaded data to disk and read it from disk

All downloaded data can be saved to disk and later read from it. Simply specify the location on disk where data should be saved (or loaded from) via the `data_folder` argument of the request's constructor. When executing the request's `get_data` method, set the argument `save_data` to `True`.

This also means that in all the future requests for data, the request will first check the provided location if the data is already there, unless you explicitly demand to redownload the data.

[20]:
```
request_all_bands = SentinelHubRequest(
    data_folder='test_dir',
    evalscript=evalscript_all_bands,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L1C,
            time_interval=('2020-06-01', '2020-06-30'),
            mosaicking_order='leastCC'
    )],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.TIFF)
    ],
    bbox=betsiboka_bbox,
    size=betsiboka_size,
    config=config
)
```

[21]:
```
%%time
all_bands_img = request_all_bands.get_data(save_data=True)
```

```
CPU times: user 601 ms, sys: 106 ms, total: 708 ms
Wall time: 6.92 s
```

[22]:
```
print(f'The output directory has been created and a tiff file with all 13 bands was␣
↪saved into ' \
      'the following structure:\n')

for folder, _, filenames in os.walk(request_all_bands.data_folder):
    for filename in filenames:
        print(os.path.join(folder, filename))
```

```
The output directory has been created and a tiff file with all 13 bands was saved␣
↪into the following structure:

test_dir/b5bf05e64663a30986783647992d69b1/response.tiff
test_dir/b5bf05e64663a30986783647992d69b1/request.json
```

[23]:
```
%%time
# try to re-download the data
all_bands_img_from_disk = request_all_bands.get_data()
```

```
CPU times: user 519 ms, sys: 7.67 ms, total: 527 ms
Wall time: 147 ms
```

[24]:
```
%%time
# force the redownload
all_bands_img_redownload = request_all_bands.get_data(redownload=True)
```

```
CPU times: user 555 ms, sys: 92.3 ms, total: 647 ms
Wall time: 6.53 s
```

Example 4.1: Save downloaded data directly to disk

The `get_data` method returns a list of numpy arrays and can save the downloaded data to disk, as we have seen in the previous example. Sometimes it is convenient to just save the data directly to disk. You can do that by using `save_data` method instead.

[25]:
```
%%time
request_all_bands.save_data()
```

```
CPU times: user 1.03 ms, sys: 0 ns, total: 1.03 ms
Wall time: 799 µs
```

[26]:
```
print(f'The output directory has been created and a tiff file with all 13 bands was␣
↪saved into ' \
      'the following structure:\n')

for folder, _, filenames in os.walk(request_all_bands.data_folder):
    for filename in filenames:
        print(os.path.join(folder, filename))
```

```
The output directory has been created and a tiff file with all 13 bands was saved␣
↪into the following structure:

test_dir/b5bf05e64663a30986783647992d69b1/response.tiff
test_dir/b5bf05e64663a30986783647992d69b1/request.json
```

3.1.6 Example 5: Other Data Collections

The `sentinelhub-py` package supports various data collections. The example below is shown for one of them, but the process is the same for all of them.

Note:

For more examples and information check the *tutorial about data collections* and Sentinel Hub documentation about data collections.

---

[27]:
```
print('Supported DataCollections:\n')
for collection in DataCollection.get_available_collections():
    print(collection)
```

```
Supported DataCollections:

DataCollection.SENTINEL2_L1C
DataCollection.SENTINEL2_L2A
DataCollection.SENTINEL1_IW
DataCollection.SENTINEL1_EW
DataCollection.SENTINEL1_EW_SH
DataCollection.SENTINEL1_IW_ASC
DataCollection.SENTINEL1_EW_ASC
DataCollection.SENTINEL1_EW_SH_ASC
DataCollection.SENTINEL1_IW_DES
DataCollection.SENTINEL1_EW_DES
DataCollection.SENTINEL1_EW_SH_DES
DataCollection.DEM
DataCollection.MODIS
```

(continues on next page)

(continued from previous page)

```
DataCollection.LANDSAT8
DataCollection.SENTINEL5P
```

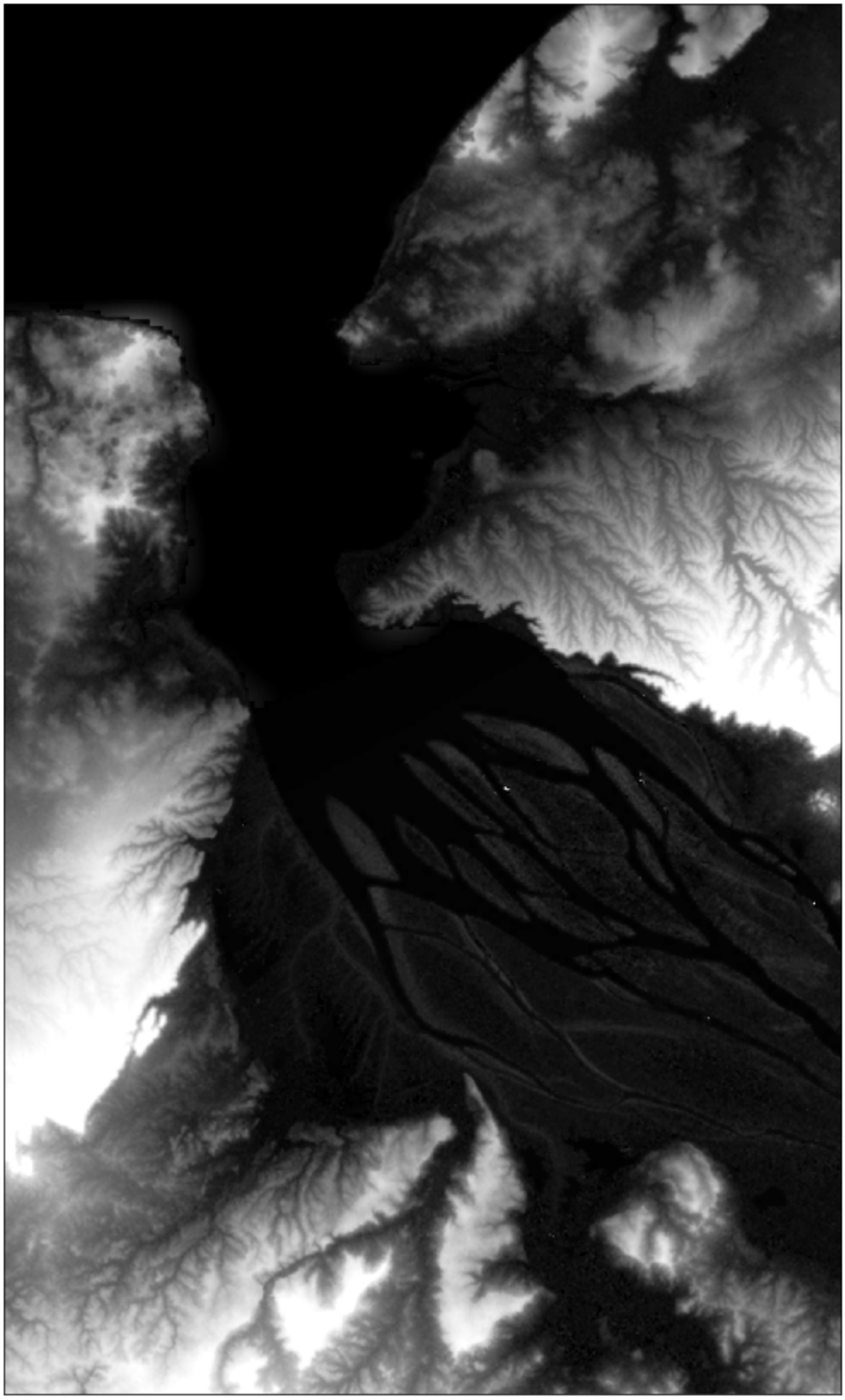
For this example let's download the digital elevation model data (DEM). The process is similar as before, we just provide the evalscript and create the request. More data on the `DEM` data collection is available here. `DEM` values are in meters and can be negative for areas which lie below sea level, so it is recommended to set the output format in your evalscript to `FLOAT32`.

[28]:
```
evalscript_dem = '''
//VERSION=3
function setup() {
  return {
    input: ["DEM"],
    output:{
      id: "default",
      bands: 1,
      sampleType: SampleType.FLOAT32
    }
  }
}

function evaluatePixel(sample) {
  return [sample.DEM]
}
'''
```

[29]:
```
dem_request = SentinelHubRequest(
    evalscript=evalscript_dem,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.DEM,
            time_interval=('2020-06-12', '2020-06-13'),
    )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.TIFF)
    ],
    bbox=betsiboka_bbox,
    size=betsiboka_size,
    config=config
)
```

[30]:
```
dem_data = dem_request.get_data()
```

[31]:
```
# Plot DEM map
# vmin = 0; cutoff at sea level (0 m)
# vmax = 120; cutoff at high values (120 m)
plot_image(dem_data[0], factor=1.0, cmap=plt.cm.Greys_r, vmin=0, vmax=120)
```



3.1.7 Example 6 : Multi-response request type

Processing API enables downloading multiple files in one response, packed together in a TAR archive.

We will get the same image as before, download in the form of digital numbers (DN) as a UINT16 TIFF file. Along with the image we will download the `inputMetadata` which contains the normalization factor value in a JSON format.

After the download we will be able to convert the `INT16` digital numbers to get the `FLOAT32` reflectances.

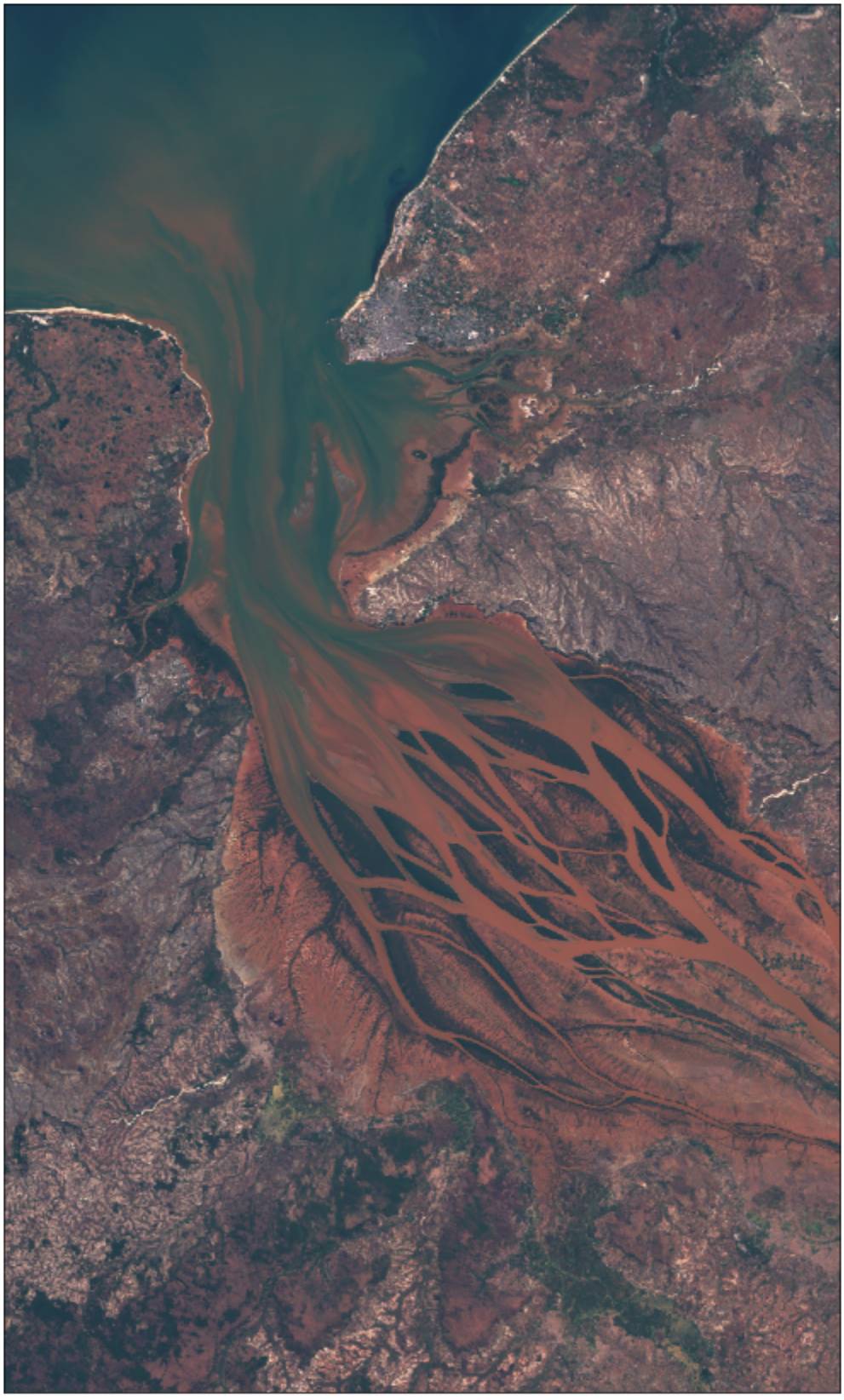
[32]:
```
evalscript = """
    //VERSION=3

    function setup() {
        return {
            input: [{
                bands: ["B02", "B03", "B04"],
                units: "DN"
            }],
            output: {
                bands: 3,
                sampleType: "INT16"
            }
        };
    }

    function updateOutputMetadata(scenes, inputMetadata, outputMetadata) {
        outputMetadata.userData = { "norm_factor":  inputMetadata.normalizationFactor␣
↪}
    }

    function evaluatePixel(sample) {
        return [sample.B04, sample.B03, sample.B02];
    }
"""

request_multitype = SentinelHubRequest(
    evalscript=evalscript,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L1C,
            time_interval=('2020-06-01', '2020-06-30'),
            mosaicking_order='leastCC'
        )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.TIFF),
        SentinelHubRequest.output_response('userdata', MimeType.JSON)
    ],
    bbox=betsiboka_bbox,
    size=betsiboka_size,
    config=config
)
```

[33]:
```
# print out information
multi_data = request_multitype.get_data()[0]
multi_data.keys()
```

[33]:
```
dict_keys(['default.tif', 'userdata.json'])
```

[34]:
```
# normalize image
img = multi_data['default.tif']
norm_factor = multi_data['userdata.json']['norm_factor']

img_float32 = img * norm_factor
```

[35]:
```
plot_image(img_float32, factor=3.5, clip_range=(0, 1))
```



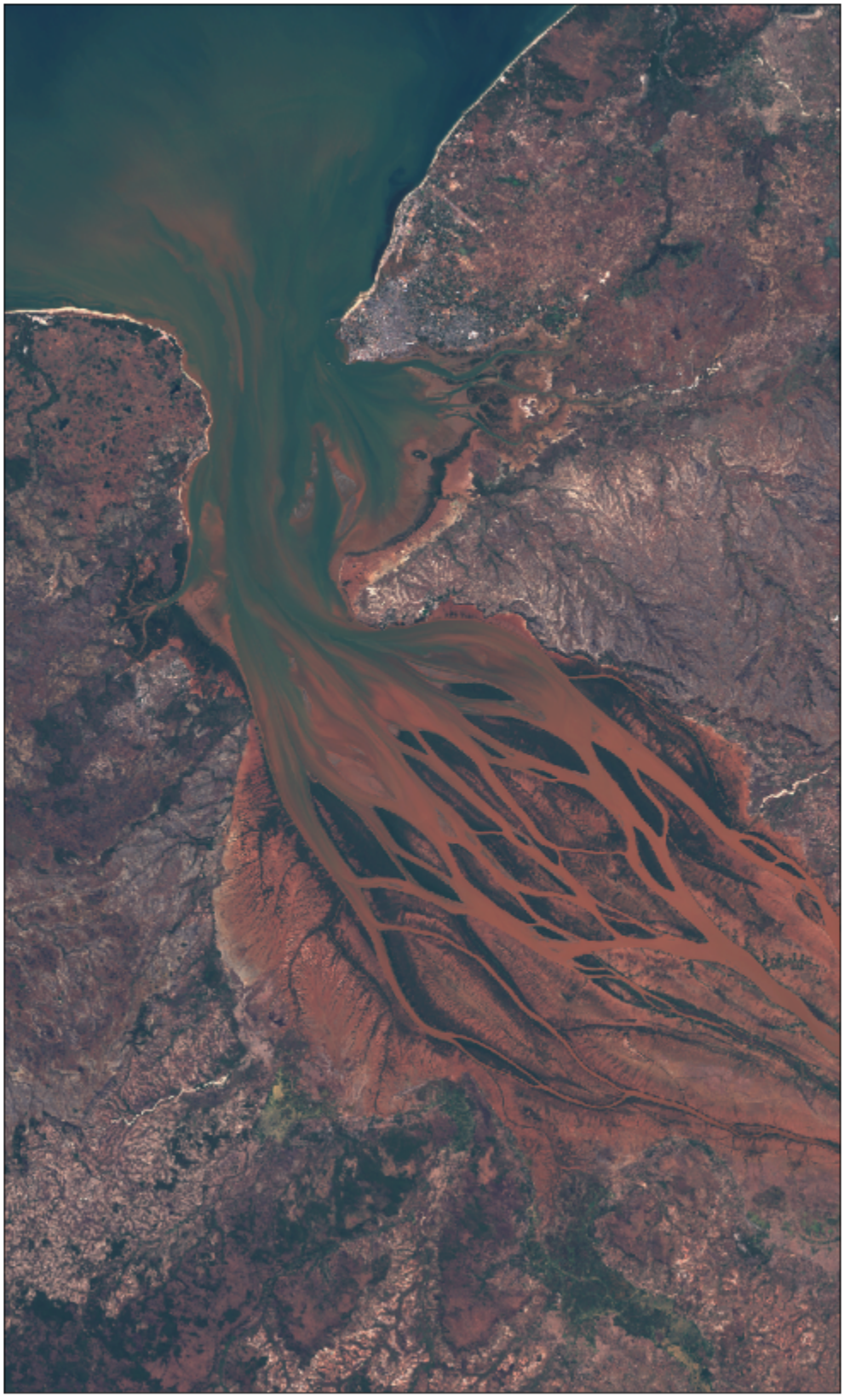
3.1.8 Example 7 : Raw dictionary request

All requests so far were built with some helper functions. We can also construct a raw dictionary as defined in the API Reference, without these helper functions, so we have full control over building the request body.

[36]:
```
request_raw_dict = {
    "input": {
        "bounds": {
            "properties": {
                "crs": betsiboka_bbox.crs.opengis_string
            },
            "bbox": list(betsiboka_bbox)
        },
        "data": [
            {
                "type": "S2L1C",
                "dataFilter": {
                    "timeRange": {"from": '2020-06-01T00:00:00Z', "to": '2020-06-
↪30T00:00:00Z'},
                    "mosaickingOrder": "leastCC",
                }
            }
        ]
    },
    "output": {
        "width": betsiboka_size[0],
        "height": betsiboka_size[1],
        "responses": [
            {
                "identifier": "default",
                "format": {
                    'type': MimeType.TIFF.get_string()
                }
            }
        ]
    },
    "evalscript": evalscript_true_color
}
```

[37]:
```
# create request
download_request = DownloadRequest(
    request_type='POST',
    url="https://services.sentinel-hub.com/api/v1/process",
    post_values=request_raw_dict,
    data_type=MimeType.TIFF,
    headers={'content-type': 'application/json'},
    use_session=True
)

# execute request
client = SentinelHubDownloadClient(config=config)
img = client.download(download_request)
```

[38]:
```
plot_image(img, factor=3.5/255, clip_range=(0,1))
```



3.1.9 Example 8 : Multiple timestamps data

It is possible to construct some logic in order to return data for multiple timestamps. By defining the `time_interval` parameter and some logic of splitting it, it is possible to create an SH reques per each "time slot" and then download the data from all the requests with the `SentinelHubDownloadClient` in `sentinelhub-py`. In this example we will create least cloudy monthly images for the year 2019.

However, this is already a functionality built on top of this SH API package. We have extended the support for such usage in our package eo-learn. We recommend to use `eo-learn` for more complex cases where you need multiple timestamps or high-resolution data for larger areas.

[39]:
```
start = datetime.datetime(2019,1,1)
end = datetime.datetime(2019,12,31)
n_chunks = 13
tdelta = (end - start) / n_chunks
edges = [(start + i*tdelta).date().isoformat() for i in range(n_chunks)]
slots = [(edges[i], edges[i+1]) for i in range(len(edges)-1)]

print('Monthly time windows:\n')
for slot in slots:
    print(slot)
```

```
Monthly time windows:

('2019-01-01', '2019-01-29')
('2019-01-29', '2019-02-26')
('2019-02-26', '2019-03-26')
('2019-03-26', '2019-04-23')
('2019-04-23', '2019-05-21')
('2019-05-21', '2019-06-18')
('2019-06-18', '2019-07-16')
('2019-07-16', '2019-08-13')
('2019-08-13', '2019-09-10')
('2019-09-10', '2019-10-08')
('2019-10-08', '2019-11-05')
('2019-11-05', '2019-12-03')
```

[40]:
```
def get_true_color_request(time_interval):
    return SentinelHubRequest(
        evalscript=evalscript_true_color,
        input_data=[
            SentinelHubRequest.input_data(
                data_collection=DataCollection.SENTINEL2_L1C,
                time_interval=time_interval,
                mosaicking_order='leastCC'
            )
        ],
        responses=[
            SentinelHubRequest.output_response('default', MimeType.PNG)
        ],
        bbox=betsiboka_bbox,
        size=betsiboka_size,
        config=config
    )
```

[41]:
```
# create a list of requests
list_of_requests = [get_true_color_request(slot) for slot in slots]
```

(continues on next page)

(continued from previous page)

```
list_of_requests = [request.download_list[0] for request in list_of_requests]

# download data with multiple threads
data = SentinelHubDownloadClient(config=config).download(list_of_requests, max_
↪threads=5)
```

[42]:
```
# some stuff for pretty plots
ncols = 4
nrows = 3
aspect_ratio = betsiboka_size[0] / betsiboka_size[1]
subplot_kw = {'xticks': [], 'yticks': [], 'frame_on': False}

fig, axs = plt.subplots(ncols=ncols, nrows=nrows, figsize=(5 * ncols * aspect_ratio,␣
↪5 * nrows),
                        subplot_kw=subplot_kw)

for idx, image in enumerate(data):
    ax = axs[idx // ncols][idx % ncols]
    ax.imshow(np.clip(image * 2.5/255, 0, 1))
    ax.set_title(f'{slots[idx][0]}  -  {slots[idx][1]}', fontsize=10)

plt.tight_layout()
```


2019-01-01 - 2019-01-29
2019-01-29 - 2019-02-26
2019-02-26 - 2019-03-26
2019-03-26 - 2019-04-23
2019-04-23 - 2019-05-21
2019-05-21 - 2019-06-18
2019-06-18 - 2019-07-16
2019-07-16 - 2019-08-13
2019-08-13 - 2019-09-10
2019-09-10 - 2019-10-08
2019-10-08 - 2019-11-05
2019-11-05 - 2019-12-03

3.2 Data collections

Sentinel Hub services enable access to a large number of satellite data collections. Specifications about data collections are available in the official Sentinel Hub service documentation. In this tutorial, we will show how to obtain data from a desired data collection with `sentinelhub-py`.

3.2.1 Use an existing data collection definition

A data collection is in the package defined with a `DataCollection` class.

[1]:
```
from sentinelhub import DataCollection
```

The class contains a collection of most commonly used data collection definitions:

[2]:
```
for collection in DataCollection.get_available_collections():
    print(collection)
```

```
DataCollection.SENTINEL2_L1C
DataCollection.SENTINEL2_L2A
DataCollection.SENTINEL1
DataCollection.SENTINEL1_IW
DataCollection.SENTINEL1_IW_ASC
DataCollection.SENTINEL1_IW_DES
DataCollection.SENTINEL1_EW
DataCollection.SENTINEL1_EW_ASC
DataCollection.SENTINEL1_EW_DES
DataCollection.SENTINEL1_EW_SH
DataCollection.SENTINEL1_EW_SH_ASC
DataCollection.SENTINEL1_EW_SH_DES
DataCollection.DEM
DataCollection.MODIS
DataCollection.LANDSAT8
DataCollection.SENTINEL5P
DataCollection.SENTINEL3_OLCI
DataCollection.SENTINEL3_SLSTR
```

Each of them is defined with a number of parameters:

[3]:
```
DataCollection.SENTINEL2_L2A
```

[3]:
```
<DataCollection.SENTINEL2_L2A: DataCollectionDefinition(
  api_id: S2L2A
  wfs_id: DSS2
  collection_type: Sentinel-2
  sensor_type: MSI
  processing_level: L2A
  bands: ('B01', 'B02', 'B03', 'B04', 'B05', 'B06', 'B07', 'B08', 'B8A', 'B09', 'B11',
↪ 'B12')
  is_timeless: False
)>
```

Such a data collection definition can be used as a parameter for a Processing API request:

[4]:
```
%matplotlib inline

from sentinelhub import SHConfig, BBox, CRS, SentinelHubRequest, MimeType
```

(continues on next page)

(continued from previous page)

```
# Write your credentials here if you haven't already put them into config.json
CLIENT_ID = ''
CLIENT_SECRET = ''

config = SHConfig()
if CLIENT_ID and CLIENT_SECRET:
    config.sh_client_id = CLIENT_ID
    config.sh_client_secret = CLIENT_SECRET


# Columbia Glacier, Alaska
glacier_bbox = BBox([-147.8, 60.96, -146.5, 61.38], crs=CRS.WGS84)
glacier_size = (700, 466)
time_interval = '2020-07-15', '2020-07-16'

evalscript_true_color = """
    //VERSION=3

    function setup() {
        return {
            input: [{
                bands: ["B02", "B03", "B04"]
            }],
            output: {
                bands: 3
            }
        };
    }

    function evaluatePixel(sample) {
        return [sample.B04, sample.B03, sample.B02];
    }
"""

request = SentinelHubRequest(
    evalscript=evalscript_true_color,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L2A,
            time_interval=time_interval,
        )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.PNG)
    ],
    bbox=glacier_bbox,
    size=glacier_size,
    config=config
)

image = request.get_data()[0]
```

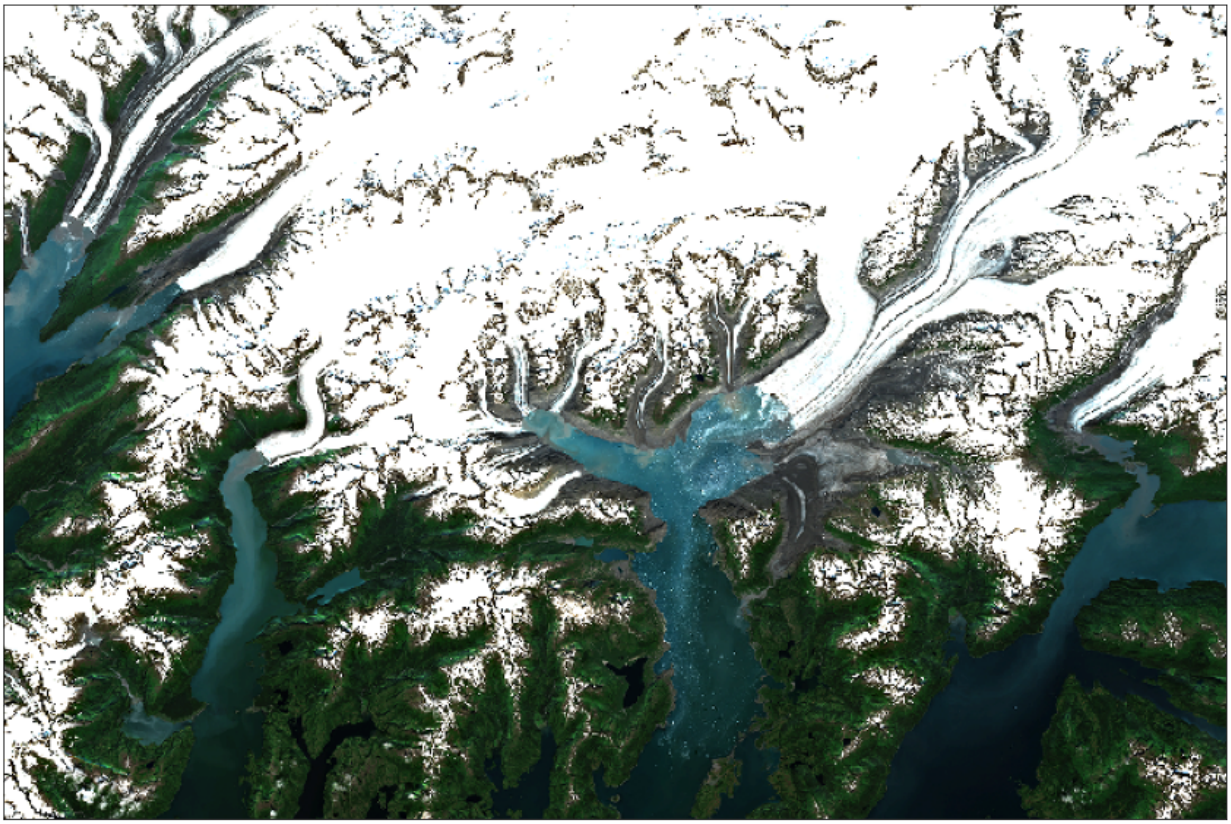
Let's plot the image:

[5]:
```
%matplotlib inline
```

(continues on next page)

(continued from previous page)

```
from utils import plot_image

plot_image(image, factor=3.5/255, clip_range=(0,1))
```



Next, we'll switch data collection to Sentinel-1, which is a SAR data collection. We'll choose only data with IW polarization and limit ourselves to ascending orbits.

[6]:
```
DataCollection.SENTINEL1_IW_ASC
```

[6]:
```
<DataCollection.SENTINEL1_IW_ASC: DataCollectionDefinition(
  api_id: S1GRD
  wfs_id: DSS3
  collection_type: Sentinel-1
  sensor_type: C-SAR
  processing_level: GRD
  swath_mode: IW
  polarization: DV
  resolution: HIGH
  orbit_direction: ASCENDING
  bands: ('VV', 'VH')
  is_timeless: False
)>
```

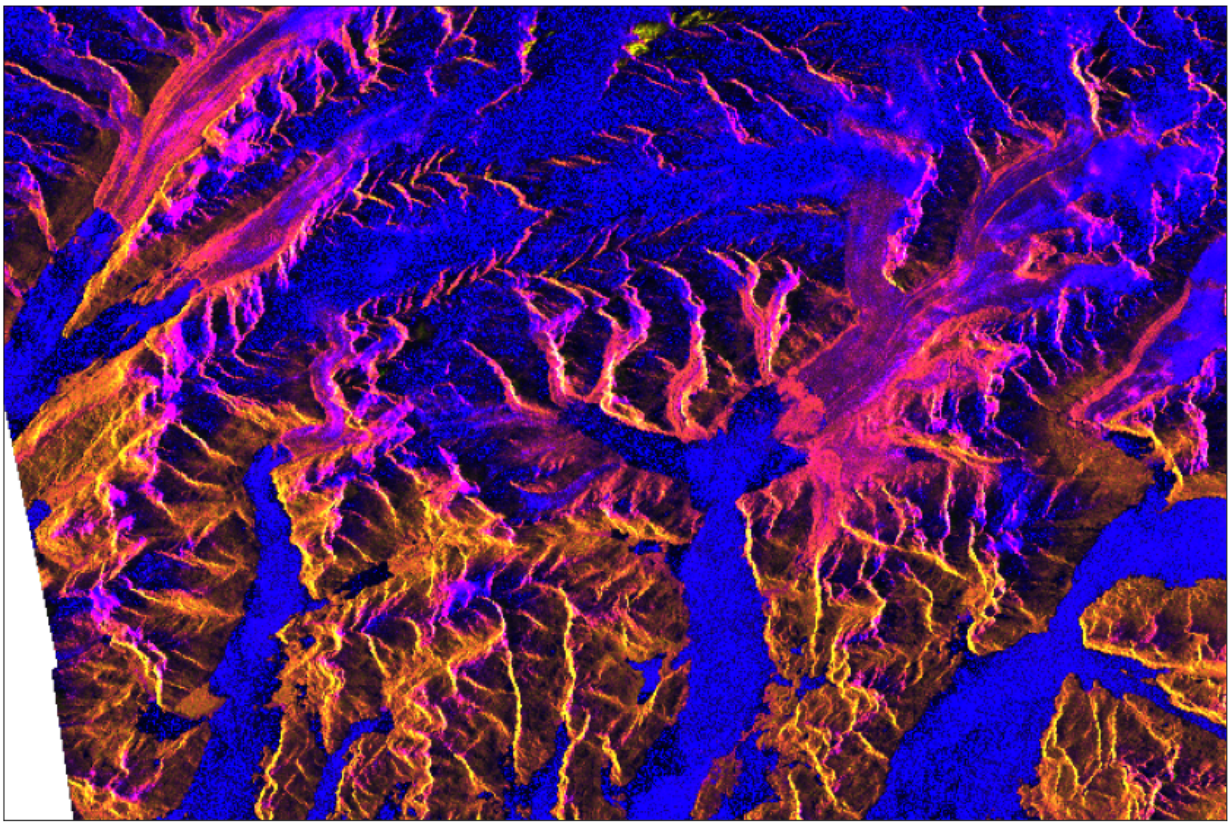
This time we'll use a simplified structure of an evalscript:

[7]:
```
evalscript = """
    //VERSION=3
```

(continues on next page)

(continued from previous page)

```
    return [VV, 2 * VH, VV / VH / 100.0, dataMask]
"""

time_interval = '2020-07-06', '2020-07-07'

request = SentinelHubRequest(
    evalscript=evalscript,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL1_IW_ASC,
            time_interval=time_interval,
        )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.PNG)
    ],
    bbox=glacier_bbox,
    size=glacier_size,
    config=config
)

image = request.get_data()[0]

plot_image(image, factor=3.5/255, clip_range=(0,1))
```



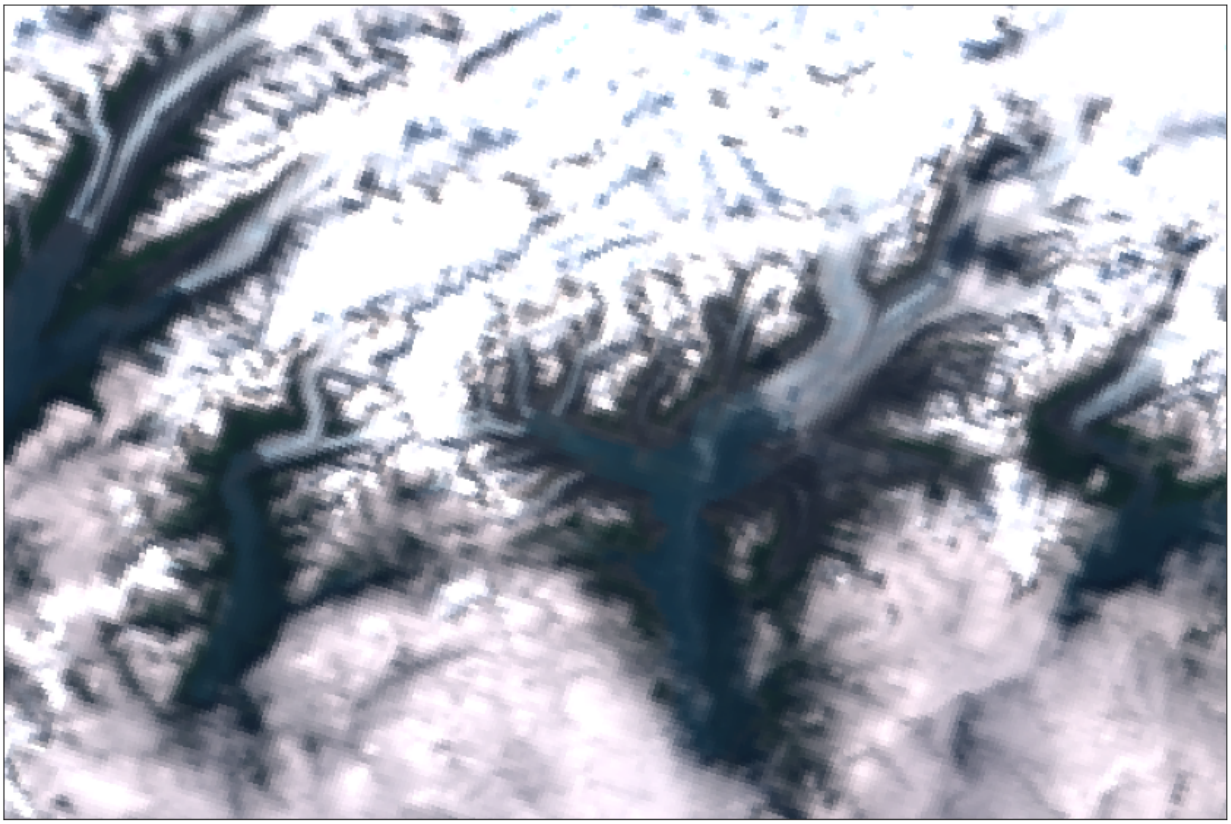
Let's also check a Sentinel-3 OLCI data collection:

[8]:
```
DataCollection.SENTINEL3_OLCI
```

[8]:
```
<DataCollection.SENTINEL3_OLCI: DataCollectionDefinition(
  api_id: S3OLCI
  wfs_id: DSS8
  service_url: https://creodias.sentinel-hub.com
  collection_type: Sentinel-3
  sensor_type: OLCI
  processing_level: L1B
  bands: ('B01', 'B02', 'B03', 'B04', 'B05', 'B06', 'B07', 'B08', 'B09', 'B10', 'B11',
↪ 'B12', 'B13', 'B14', 'B15', 'B16', 'B17', 'B18', 'B19', 'B20', 'B21')
  is_timeless: False
)>
```

Notice that its definition contains a `service_url` parameter. Its value is used to override a default `sh_base_url` defined in `config` object.

[9]:
```
evalscript = """
    //VERSION=3

    return [B08, B06, B04]
"""

time_interval = '2020-07-06', '2020-07-07'

request = SentinelHubRequest(
    evalscript=evalscript,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL3_OLCI,
            time_interval=time_interval,
        )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.PNG)
    ],
    bbox=glacier_bbox,
    size=glacier_size,
    config=config
)

image = request.get_data()[0]

plot_image(image, factor=1.5/255, clip_range=(0,1))
```



3.2.2 Define a new data collection

To support a growing number of new data collections it is possible to define a new data collection. A data collection definition supports many parameters, but it is only required to fill those parameters that will actually be required in the code.

[10]:
```
DataCollection.define(
    name='CUSTOM_SENTINEL1',
    api_id='S1GRD',
    wfs_id='DSS3',
    service_url='https://services.sentinel-hub.com',
    collection_type='Sentinel-1',
    sensor_type='C-SAR',
    processing_level='GRD',
    swath_mode='IW',
    polarization='SV',
    resolution='HIGH',
    orbit_direction='ASCENDING',
    timeliness='NRT10m',
    bands=('VV',),
    is_timeless=False
)
```

[10]:
```
<DataCollection.CUSTOM_SENTINEL1: DataCollectionDefinition(
  api_id: S1GRD
  wfs_id: DSS3
```

(continues on next page)

(continued from previous page)

```
  service_url: https://services.sentinel-hub.com
  collection_type: Sentinel-1
  sensor_type: C-SAR
  processing_level: GRD
  swath_mode: IW
  polarization: SV
  resolution: HIGH
  orbit_direction: ASCENDING
  timeliness: NRT10m
  bands: ('VV',)
  is_timeless: False
)>
```

The most common examples of user-defined data collections are "bring your own data" (BYOC) data collections, where users can bring their own data and access it with Sentinel Hub service. To be able to do that you will need to prepare a few things. Roughly speaking, these are the following (find all details here):

- Convert your data to Cloud Optimized GeoTiff (COG) format. Store it in AWS S3 bucket and allow SH to access it.
- Create a collection in SH, which points to the S3 bucket. Within the collection, ingest the tiles from the bucket.

To demonstrate this, we have prepared an example collection with a Slovenian land-cover reference map. It is available with the collection id at `7453e962-0ee5-4f74-8227-89759fbe9ba9`.

Now we can define a new BYOC data collection either with `DataCollection.define` method or with a more convenient `DataCollection.define_byoc` method:

[11]:
```
collection_id = '7453e962-0ee5-4f74-8227-89759fbe9ba9'

byoc = DataCollection.define_byoc(
    collection_id,
    name='SLOVENIA_LAND_COVER',
    is_timeless=True
)

byoc
```

[11]:
```
<DataCollection.SLOVENIA_LAND_COVER: DataCollectionDefinition(
  api_id: byoc-7453e962-0ee5-4f74-8227-89759fbe9ba9
  wfs_id: DSS10-7453e962-0ee5-4f74-8227-89759fbe9ba9
  collection_type: BYOC
  collection_id: 7453e962-0ee5-4f74-8227-89759fbe9ba9
  is_timeless: True
)>
```

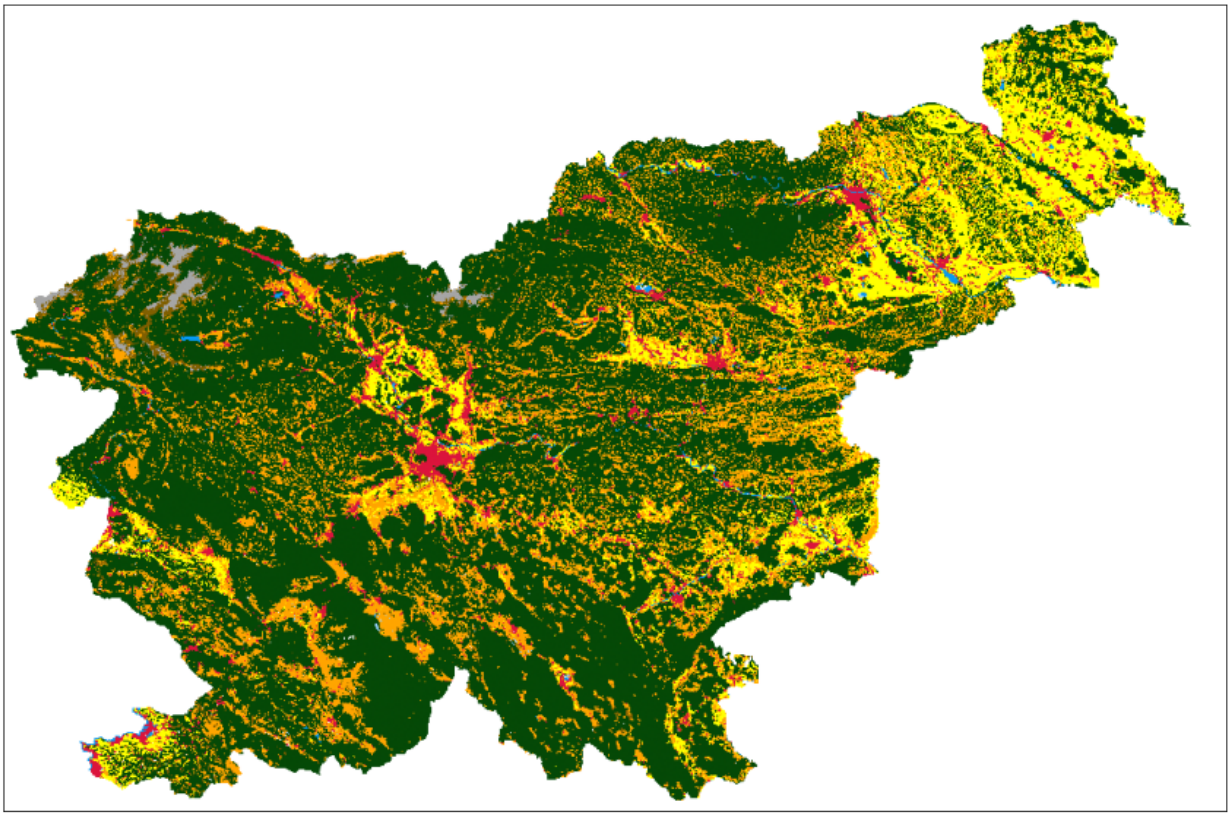
Let's load data for defined BYOC data collection:

[12]:
```
from sentinelhub import bbox_to_dimensions

slovenia_bbox = BBox([13.353882, 45.402307, 16.644287, 46.908998], crs=CRS.WGS84)
slovenia_size = bbox_to_dimensions(slovenia_bbox, resolution=240)

evalscript_byoc = """
//VERSION=3
function setup() {
  return {
    input: ["lulc_reference"],
```

(continues on next page)

(continued from previous page)

```
    output: { bands: 3 }
  };
}

var colorDict = {
  0: [255/255, 255/255, 255/255],
  1: [255/255, 255/255, 0/255],
  2: [5/255, 73/255, 7/255],
  3: [255/255, 165/255, 0/255],
  4: [128/255, 96/255, 0/255],
  5: [6/255, 154/255, 243/255],
  6: [149/255, 208/255, 252/255],
  7: [150/255, 123/255, 182/255],
  8: [220/255, 20/255, 60/255],
  9: [166/255, 166/255, 166/255],
  10: [0/255, 0/255, 0/255]
}

function evaluatePixel(sample) {
  return colorDict[sample.lulc_reference];
}
"""

byoc_request = SentinelHubRequest(
    evalscript=evalscript_byoc,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=byoc
        )],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.TIFF)
    ],
    bbox=slovenia_bbox,
    size=slovenia_size,
    config=config
)

byoc_data = byoc_request.get_data()
plot_image(byoc_data[0], factor=1/255)
```



Another option is to take an existing data collection and create a new data collection from it. The following will create a data collection with a different `service_url` parameter. Instead of collecting data from a default Sentinel Hub deployment it will collect data from MUNDI deployment.

[13]:
```
s2_l2a_mundi = DataCollection.define_from(
    DataCollection.SENTINEL2_L2A,
    'SENTINEL2_L2A_MUNDI',
    service_url='https://shservices.mundiwebservices.com'
)

s2_l2a_mundi
```

[13]:
```
<DataCollection.SENTINEL2_L2A_MUNDI: DataCollectionDefinition(
  api_id: S2L2A
  wfs_id: DSS2
  service_url: https://shservices.mundiwebservices.com
  collection_type: Sentinel-2
  sensor_type: MSI
  processing_level: L2A
  bands: ('B01', 'B02', 'B03', 'B04', 'B05', 'B06', 'B07', 'B08', 'B8A', 'B09', 'B11',
↪ 'B12')
  is_timeless: False
)>
```

[14]:
```
time_interval = '2020-06-01', '2020-07-01'

evalscript = """
    //VERSION=3
```

(continues on next page)

(continued from previous page)

```
    function setup() {
        return {
            input: [{
                bands: ["B02", "B03", "B04"]
            }],
            output: {
                bands: 3
            }
        };
    }

    function evaluatePixel(sample) {
        return [sample.B04, sample.B03, sample.B02];
    }
"""

request = SentinelHubRequest(
    evalscript=evalscript,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=s2_l2a_mundi,
            time_interval=time_interval,
            mosaicking_order='leastCC'
        )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.PNG)
    ],
    bbox=slovenia_bbox,
    size=slovenia_size,
    config=config
)

image = request.get_data()[0]

plot_image(image, factor=3.5/255, clip_range=(0,1))
```

3.3 Sentinel Hub OGC web services from within Python

Disclaimer:

Sentinel Hub service has a dedicated Processing API that extends the limiting capabilities of the standard OGC endpoints. We advise you to move to Processing API and enjoy the full power of Sentinel Hub services. See the *notebook* with Processing API examples.

3.3.1 Web Map Service (WMS) and Web Coverage Service (WCS)

In this example notebook we show how to use WMS and WCS services provided by Sentinel Hub to download satellite imagery. We describe how to use various parameters and configurations to obtain either processed products or raw band data.

We start with examples using Sentinel-2 L1C data and then show how to also obtain Sentinel-2 L2A, Sentinel-1, Landsat 8, MODIS and DEM data.

Prerequisites

Sentinel Hub account

In order to use Sentinel Hub services you will need a Sentinel Hub account. If you do not have one yet, you can create a free trial account at Sentinel Hub webpage. If you are a researcher you can even apply for a free non-commercial account at ESA OSEO page.

Once you have the account set up, login to Sentinel Hub Configurator. Inside there will already exist one configuration with an **instance ID** (alpha-numeric code of length 36). For this tutorial it is recommended that you create a new configuration (`"Add new configuration"`) and set the configuration to be based on **Python scripts template**. Such configuration will already contain all layers used in these examples. Otherwise you will have to define the layers for your configuration yourself.

After you have decided which configuration to use, you have two options. You can either put configuration's **instance ID** into `sentinelhub` package's configuration file following the configuration instructions or you can write it down in the following cell:

[1]:
```
from sentinelhub import SHConfig


INSTANCE_ID = ''  # In case you put instance ID into configuration file you can leave␣
↪this unchanged

if INSTANCE_ID:
    config = SHConfig()
    config.instance_id = INSTANCE_ID
else:
    config = None
```

Imports

[2]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

[3]:
```
import datetime
import numpy as np

import matplotlib.pyplot as plt
```

Note: `matplotlib` is not a dependency of `sentinelhub`.

[4]:
```
from sentinelhub import WmsRequest, WcsRequest, MimeType, CRS, BBox, DataCollection
```

[5]:
```
def plot_image(image, factor=1):
    """
    Utility function for plotting RGB images.
    """
    fig = plt.subplots(nrows=1, ncols=1, figsize=(15, 7))

    if np.issubdtype(image.dtype, np.floating):
        plt.imshow(np.minimum(image * factor, 1))
    else:
        plt.imshow(image)
```

Setting area of interest

We will download Sentinel-2 imagery of Betsiboka Estuary such as the one shown below (taken by Sentinel-2 on 2017-12-15):

The bounding box in *WGS84* coordinate system is (longitude and latitude coordinates of upper left and lower right corners):

[6]:
```
betsiboka_coords_wgs84 = [46.16, -16.15, 46.51, -15.58]
```

All requests require bounding box to be given as an instance of `sentinelhub.geometry.BBox` with corresponding Coordinate Reference System (`sentinelhub.geometry.CRS`). In our case it is in *WGS84* and we can use the predefined *WGS84* coordinate reference system from `sentinelhub.geometry.CRS`.

[7]:
```
betsiboka_bbox = BBox(bbox=betsiboka_coords_wgs84, crs=CRS.WGS84)
```

WMS request

Example 1: True color (PNG) on a specific date

We need to specify the following arguments in the initialization of a `WmsRequest`:

- `layer` - set it to `'TRUE-COLOR-S2-L1C'`
  - In case you are not using a configuration based on **Python scripts template** you will now have to create a layer named `TRUE-COLOR-S2-L1C` yourself. In Sentinel Hub Configurator go to your configuration, add new layer which will use Sentinel-2 L1C data collection and predefined product `TRUE COLOR, RGB Visualization` for `Data processing` parameter.
- `bbox` - see above
- `time` - acquisition date
  - we'll set it to 2017-12-15
- `width` and `height` - width and height of a returned image
  - we'll set them to 512 and 856, respectively
  - we could only set one of the two parameters and the other one would be set automatically in a way that image would best fit bounding box ratio
- `instance_id` - see above

All of the above arguments are obligatory and have to be set for all `WmsRequest`.

[8]:
```
wms_true_color_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='TRUE-COLOR-S2-L1C',
    bbox=betsiboka_bbox,
    time='2017-12-15',
    width=512,
    height=856,
    config=config
)
```

[9]:
```
wms_true_color_img = wms_true_color_request.get_data()
```

Method `get_data()` will always return a list images in form of numpy arrays.

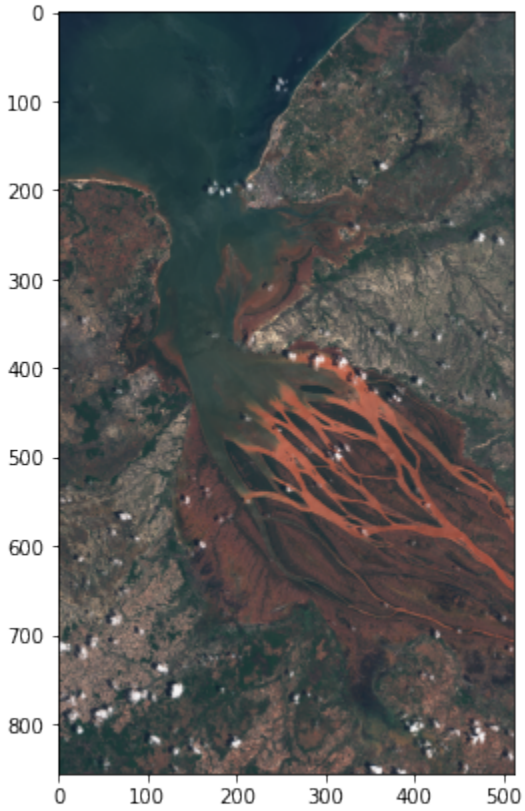
[10]:
```
print('Returned data is of type = %s and length %d.' % (type(wms_true_color_img),␣
↪len(wms_true_color_img)))
```

```
Returned data is of type = <class 'list'> and length 1.
```

[11]:
```
print('Single element in the list is of type {} and has shape {}'.format(type(wms_
↪true_color_img[-1]),
                                                                          wms_true_
↪color_img[-1].shape))
```

```
Single element in the list is of type <class 'numpy.ndarray'> and has shape (856, 512,
↪ 4)
```

[12]:
```
plot_image(wms_true_color_img[-1])
```



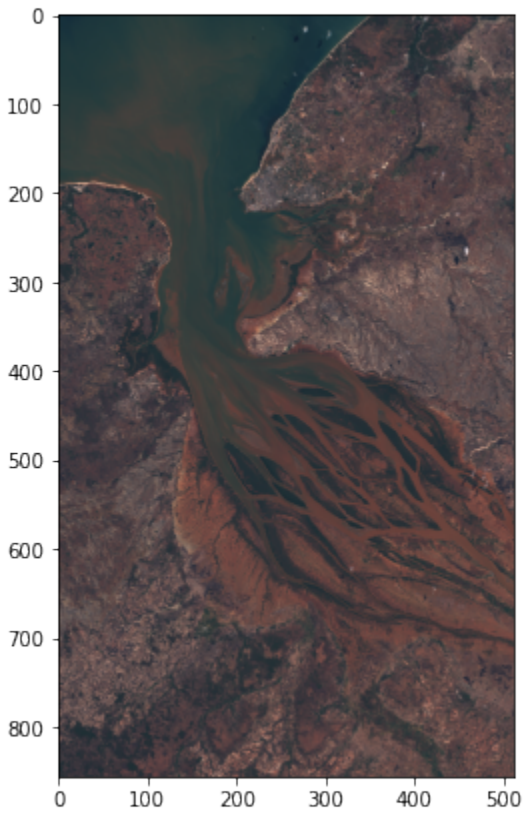
Example 2: True color of the latest acquisition

In order to get the latest Sentinel-2 acquisition set the `time` argument to `'latest'`.

[13]:
```
wms_true_color_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='TRUE-COLOR-S2-L1C',
    bbox=betsiboka_bbox,
    time='latest',
    width=512,
    height=856,
    config=config
)
```

[14]:
```
wms_true_color_img = wms_true_color_request.get_data()
```

[15]:
```
plot_image(wms_true_color_img[-1])
```



[16]:
```
print('The latest Sentinel-2 image of this area was taken on {}.'.format(wms_true_
↪color_request.get_dates()[-1]))
```

```
The latest Sentinel-2 image of this area was taken on 2020-08-31 07:14:10.
```

In case a part of the image above is completely white that is because the latest acquisition only partially intersected the specified bounding box. To avoid that we could use a `time_difference` parameter described in *Example 8*.

Example 3: True color of the multiple acquisitions in certain time window

In order to get all Sentinel-2 acquisitions taken in a certain time interval set the `time` argument to tuple with two elements (`start date`,`end date`).

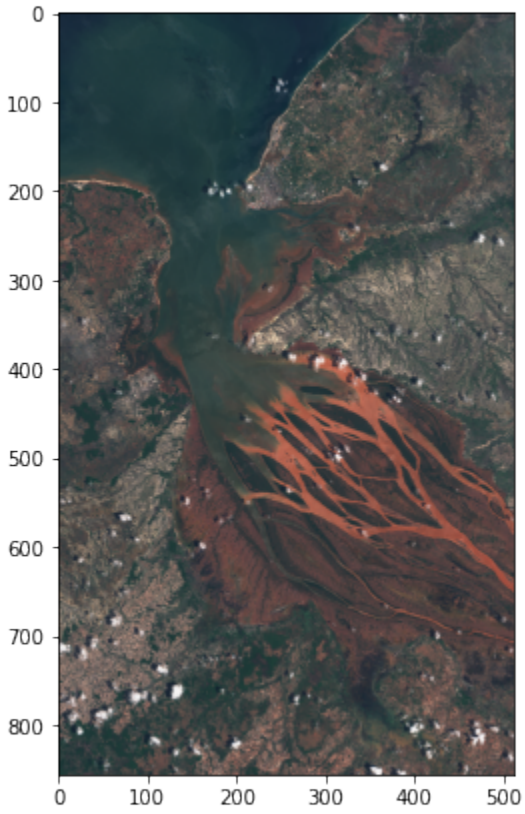
[17]:
```
wms_true_color_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='TRUE-COLOR-S2-L1C',
    bbox=betsiboka_bbox,
    time=('2017-12-01', '2017-12-31'),
    width=512,
    height=856,
    config=config
)
```

[18]:
```
wms_true_color_img = wms_true_color_request.get_data()
```

[19]:
```
print('There are %d Sentinel-2 images available for December 2017.' % len(wms_true_
↪color_img))
```

```
There are 6 Sentinel-2 images available for December 2017.
```

[20]:
```
plot_image(wms_true_color_img[2])
```



[21]:
```
print('These %d images were taken on the following dates:' % len(wms_true_color_img))
for index, date in enumerate(wms_true_color_request.get_dates()):
    print(' - image %d was taken on %s' % (index, date))
```

```
These 6 images were taken on the following dates:
 - image 0 was taken on 2017-12-05 07:13:30
 - image 1 was taken on 2017-12-10 07:12:10
 - image 2 was taken on 2017-12-15 07:12:03
 - image 3 was taken on 2017-12-20 07:12:10
 - image 4 was taken on 2017-12-25 07:12:04
 - image 5 was taken on 2017-12-30 07:12:09
```

Example 4: True color of the multiple acquisitions in certain time window with cloud coverage less than 30%

In order to get only Sentinel-2 acquisitions with cloud coverage less than certain amount set `maxcc` argument to that value. Note that this cloud coverage is estimated on the entire Sentinel-2 tile and not just for the region defined by our bounding box.

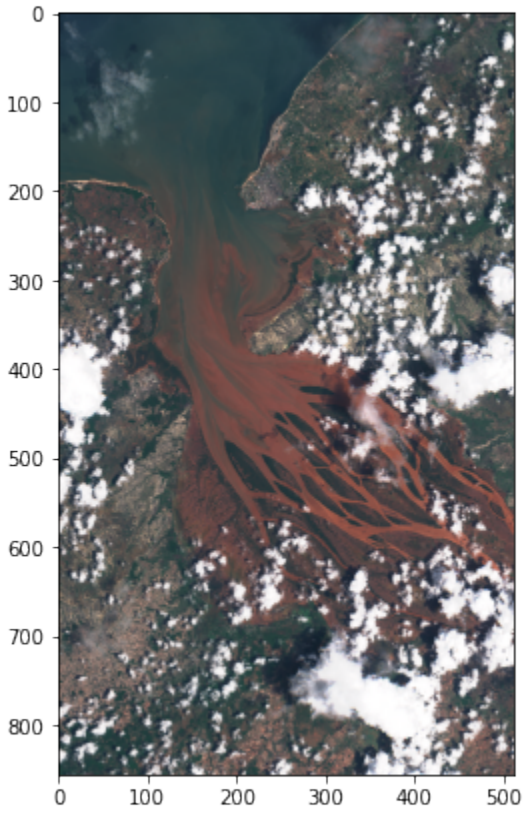
[22]:
```
wms_true_color_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='TRUE-COLOR-S2-L1C',
    bbox=betsiboka_bbox,
    time=('2017-12-01', '2017-12-31'),
    width=512, height=856,
    maxcc=0.3,
    config=config
)
```

[23]:
```
wms_true_color_img = wms_true_color_request.get_data()
```

[24]:
```
print('There are %d Sentinel-2 images available for December 2017 with cloud coverage␣
↪less ' \
      'than %1.0f%%.' % (len(wms_true_color_img), wms_true_color_request.maxcc * 100.
↪0))
```

```
There are 2 Sentinel-2 images available for December 2017 with cloud coverage less␣
↪than 30%.
```

[25]:
```
plot_image(wms_true_color_img[-1])
```



[26]:
```
print('These %d images were taken on the following dates:' % len(wms_true_color_img))
for index, date in enumerate(wms_true_color_request.get_dates()):
    print(' - image %d was taken on %s' % (index, date))
```

```
These 2 images were taken on the following dates:
 - image 0 was taken on 2017-12-15 07:12:03
```

(continues on next page)

(continued from previous page)

```
 - image 1 was taken on 2017-12-20 07:12:10
```

Example 5: All Sentinel-2's raw band values

Now let's use a layer named `BANDS-S2-L1C` which will return all Sentinel-2 spectral bands with raw values.

If you are not using a configuration based on **Python scripts template** you will again have to create such layer manually. In that case define `Data processing` parameter with the following custom script:

```
return [B01,B02,B03,B04,B05,B06,B07,B08,B8A,B09,B10,B11,B12]
```

We have to set the `image_format` argument to `sentinelhub.constants.MimeType.TIFF`, since we can't pack all Sentinel-2's 13 bands into a png image. A type of returned data (uint8, uint16 or float32) should be set in an layer definition in the Configurator.

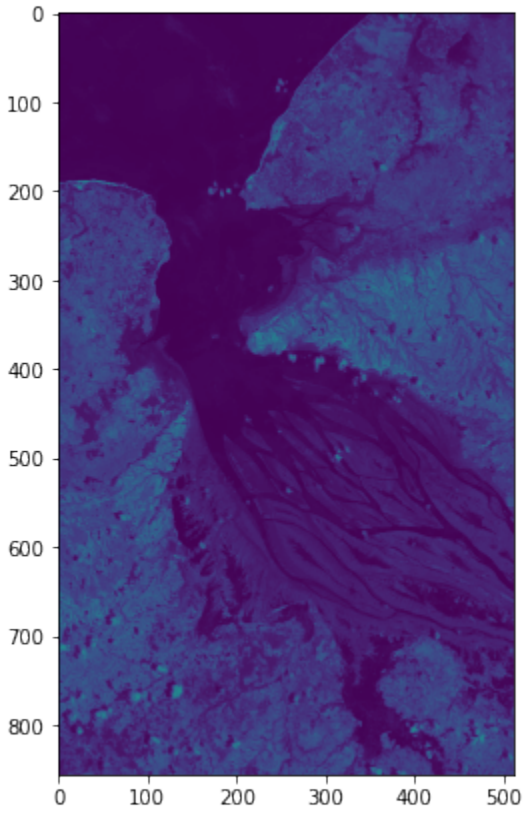
[27]:
```
wms_bands_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='BANDS-S2-L1C',
    bbox=betsiboka_bbox,
    time='2017-12-15',
    width=512,
    height=856,
    image_format=MimeType.TIFF,
    config=config
)

wms_bands_img = wms_bands_request.get_data()
```

[28]:
```
wms_bands_img[-1][:, :, 12].shape
```

[28]:
```
(856, 512)
```

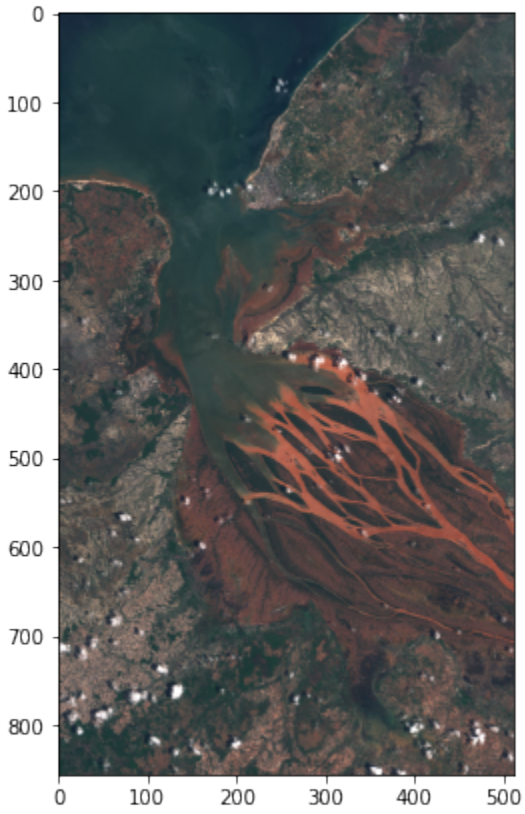
Image showing SWIR band B12

[29]:
```
plot_image(wms_bands_img[-1][:, :, 12])
```



From raw bands we can also construct a true color image

[30]:
```
plot_image(wms_bands_img[-1][:, :, [3, 2, 1]], 2.5)
```



Example 6: Save downloaded data to disk and read it from disk

All downloaded data can be saved to disk and later read from it. Simply specify the location on disk where data should be saved (or loaded from) via `data_folder` argument of request's constructor and set the argument `save_data` of `get_data` method to `True`.

[31]:
```
wms_bands_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    data_folder='test_dir',
    layer='BANDS-S2-L1C',
    bbox=betsiboka_bbox,
    time='2017-12-15',
    width=512,
    height=856,
    image_format=MimeType.TIFF,
    config=config
)
```

[32]:
```
%%time
wms_bands_img = wms_bands_request.get_data(save_data=True)
```

```
CPU times: user 186 ms, sys: 16 ms, total: 202 ms
Wall time: 108 ms
```

The output directory has been created and a tiff file with all 13 bands was saved into the following structure:

[33]:
```
import os

for folder, _, filenames in os.walk(wms_bands_request.data_folder):
    for filename in filenames:
        print(os.path.join(folder, filename))
```

```
test_dir/467e69e80e29a087e4467f62e6d07a0e/response.tiff
test_dir/467e69e80e29a087e4467f62e6d07a0e/request.json
```

Since data has been already downloaded the next request will read the data from disk instead of downloading it. That will be much faster.

[34]:
```
wms_bands_request_from_disk = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    data_folder='test_dir',
    layer='BANDS-S2-L1C',
    bbox=betsiboka_bbox,
    time='2017-12-15',
    width=512,
    height=856,
    image_format=MimeType.TIFF,
    config=config
)
```

[35]:
```
%%time

wms_bands_img_from_disk = wms_bands_request_from_disk.get_data()
```

```
CPU times: user 194 ms, sys: 7.78 ms, total: 202 ms
Wall time: 111 ms
```

[36]:
```
if np.array_equal(wms_bands_img[-1], wms_bands_img_from_disk[-1]):
    print('Arrays are equal.')
else:
    print('Arrays are different.')
```

```
Arrays are equal.
```

If you need to redownload the data again, just set the `redownload` argument of `get_data()` method to `True`.

[37]:
```
%%time
wms_bands_img_redownload = wms_bands_request_from_disk.get_data(redownload=True)
```

```
CPU times: user 291 ms, sys: 77 ms, total: 368 ms
Wall time: 11.2 s
```

Example 7: Save downloaded data directly to disk

The `get_data` method returns a list of numpy arrays and can save the downloaded data to disk, as we have seen in the previous example. Sometimes you would just like to save the data directly to disk for later use. You can do that by using `save_data` method instead.

[38]:
```
wms_true_color_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    data_folder='test_dir_tiff',
    layer='TRUE-COLOR-S2-L1C',
```

(continues on next page)

(continued from previous page)

```
    bbox=betsiboka_bbox,
    time=('2017-12-01','2017-12-31'),
    width=512,
    height=856,
    image_format=MimeType.TIFF,
    config=config
)
```

[39]:
```
%%time
wms_true_color_request.save_data()
```

```
CPU times: user 5.23 ms, sys: 0 ns, total: 5.23 ms
Wall time: 3.11 ms
```

The output directory has been created and tiff files for all 6 images should be in it.

[40]:
```
os.listdir(wms_true_color_request.data_folder)
```

[40]:
```
['ee59102678fde525f06418601da3b114',
 '5edecb454b02935703eda6098379d16a',
 '00e8db7152a73a71c3a2e2205801ac3e',
 '2f385cca8d34e094af28619ef83d23fc',
 '38d2c6811d5ab83de501887a24e3b5f3',
 '9a6283653899db769b43bfe6077840c2']
```

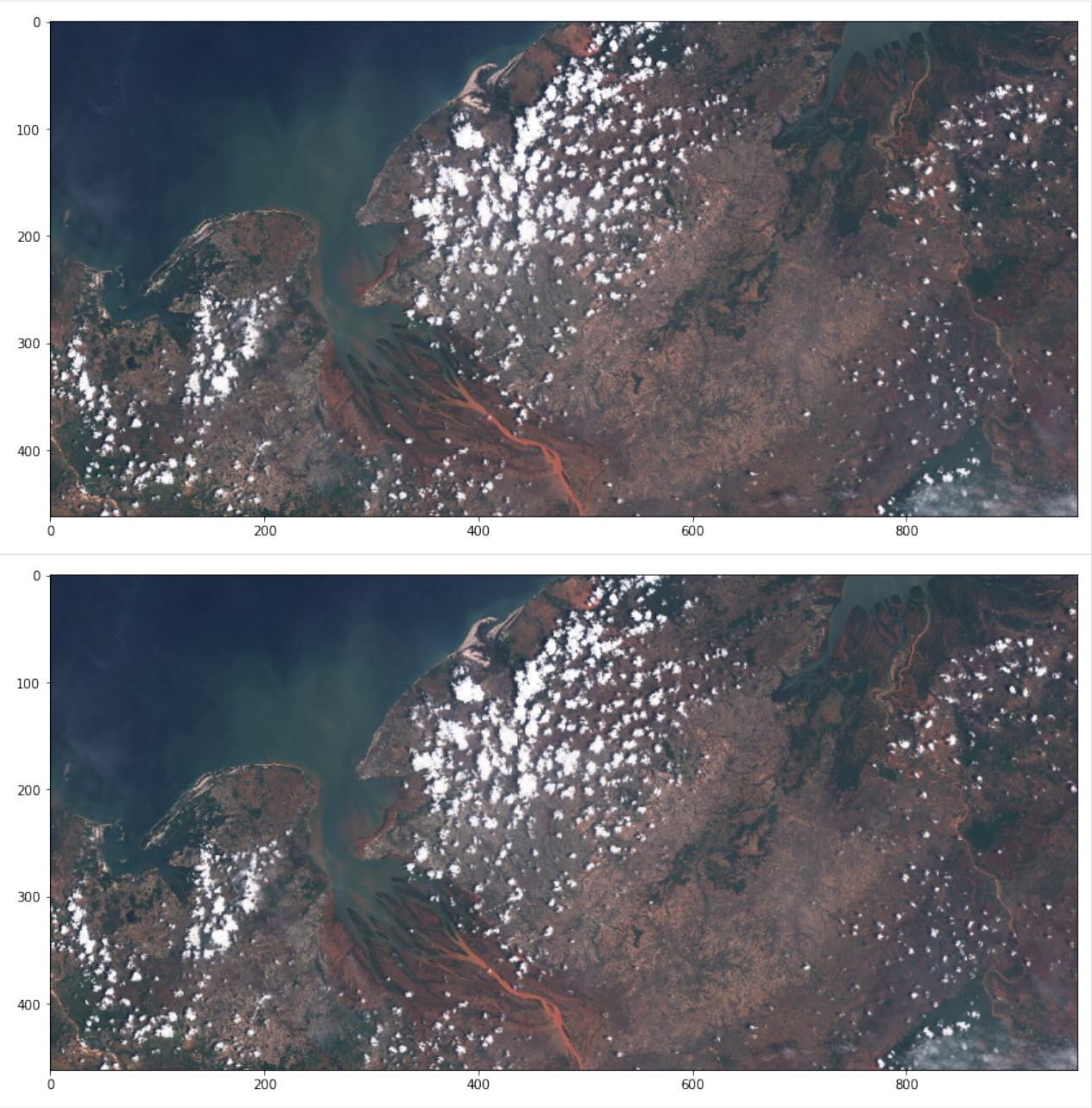
Example 8: Merging two or more download requests into one

If the bounding box spans over two or more Sentinel-2 tiles and each of them has slightly different time stamp, then download request will be created for each time stamp. Therefore we will obtain two or more images which could be completely the same or partially blank. It depends on whether the tiles from the same orbit are from the same or from two different data strips.

Let's look at the specific example. Again, we're going to look at Betsiboka estuary, but we'll increase the bounding box so that we cover an area of two different Senteinel-2 tiles.

[41]:
```
betsiboka_bbox_large = BBox([45.88, -16.12, 47.29, -15.45], crs=CRS.WGS84)

wms_true_color_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='TRUE-COLOR-S2-L1C',
    bbox=betsiboka_bbox_large,
    time='2015-12-01',
    width=960,
    image_format=MimeType.PNG,
    config=config
)

wms_true_color_img = wms_true_color_request.get_data()
```

[42]:
```
plot_image(wms_true_color_img[0])
plot_image(wms_true_color_img[1])
```



Clearly these are the same images and we usually would want to get only one. We can do that by widening the time interval in which two or more download requests are considered to be the same. In our example it is enough to widen the time window for 10 minutes, but usually it can be up to two hours. This is done by setting the `time_difference` argument.

[43]:
```
wms_true_color_request_with_deltat = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='TRUE-COLOR-S2-L1C',
    bbox=betsiboka_bbox_large,
    time='2015-12-01',
    width=960,
    image_format=MimeType.PNG,
    time_difference=datetime.timedelta(hours=2),
```

(continues on next page)

(continued from previous page)

```
    config=config
)

wms_true_color_img = wms_true_color_request_with_deltat.get_data()
```

[44]:
```
print('These %d images were taken on the following dates:' % len(wms_true_color_img))
for index, date in enumerate(wms_true_color_request_with_deltat.get_dates()):
    print(' - image %d was taken on %s' % (index, date))
```

```
These 1 images were taken on the following dates:
 - image 0 was taken on 2015-12-01 07:12:50
```

[45]:
```
plot_image(wms_true_color_img[-1])
```

WCS request

The use of `WcsRequest` is exactly the same as of the `WmsRequest` shown above. The only difference is that instead of specifying image size we specify the spatial resolution of the image. We do that by setting the `resx` and `resy` arguments to the desired resolution in meters. E.g. setting `resx='10m'` and `resy='10m'` will return an image where every pixel will cover an area of size `10m x 10m`.

Every other parameter described in this tutorial will work the same for WMS and WCS requests.

Example 9: True color with specified resolution

[46]:
```
wcs_true_color_request = WcsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='TRUE-COLOR-S2-L1C',
    bbox=betsiboka_bbox,
    time='2017-12-15',
    resx='60m',
```

(continues on next page)

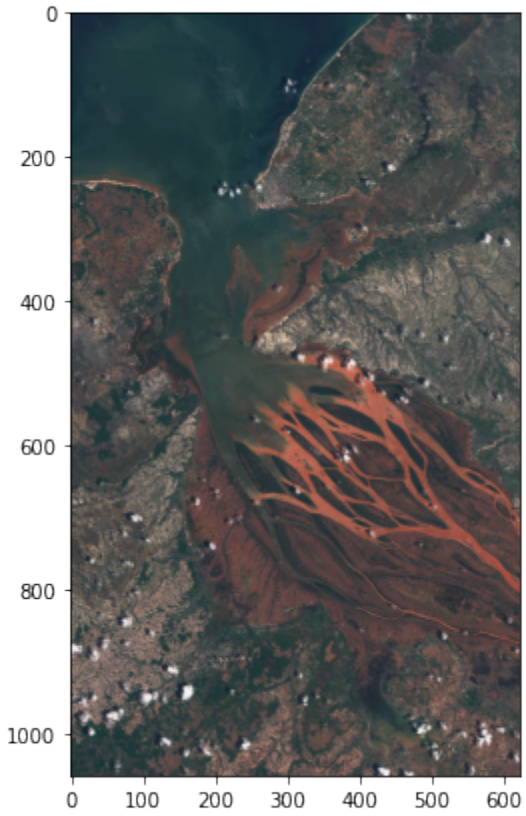
(continued from previous page)

```
    resy='60m',
    config=config
)

wcs_true_color_img = wcs_true_color_request.get_data()
```

[47]:
```
print('Single element in the list is of type = {} and has shape {}'.format(type(wcs_
↪true_color_img[-1]),
                                                                            wcs_true_
↪color_img[-1].shape))
```

```
Single element in the list is of type = <class 'numpy.ndarray'> and has shape (1057,␣
↪624, 4)
```

[48]:
```
plot_image(wcs_true_color_img[-1])
```



Custom URL Parameters

Sentinel Hub OGC services have various custom URL parameters described at the webpage. Many of them are supported in this package and some of them might be added in the future. Let's check which ones currently exist.

[49]:
```
from sentinelhub import CustomUrlParam

list(CustomUrlParam)
```

[49]:
```
[<CustomUrlParam.SHOWLOGO: 'ShowLogo'>,
 <CustomUrlParam.ATMFILTER: 'AtmFilter'>,
 <CustomUrlParam.EVALSCRIPT: 'EvalScript'>,
 <CustomUrlParam.EVALSCRIPTURL: 'EvalScriptUrl'>,
 <CustomUrlParam.PREVIEW: 'Preview'>,
 <CustomUrlParam.QUALITY: 'Quality'>,
 <CustomUrlParam.UPSAMPLING: 'Upsampling'>,
 <CustomUrlParam.DOWNSAMPLING: 'Downsampling'>,
 <CustomUrlParam.TRANSPARENT: 'Transparent'>,
 <CustomUrlParam.BGCOLOR: 'BgColor'>,
 <CustomUrlParam.GEOMETRY: 'Geometry'>,
 <CustomUrlParam.MINQA: 'MinQA'>]
```

Many of these parameters already appear in Sentinel Hub Configurator as a property of an instance or a layer. However any parameter we specify in the code will automatically override the definition in Configurator for our request.

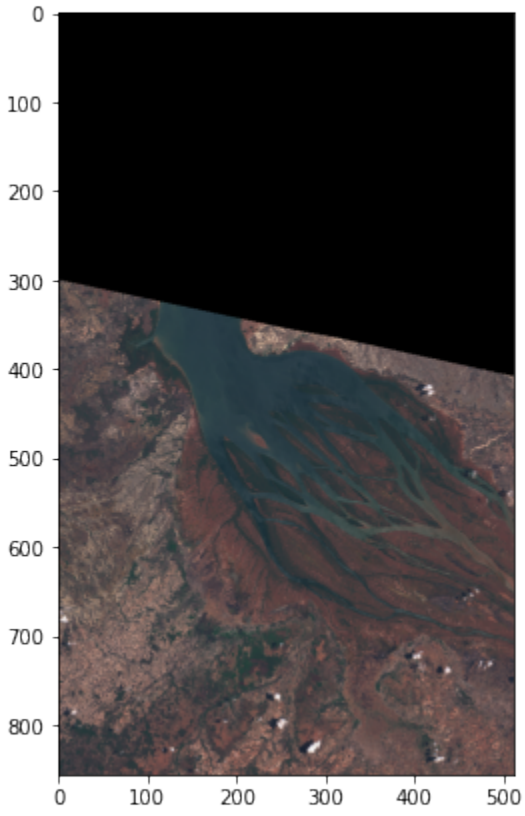
Example 10: Using Custom URL Parameters

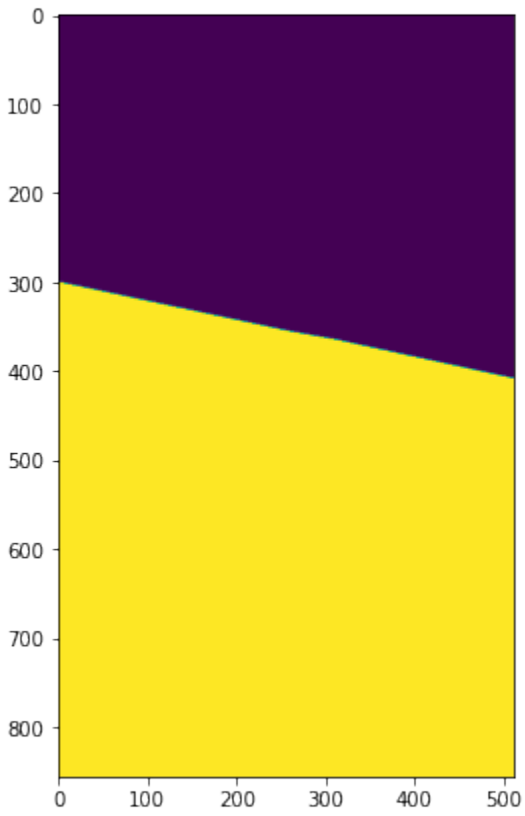
We can request true color image with atmospheric correction, transparency layer and and no logo.

[50]:
```
custom_wms_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='TRUE-COLOR-S2-L1C',
    bbox=betsiboka_bbox,
    time='2019-11-05',
    width=512,
    height=856,
    custom_url_params={
        CustomUrlParam.TRANSPARENT: True,
        CustomUrlParam.SHOWLOGO: False
    },
    config=config
)

custom_wms_data = custom_wms_request.get_data()
```

Obtained true color images have a transparency channel indicating which parts of the image have no data.

[51]:
```
plot_image(custom_wms_data[-1][:, :, :3])
plot_image(custom_wms_data[-1][:, :, 3])
```


0
100
200
300
400
500
600
700
800
0
100
200
300
400
500

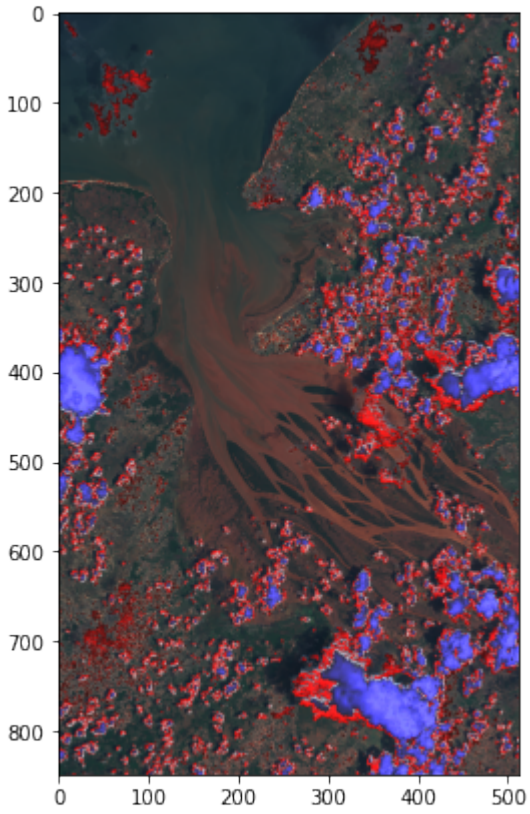

0
100
200
300
400
500
600
700
800
0
100
200
300
400
500

Example 11: Evalscript

Instead of using Sentinel Hub Configurator we can define our own custom layers inside Python with `CustomUrlParam.EVALSCRIPT`. All we need is a chunk of code written in Javascript which is not too long to fit into an URL.

Let's implement a simple cloud detection algorithm.

[52]:
```
# by Braaten, Cohen, Yang 2015
my_evalscript = '''
var bRatio = (B01 - 0.175) / (0.39 - 0.175);
var NGDR = (B01 - B02) / (B01 + B02);

function clip(a) {
    return a>0 ? (a<1 ? a : 1) : 0;
}

if (bRatio > 1) {
    var v = 0.5*(bRatio - 1);
    return [0.5*clip(B04), 0.5*clip(B03), 0.5*clip(B02) + v];
}

if (bRatio > 0 && NGDR > 0) {
    var v = 5 * Math.sqrt(bRatio * NGDR);
    return [0.5 * clip(B04) + v, 0.5 * clip(B03), 0.5 * clip(B02)];
}

return [2*B04, 2*B03, 2*B02];
'''

evalscript_wms_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='TRUE-COLOR-S2-L1C', # Layer parameter can be any existing Sentinel-2 L1C␣
↪layer
    bbox=betsiboka_bbox,
    time='2017-12-20',
    width=512,
    custom_url_params={CustomUrlParam.EVALSCRIPT: my_evalscript},
    config=config
)

evalscript_wms_data = evalscript_wms_request.get_data()
plot_image(evalscript_wms_data[0])
```



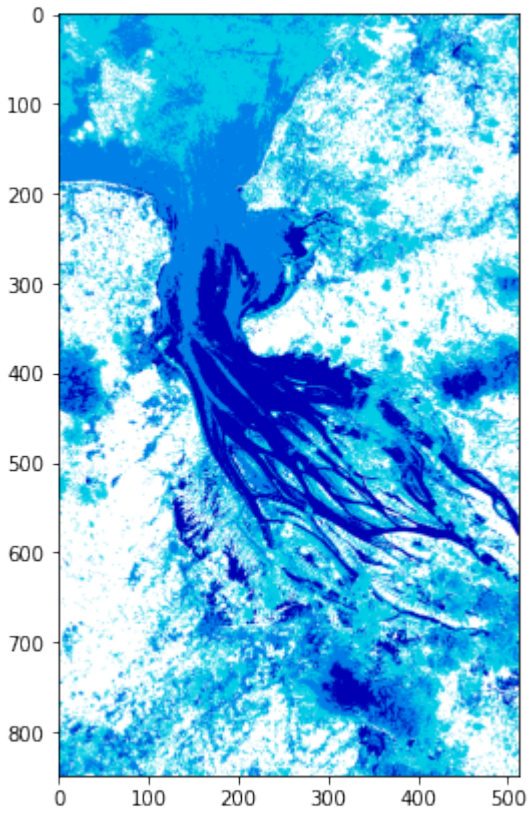
Note: We still had to specify an existing layer from Configurator. That is because each layer is linked with it's data collection and we cannot override layer's data collection from the code.

Example 12: Evalscript URL

Another option is to simply provide an URL address of an evalscript written in Javascript. For that purpose we have created a collection of useful custom scripts on Github.

Let's select a script for calculating moisture index and provide its URL as a value of parameter `CustomUrlParam.EVALSCRIPTURL`.

[53]:
```
my_url = 'https://raw.githubusercontent.com/sentinel-hub/custom-scripts/master/
↪sentinel-2/ndmi_special/script.js'

evalscripturl_wms_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='TRUE-COLOR-S2-L1C', # Layer parameter can be any existing Sentinel-2 L1C␣
↪layer
    bbox=betsiboka_bbox,
    time='2017-12-20',
    width=512,
    custom_url_params={CustomUrlParam.EVALSCRIPTURL: my_url},
    config=config
)

evalscripturl_wms_data = evalscripturl_wms_request.get_data()
plot_image(evalscripturl_wms_data[0])
```



Data Collections

The package supports various data collections. Default data collection is Sentinel-2 L1C however currently the following is supported:

[54]:
```
for collection in DataCollection.get_available_collections():
    print(collection)
```

```
DataCollection.SENTINEL2_L1C
DataCollection.SENTINEL2_L2A
DataCollection.SENTINEL1
DataCollection.SENTINEL1_IW
DataCollection.SENTINEL1_IW_ASC
DataCollection.SENTINEL1_IW_DES
DataCollection.SENTINEL1_EW
DataCollection.SENTINEL1_EW_ASC
DataCollection.SENTINEL1_EW_DES
DataCollection.SENTINEL1_EW_SH
DataCollection.SENTINEL1_EW_SH_ASC
DataCollection.SENTINEL1_EW_SH_DES
DataCollection.DEM
DataCollection.MODIS
DataCollection.LANDSAT8
DataCollection.SENTINEL5P
DataCollection.SENTINEL3_OLCI
DataCollection.SENTINEL3_SLSTR
```

In order to obtain data from any of these data collections with `WmsRequest` or `WcsRequest` we have to do the following:

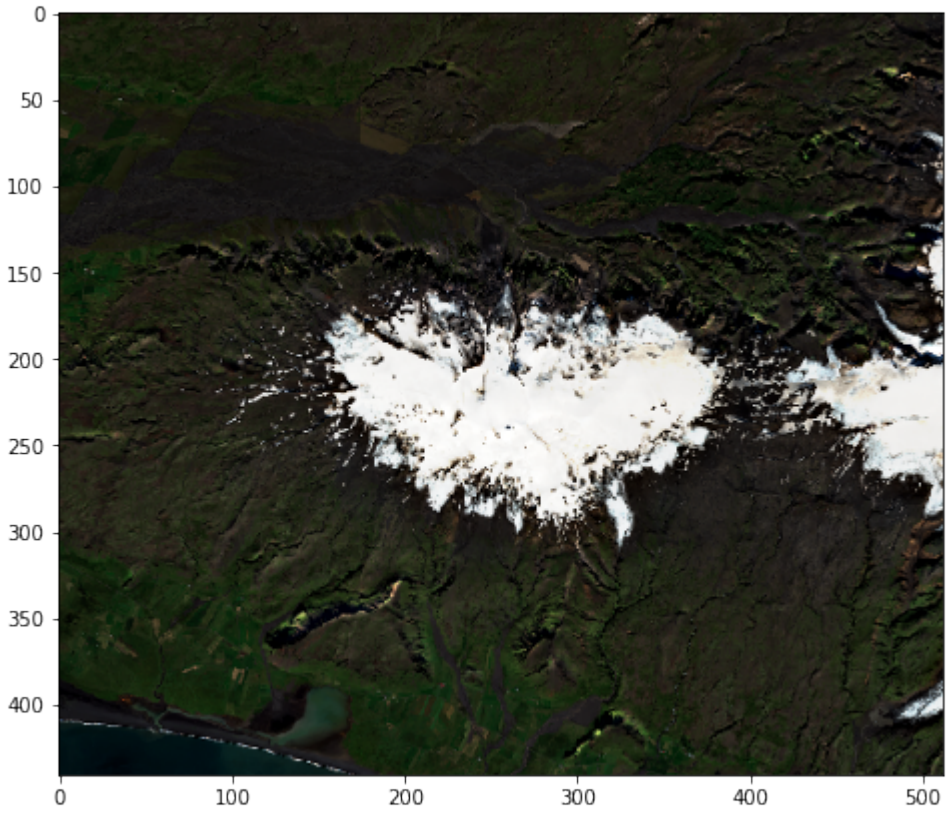
- Use a configuration based on **Python scripts template** or create a new layer in Sentinel Hub Configurator that is defined to use desired satellite data collection. Set the `layer` parameter of `WmsRequest` or `WcsRequest` to the name of this newly created layer.
- Set `data_collection` parameter of `WmsRequest` or `WcsRequest` to the same data collection (using one of the objects from the list above).

Example 13: Sentinel-2 L2A

When you have a layer named `TRUE-COLOR-S2-L2A` in your configuration let's try to obtain some level 2A images. Unfortunately L2A images are being processed only for some regions around the globe and Betsiboka Estuary is not one of them.

Instead let's check Eyjafjallajökull volcano on Iceland. This time we will provide coordinates in Popular Web Mercator CRS.

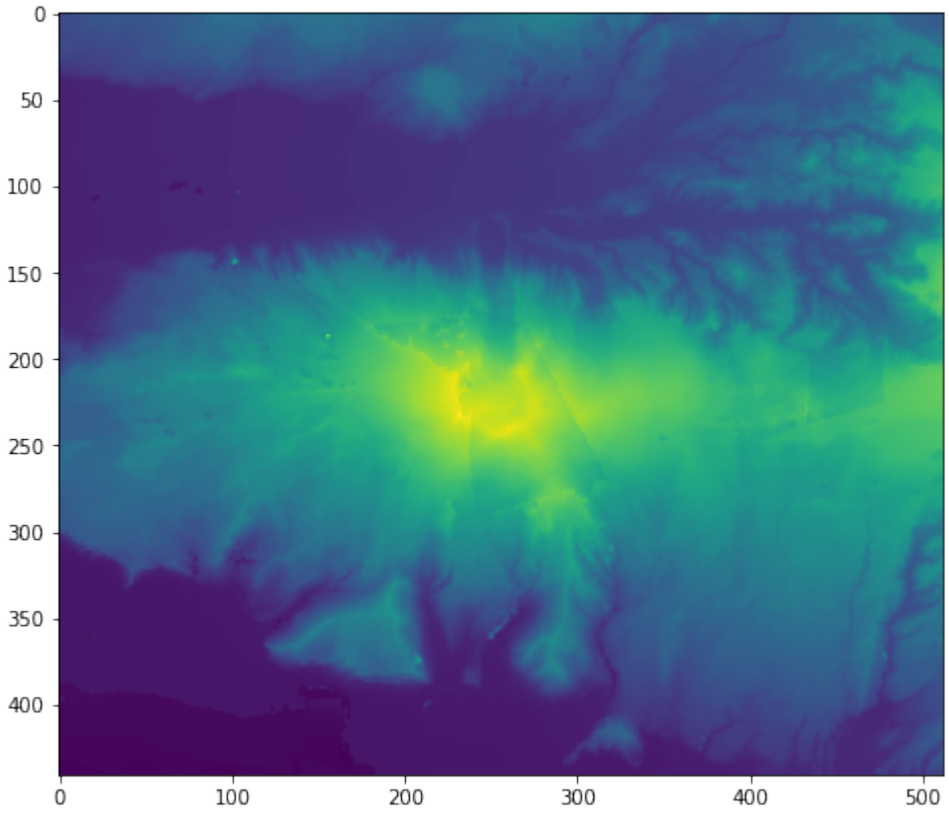
[55]:
```
volcano_bbox = BBox(bbox=[(-2217485.0, 9228907.0), (-2150692.0, 9284045.0)], crs=CRS.
↪POP_WEB)

l2a_request = WmsRequest(
    data_collection=DataCollection.SENTINEL2_L2A,
    layer='TRUE-COLOR-S2-L2A',
    bbox=volcano_bbox,
    time='2017-08-30',
    width=512,
    config=config
)

l2a_data = l2a_request.get_data()
plot_image(l2a_data[0])
```



Example 14: DEM

Request using Mapzen DEM as a data collection does not require a time parameter.

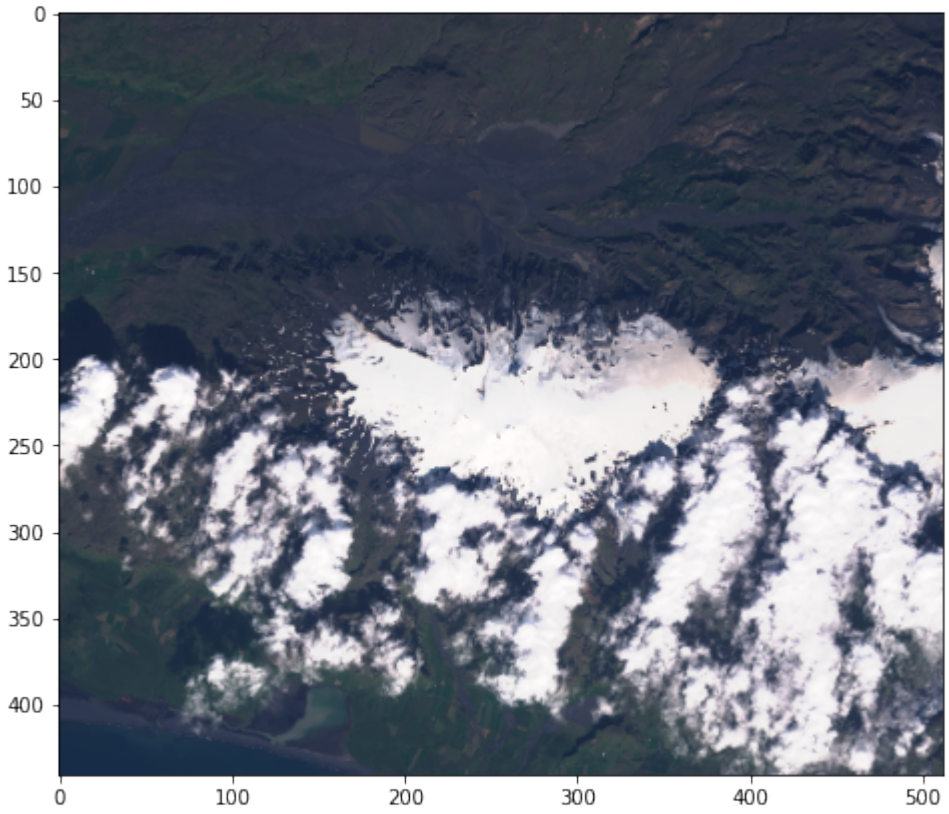
[56]:
```
dem_request = WmsRequest(
    data_collection=DataCollection.DEM,
    layer='DEM',
    bbox=volcano_bbox,
    width=512,
    image_format=MimeType.TIFF,
    custom_url_params={CustomUrlParam.SHOWLOGO: False},
    config=config
)

dem_image = dem_request.get_data()[0]

# Taking the first channel because the second channel is a valid data mask
dem_values = dem_image[..., 0]

plot_image(dem_values, 1 / np.amax(dem_values))
```



Example 15: Landsat 8

To view Landsat 8 L1C image we require a layer `TRUE-COLOR-L8` with predefined true color RGB template.

[57]:
```
l8_request = WmsRequest(
    data_collection=DataCollection.LANDSAT8,
    layer='TRUE-COLOR-L8',
    bbox=volcano_bbox,
    time='2017-08-20',
    width=512,
    config=config
)

l8_data = l8_request.get_data()
plot_image(l8_data[-1])
```



Example 15: Sentinel-1

If we would like to avoid clouds using Sentinel-1 radar data seems like a good idea. The package supports obtaining multiple types of Sentinel-1 data. In this example we will use `DataCollection.SENTINEL1_IW`. While creating layer in Sentinel Hub Configurator we must be careful to use the same settings as the supported data collection:

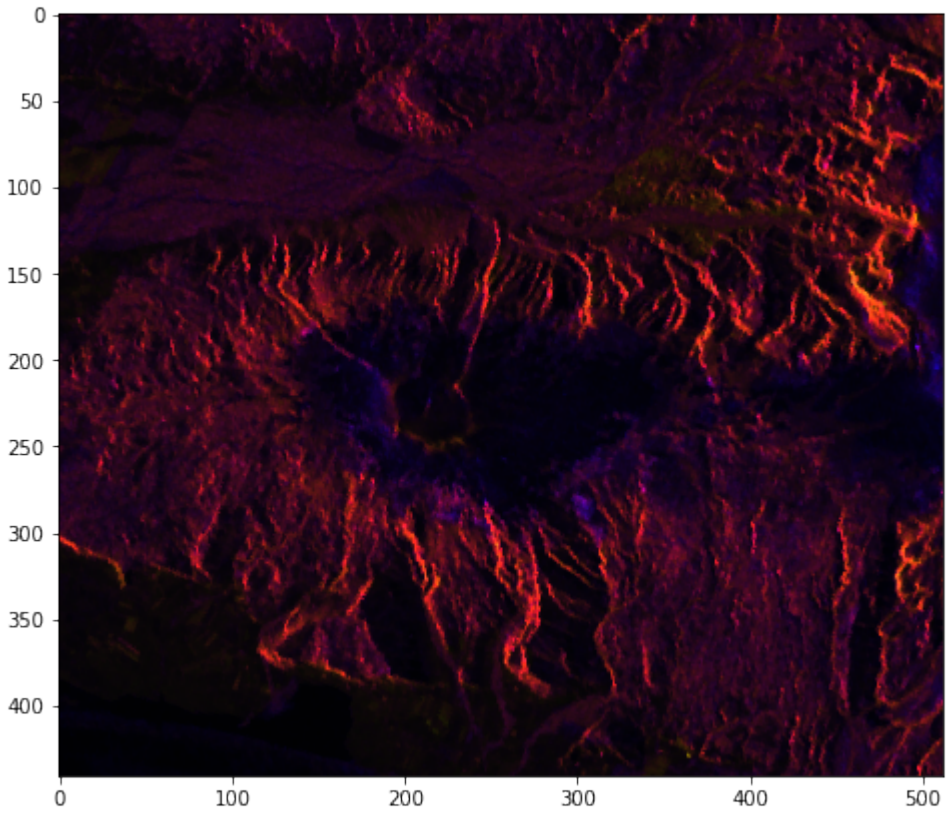
[58]:
```
DataCollection.SENTINEL1_IW
```

[58]:
```
<DataCollection.SENTINEL1_IW: DataCollectionDefinition(
  api_id: S1GRD
  wfs_id: DSS3
  collection_type: Sentinel-1
  sensor_type: C-SAR
  processing_level: GRD
  swath_mode: IW
  polarisation: DV
  resolution: HIGH
  orbit_direction: both
  bands: ('VV', 'VH')
  is_timeless: False
)>
```

This tells us we have to set acquisition parameter to `IW`, polarisation to `DV`, resolution to `High` and orbit direction to `Both`. After that let's name the layer `TRUE-COLOR-S1-IW` and use the following custom script

```
return [VV, 2 * VH, VV / VH / 100.0]
```

[59]:
```
s1_request = WmsRequest(
    data_collection=DataCollection.SENTINEL1_IW,
    layer='TRUE-COLOR-S1-IW',
    bbox=volcano_bbox,
    time='2017-10-03',
    width=512,
    config=config
)

s1_data = s1_request.get_data()
plot_image(s1_data[-1])
```



Example 16: Sentinel-1, ascending orbit direction

Sentinel-1 data is acquired when a satellite travels either from approx. north to south (i.e. `DESCENDING` orbit direction) or from approx. south to north (i.e.‘‘ASCENDING‘‘ orbit direction). With sentinelhub-py package one can request only the data with `ASCENDING` orbit direction if using a data collection ending with “_ASC” (or only data with `DESCENDING` orbit direction when using data collection ending with “_DES”).

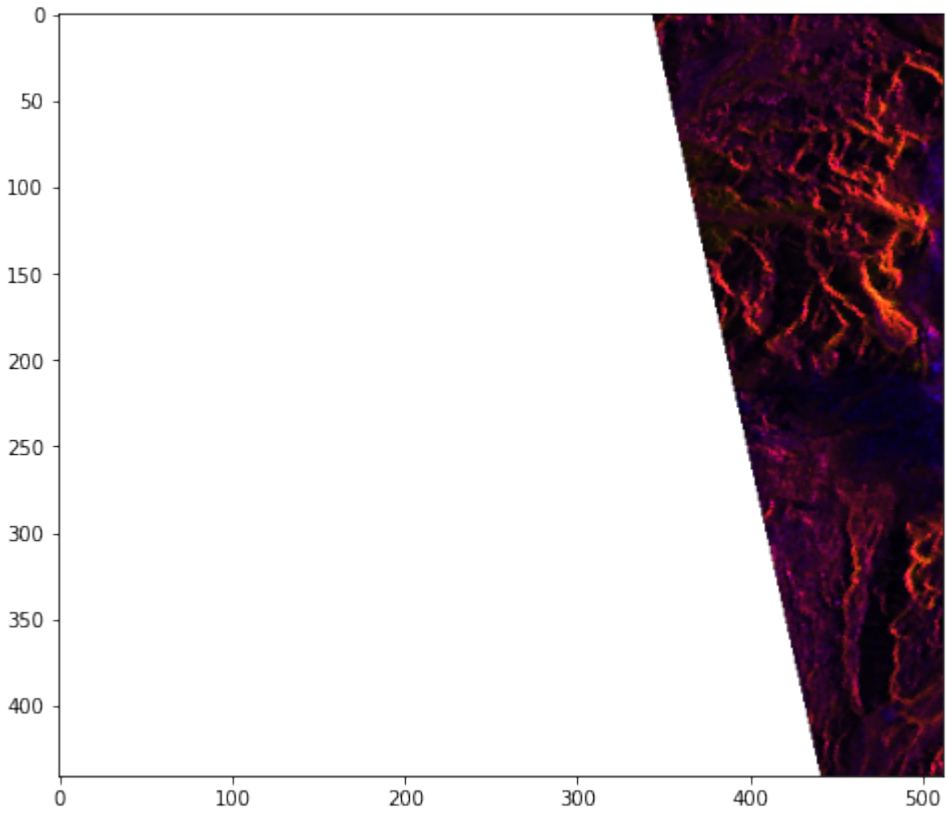
E.g., the request below will fetch only the data with `ASCENDING` orbit direction:

[60]:
```
s1_asc_request = WmsRequest(
    data_collection=DataCollection.SENTINEL1_IW_ASC,
    layer='TRUE-COLOR-S1-IW',
    bbox=volcano_bbox,
    time=('2017-10-03','2017-10-05'),
    width=512,
```

(continues on next page)

(continued from previous page)

```
    config=config
)

s1_asc_data = s1_asc_request.get_data()
plot_image(s1_asc_data[-1])
```



Example 17: Custom (BYOC) data

It is possible that you "bring your own data" (BYOC) and access it using Sentinel Hub in a similar manner as any other satellite data. To be able to access your own data using Sentinel Hub you will need to prepare a few things. Roughly speaking these are (find all details here: https://docs.sentinel-hub.com/api/latest/#/API/byoc): - Convert your data to Cloud Optimized GeoTiff (COG) format. Store it in AWS S3 bucket and allow SH to access it. - Create collection in SH, which points to the S3 bucket. Within the collection, create tiles. - Create new layer in your configuration, which points to the collection. - Request the data using SentinelHub.py package

To demonstrate this, we have prepeared a demo S3 bucket, demo collection and demo layer in our configuration. To access this data we need collection id = '31df1de4-8bd4-43e0-8c3f-b04262d111b6' and layer id = 'DEMO_BYOC_LAYER'. Let see how to request the custom data.

[ ]:
```
byoc_bbox = BBox([13.82387,45.85221, 13.83313, 45.85901], crs=CRS.WGS84)

collection_id = '<your collection id>'
layer = 'DEMO_BYOC_LAYER'

byoc_request = WmsRequest(
    data_collection=DataCollection.define_byoc(collection_id),
```

(continues on next page)

(continued from previous page)

```
    layer=layer,
    bbox=byoc_bbox,
    width=512,
    config=config
)

byoc_data = byoc_request.get_data()
plot_image(byoc_data[-1])
```

3.4 Large area utilities

Many times we would like to run algorithms for large geographical areas. However due to large amount of data we probably won't have enough working memory to do that all at once. So we need to split the area into smaller parts and properly manage them. To make this easier `sentinelhub` package implements utilities for splitting areas into smaller bounding boxes.

3.4.1 Prerequisites

Imports

[1]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline

import itertools

import numpy as np
from shapely.geometry import shape, Polygon, MultiPolygon, MultiLineString

from sentinelhub import BBoxSplitter, OsmSplitter, TileSplitter, CustomGridSplitter,␣
↪UtmZoneSplitter, UtmGridSplitter
from sentinelhub import BBox, read_data, CRS, DataCollection
```

The following packages are not included or required for `sentinelhub` package however they are used in this example notebook. In case you would like to reproduce some visualizations you will have to install them separately.

[2]:
```
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon as plt_polygon

from mpl_toolkits.basemap import Basemap  # Available here: https://github.com/
↪matplotlib/basemap
```

Collecting data

For start we need a geometry of the area of interest. The `sentinelhub` package uses `shapely` package to work with geometries. Therefore any geometry inputs need to be an instance of `shapely.geometry.multipolygon.MultiPolygon` or `shapely.geometry.polygon.Polygon`.

In these examples we will use a geometry of Hawaii islands.

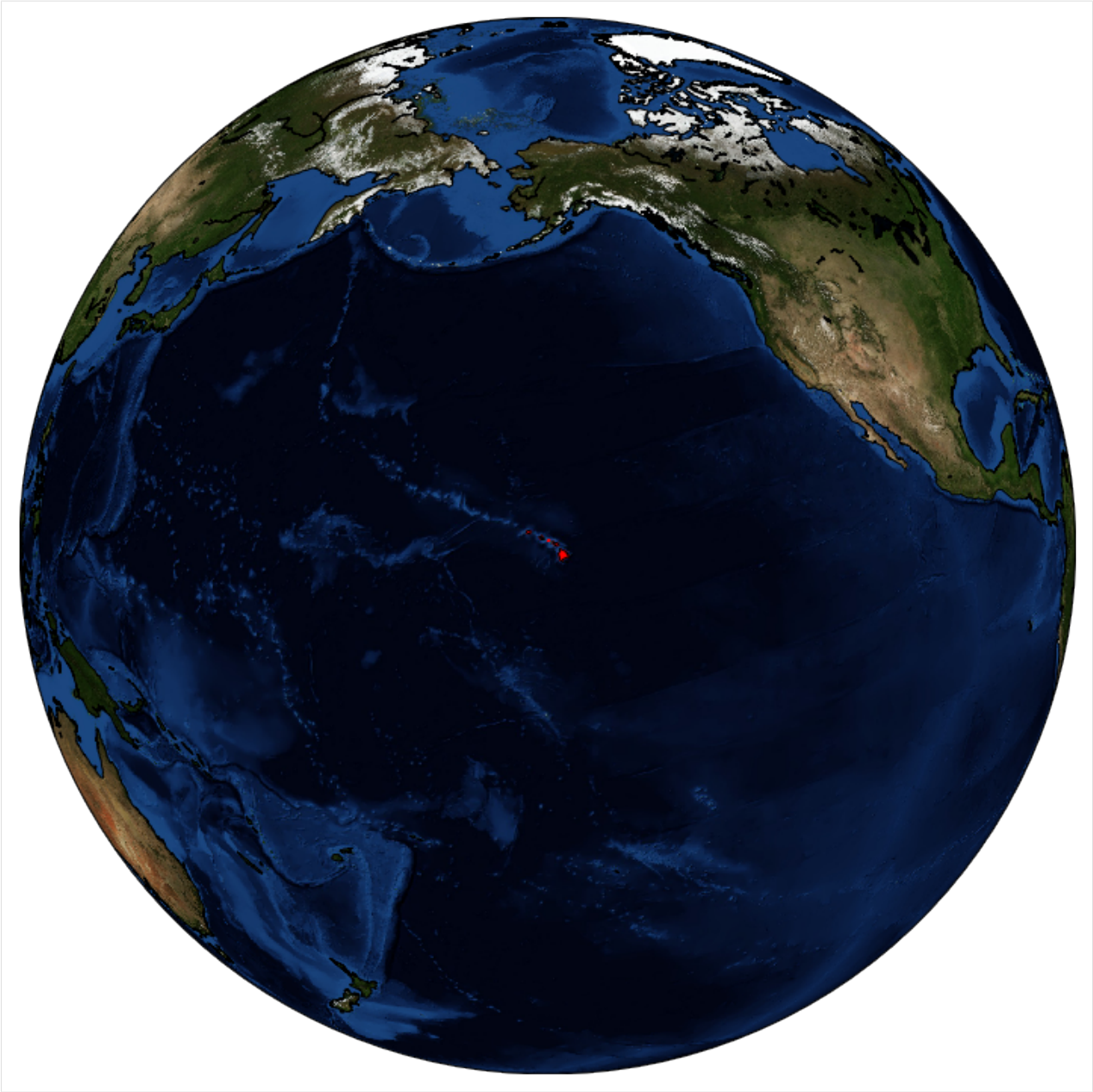
[3]:
```
INPUT_FILE = './data/Hawaii.json'

geo_json = read_data(INPUT_FILE)
hawaii_area = shape(geo_json["features"][0]["geometry"])

type(hawaii_area)
```

[3]:
```
shapely.geometry.multipolygon.MultiPolygon
```

Let's check how the area looks on the world map.

[4]:
```
def show_area(area_shape, area_buffer=0.3):
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111)

    minx, miny, maxx, maxy = area_shape.bounds
    lng, lat = (minx + maxx) / 2, (miny + maxy) / 2

    m = Basemap(projection='ortho',lat_0=lat, lon_0=lng, resolution='l')
    m.drawcoastlines()
    m.bluemarble()

    if isinstance(area_shape, Polygon):
        area_shape = [area_shape]
    for polygon in area_shape:
        x, y = np.array(polygon.boundary)[0]
        m_poly = []
        for x, y in np.array(polygon.boundary):
            m_poly.append(m(x, y))
        ax.add_patch(plt_polygon(np.array(m_poly), closed=True, facecolor='red',␣
↪edgecolor='red'))

    plt.tight_layout()
    plt.show()

show_area(hawaii_area)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or␣
↪[0..255] for integers).
```



3.4.2 Area Splitting

We would like to split the area into smaller bounding boxes which could them be used for obtaining data by calling WMS/WCS requests. The package implements 3 different ways of splitting the area.

Splitting the bounding box

The most straight forward approach is to calculate the area bounding box and split it into smaller parts of equal size.

As an input we need to provide a list of geometries, their CRS, and `int` or `tuple` specifying to how many parts bounding box will be split.

[5]:
```
bbox_splitter = BBoxSplitter([hawaii_area], CRS.WGS84, (5, 4))  # bounding box will␣
↪be split into grid of 5x4 bounding boxes

print('Area bounding box: {}\n'.format(bbox_splitter.get_area_bbox().__repr__()))

bbox_list = bbox_splitter.get_bbox_list()
info_list = bbox_splitter.get_info_list()

print('Each bounding box also has some info how it was created.\nExample:\n'
      'bbox: {}\ninfo: {}\n'.format(bbox_list[0].__repr__(), info_list[0]))
```

```
Area bounding box: BBox(((-159.764448, 18.948267), (-154.807817, 22.228955)), crs=CRS(
↪'4326'))

Each bounding box also has some info how it was created.
Example:
bbox: BBox(((-159.764448, 21.408783), (-158.77312179999998, 22.228955)), crs=CRS('4326
↪'))
info: {'parent_bbox': BBox(((-159.764448, 18.948267), (-154.807817, 22.228955)),␣
↪crs=CRS('4326')), 'index_x': 0, 'index_y': 3}
```

Besides the list of bounding boxes it is also possible to get a list of geometries (i.e. intersection of each bounding box with entire area of interest).

[6]:
```
geometry_list = bbox_splitter.get_geometry_list()

geometry_list[0]
```

[6]:

In order to visualize the splits let's use the following function

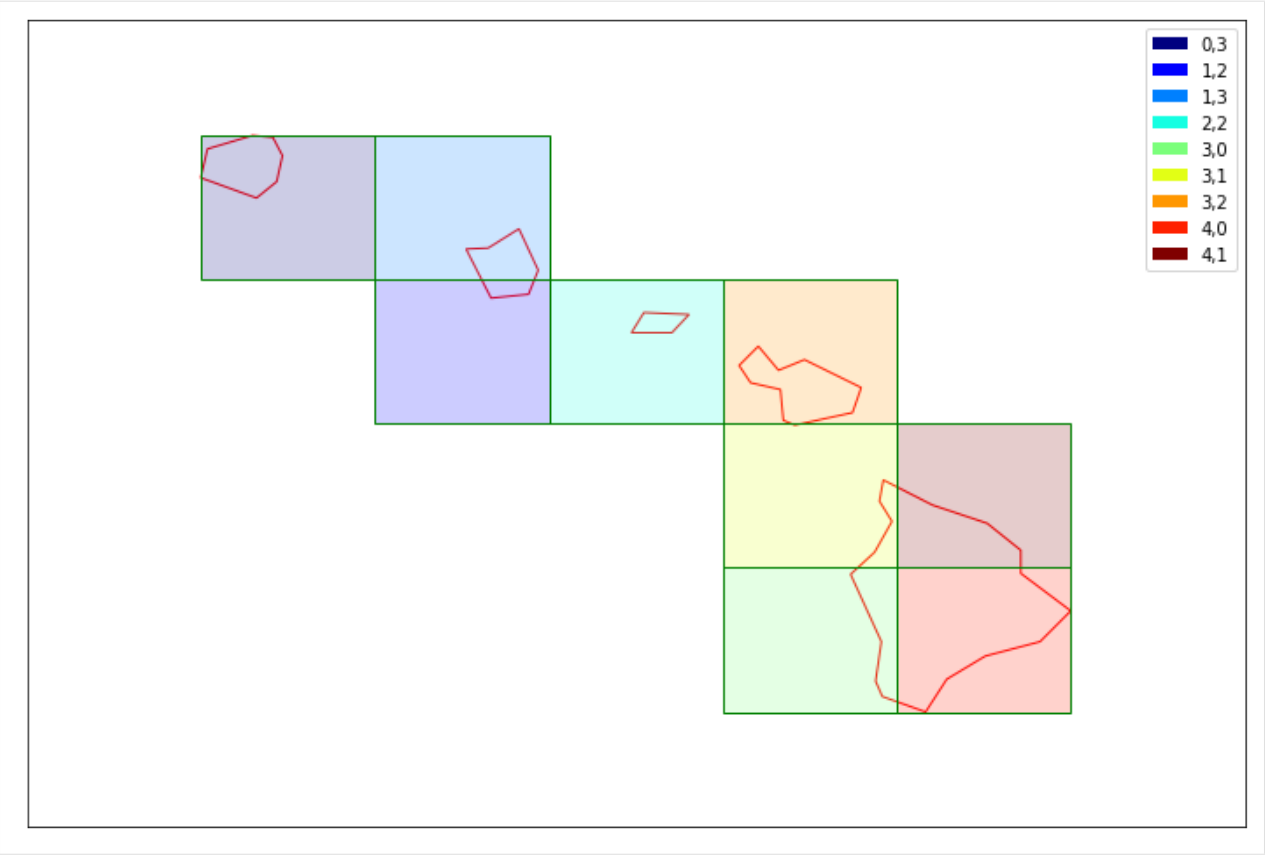
[7]:
```
def show_splitter(splitter, alpha=0.2, area_buffer=0.2, show_legend=False):
    area_bbox = splitter.get_area_bbox()
    minx, miny, maxx, maxy = area_bbox
    lng, lat = area_bbox.middle
    w, h = maxx - minx, maxy - miny
    minx = minx - area_buffer * w
    miny = miny - area_buffer * h
    maxx = maxx + area_buffer * w
    maxy = maxy + area_buffer * h

    fig=plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111)

    base_map = Basemap(projection='mill', lat_0=lat, lon_0=lng, llcrnrlon=minx,␣
↪llcrnrlat=miny,
                       urcrnrlon=maxx, urcrnrlat=maxy, resolution='l', epsg=4326)
    base_map.drawcoastlines(color=(0, 0, 0, 0))

    area_shape = splitter.get_area_shape()
    if isinstance(area_shape, Polygon):
        area_shape = [area_shape]
    for polygon in area_shape:
        if isinstance(polygon.boundary, MultiLineString):
            for linestring in polygon.boundary:
                ax.add_patch(plt_polygon(np.array(linestring), closed=True,␣
↪facecolor=(0, 0, 0, 0), edgecolor='red'))
```

(continues on next page)

(continued from previous page)

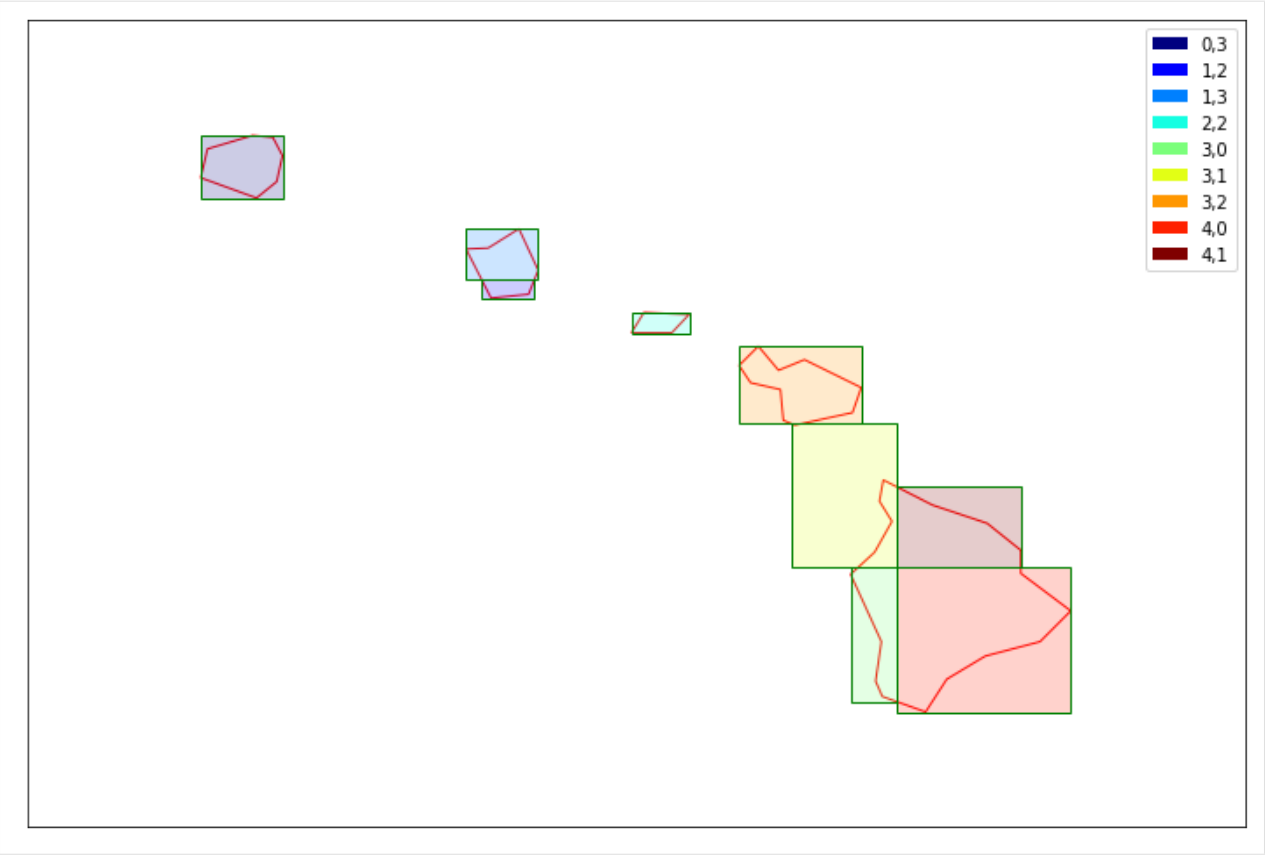
```
        else:
            ax.add_patch(plt_polygon(np.array(polygon.boundary), closed=True,␣
↪facecolor=(0, 0, 0, 0), edgecolor='red'))

    bbox_list = splitter.get_bbox_list()
    info_list = splitter.get_info_list()

    cm = plt.get_cmap('jet', len(bbox_list))
    legend_shapes = []
    for i, (bbox, info) in enumerate(zip(bbox_list, info_list)):
        wgs84_bbox = bbox.transform(CRS.WGS84).get_polygon()

        tile_color = tuple(list(cm(i))[:3] + [alpha])
        ax.add_patch(plt_polygon(np.array(wgs84_bbox), closed=True, facecolor=tile_
↪color, edgecolor='green'))

        if show_legend:
            legend_shapes.append(plt.Rectangle((0,0),1,1, fc=cm(i)))

    if show_legend:
        legend_names = []
        for info in info_list:
            legend_name = '{},{}'.format(info['index_x'], info['index_y'])

            for prop in ['grid_index', 'tile']:
                if prop in info:
                    legend_name = '{},{}'.format(info[prop], legend_name)

            legend_names.append(legend_name)

        plt.legend(legend_shapes, legend_names)
    plt.tight_layout()
    plt.show()
```

[8]:
```
show_splitter(bbox_splitter, show_legend=True)
```



Splitter automatically removed the bounding boxes that did not intersect with the geometry of Hawaii Islands. However the majority of the area inside bounding boxes is still outside our geometry. Therefore each splitter has also an optional parameter `reduce_bbox_sizes`.

[9]:
```
bbox_splitter_reduced = BBoxSplitter([hawaii_area], CRS.WGS84, (5, 4), reduce_bbox_
↪sizes=True)

show_splitter(bbox_splitter_reduced, show_legend=True)
```



By specifying finer splitting we could even further reduce the total area of bounding boxes.

Splitting in OSM grid

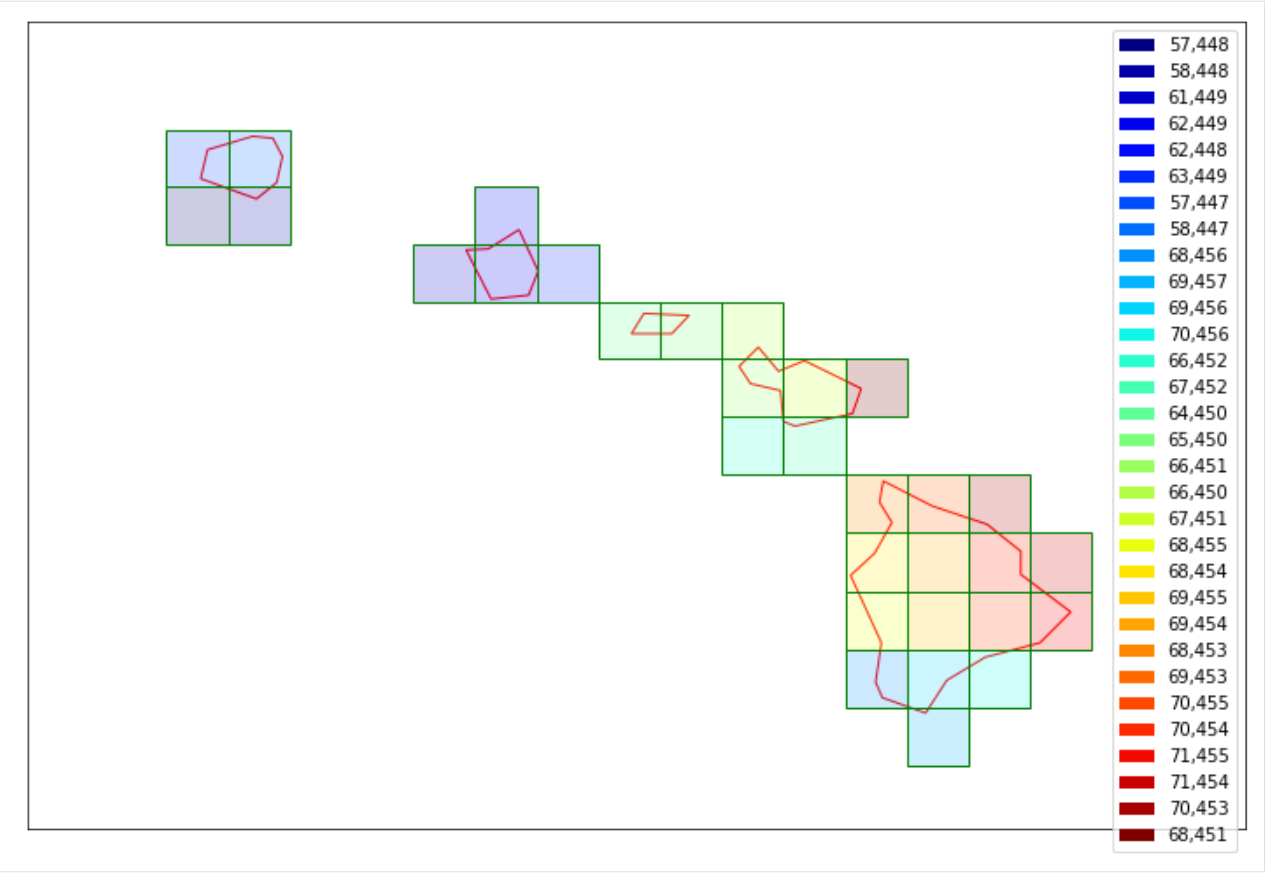
Sometimes it is better to have a splitting grid independent from the given geometries. That way if the geometries at some point slightly change the grid will stay the same.

The following splitter implements Open Street Map's grid.

[10]:
```
osm_splitter = OsmSplitter([hawaii_area], CRS.WGS84, zoom_level=10)

print(repr(osm_splitter.get_bbox_list()[0]))
print(osm_splitter.get_info_list()[0])
```

```
BBox(((-159.96093749999997, 21.616579336740603), (-159.609375, 21.943045533438177)),␣
↪crs=CRS('4326'))
{'zoom_level': 10, 'index_x': 57, 'index_y': 448}
```

[11]:
```
show_splitter(osm_splitter, show_legend=True)
```



Splitting in satellite's tile grid

If we would like to work on a level of satellite tiles and split them we can use the `TileSplitter`. It works in combination with Sentinel Hub WFS service therefore an instance ID is required. We also need to specify `time_interval` and `data_collection`.

[12]:
```
from sentinelhub import SHConfig

INSTANCE_ID = ''  # In case you put instance ID into configuration file you can leave␣
↪this unchanged

if INSTANCE_ID:
    config = SHConfig()
    config.instance_id = INSTANCE_ID
else:
    config = None
```

[13]:
```
tile_splitter = TileSplitter(
    [hawaii_area],
    CRS.WGS84,
    ('2017-10-01', '2018-03-01'),
    data_collection=DataCollection.SENTINEL2_L1C,
    config=config
)
```

(continues on next page)

(continued from previous page)

```
tile_bbox_list = tile_splitter.get_bbox_list()

print(len(tile_bbox_list))
print(tile_bbox_list[0].__repr__())
print(tile_splitter.get_info_list()[0])
```

```
16
BBox(((499980.0, 2290200.0), (609780.0, 2400000.0)), crs=CRS('32604'))
{'parent_bbox': BBox(((499980.0, 2290200.0), (609780.0, 2400000.0)), crs=CRS('32604
↪')), 'index_x': 0, 'index_y': 0, 'tile': '4QEJ'}
```

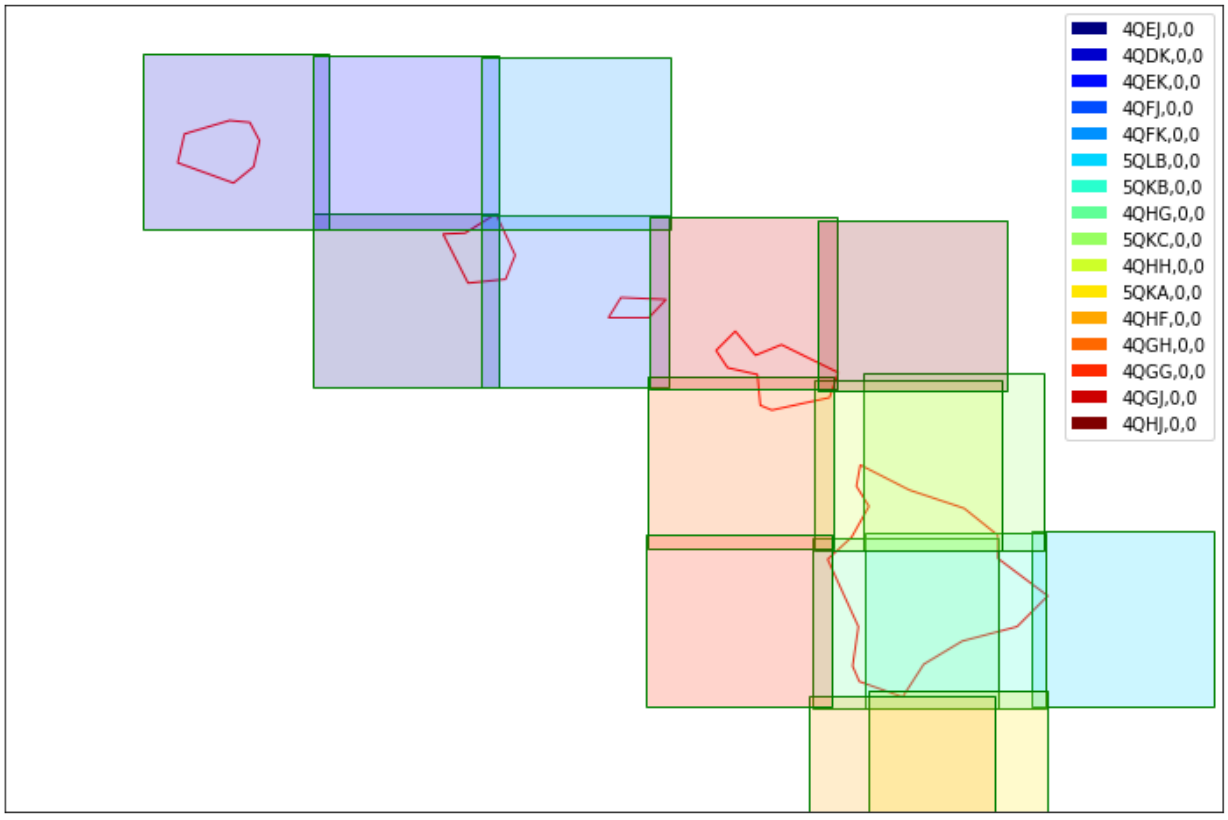
`TileSplitter` by default returns bounding boxes in the satellite tile CRS. In order to transform them we can use `BBox.transform(target_crs)` method or by specifying `crs` parameter in `get_bbox_list` method.

Note: This will only transform bounding box vertices and therefore the new bounding box will not be completely aligned with the original one.

[14]:
```
tile_splitter.get_bbox_list(crs=CRS.WGS84)[0]
```

[14]:

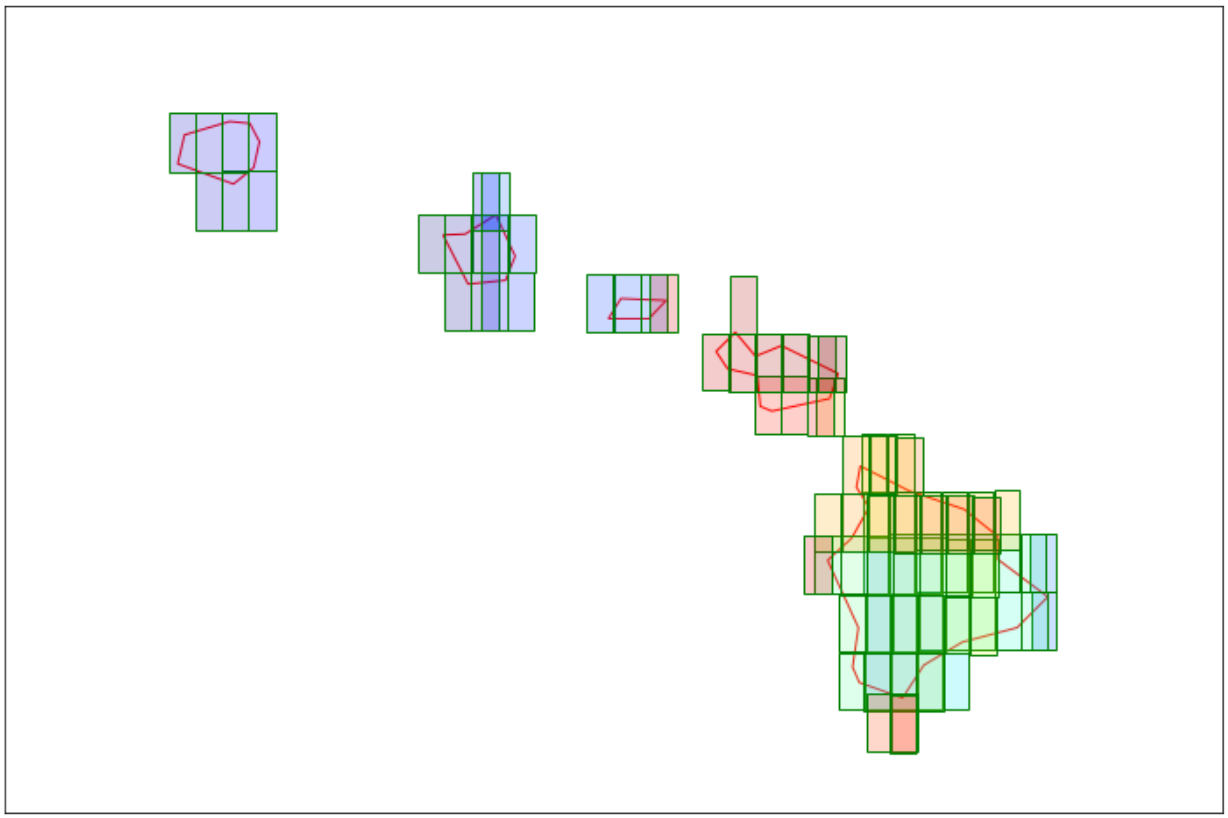
[15]:
```
show_splitter(tile_splitter, show_legend=True)
```



Obtained Sentinel-2 tiles intersect each other. Therefore this splitter is only useful if we are analyzing data on level of original satellite tiles.

We can also specify to further split the satellite tiles.

[16]:
```
finer_tile_splitter = TileSplitter(
    [hawaii_area],
    CRS.WGS84,
    ('2017-10-01', '2018-03-01'),
    tile_split_shape=(7, 3),
    data_collection=DataCollection.SENTINEL2_L1C,
    config=config
)

show_splitter(finer_tile_splitter, show_legend=False)
```



Splitting in custom grid

In case none of the above splitters suffices there is also a `CustomGridSplitter`. All we need is a list of bounding boxes, which define a new way of splitting the area.

The following example is just a split into bounding boxes with integer value latitude and longitude:

[17]:
```
bbox_grid = [BBox((x, y, x + 1, y + 1), CRS.WGS84) for x, y in itertools.
↪product(range(-159, -155), range(18, 23))]

bbox_grid
```

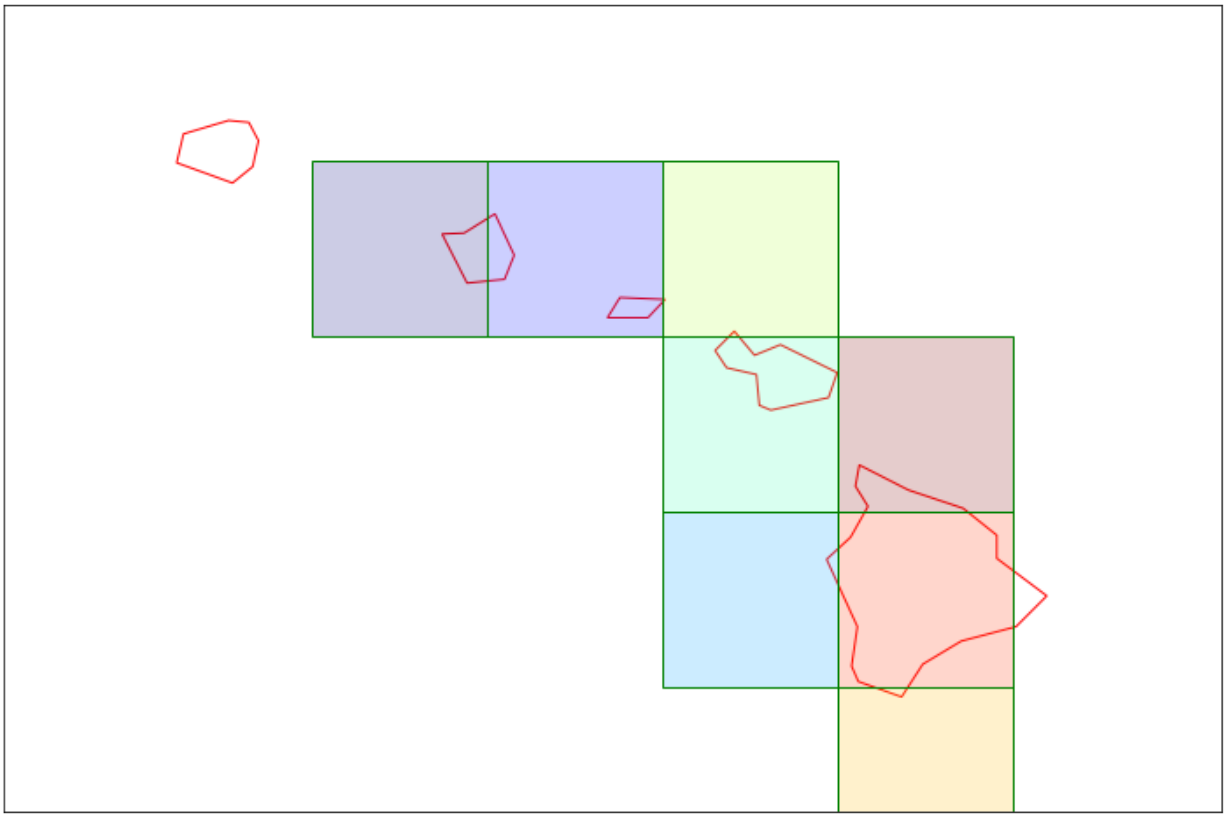
[17]:
```
[BBox(((-159.0, 18.0), (-158.0, 19.0)), crs=CRS('4326')),
 BBox(((-159.0, 19.0), (-158.0, 20.0)), crs=CRS('4326')),
 BBox(((-159.0, 20.0), (-158.0, 21.0)), crs=CRS('4326')),
 BBox(((-159.0, 21.0), (-158.0, 22.0)), crs=CRS('4326')),
```

(continues on next page)

(continued from previous page)

```
BBox(((-159.0, 22.0), (-158.0, 23.0)), crs=CRS('4326')),
BBox(((-158.0, 18.0), (-157.0, 19.0)), crs=CRS('4326')),
BBox(((-158.0, 19.0), (-157.0, 20.0)), crs=CRS('4326')),
BBox(((-158.0, 20.0), (-157.0, 21.0)), crs=CRS('4326')),
BBox(((-158.0, 21.0), (-157.0, 22.0)), crs=CRS('4326')),
BBox(((-158.0, 22.0), (-157.0, 23.0)), crs=CRS('4326')),
BBox(((-157.0, 18.0), (-156.0, 19.0)), crs=CRS('4326')),
BBox(((-157.0, 19.0), (-156.0, 20.0)), crs=CRS('4326')),
BBox(((-157.0, 20.0), (-156.0, 21.0)), crs=CRS('4326')),
BBox(((-157.0, 21.0), (-156.0, 22.0)), crs=CRS('4326')),
BBox(((-157.0, 22.0), (-156.0, 23.0)), crs=CRS('4326')),
BBox(((-156.0, 18.0), (-155.0, 19.0)), crs=CRS('4326')),
BBox(((-156.0, 19.0), (-155.0, 20.0)), crs=CRS('4326')),
BBox(((-156.0, 20.0), (-155.0, 21.0)), crs=CRS('4326')),
BBox(((-156.0, 21.0), (-155.0, 22.0)), crs=CRS('4326')),
BBox(((-156.0, 22.0), (-155.0, 23.0)), crs=CRS('4326'))]
```

[18]:
```
custom_grid_splitter = CustomGridSplitter([hawaii_area], CRS.WGS84, bbox_grid)

show_splitter(custom_grid_splitter)
```



Note that polygons, which are outside of the given collection of bounding boxes, will not affect the tiling.

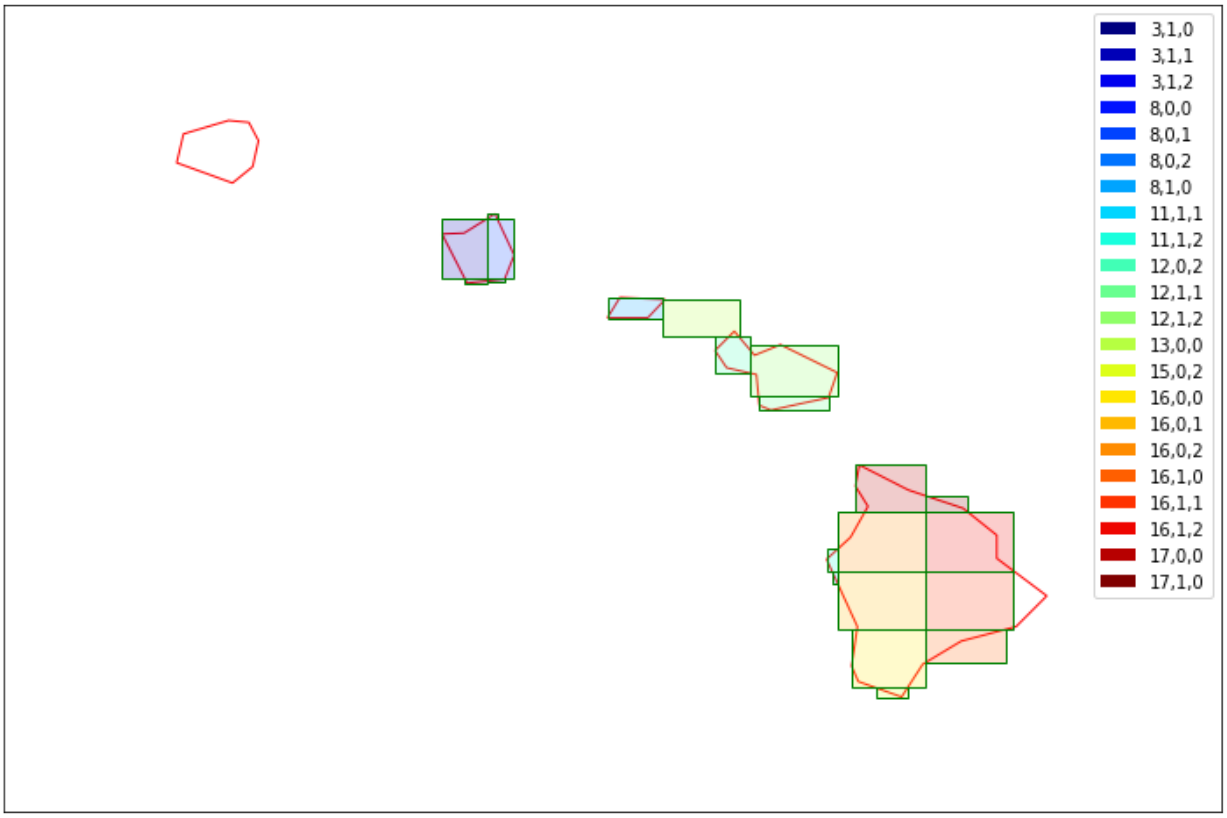
Just like in previous examples we can further split each of the bounding boxes and reduce their size to better fit given shapes.

[19]:
```
finer_custom_grid_splitter = CustomGridSplitter([hawaii_area], CRS.WGS84, bbox_grid,
```

(continues on next page)

(continued from previous page)

```
                                                bbox_split_shape=(2, 3), reduce_bbox_
↪sizes=True)

show_splitter(finer_custom_grid_splitter, show_legend=True)
```

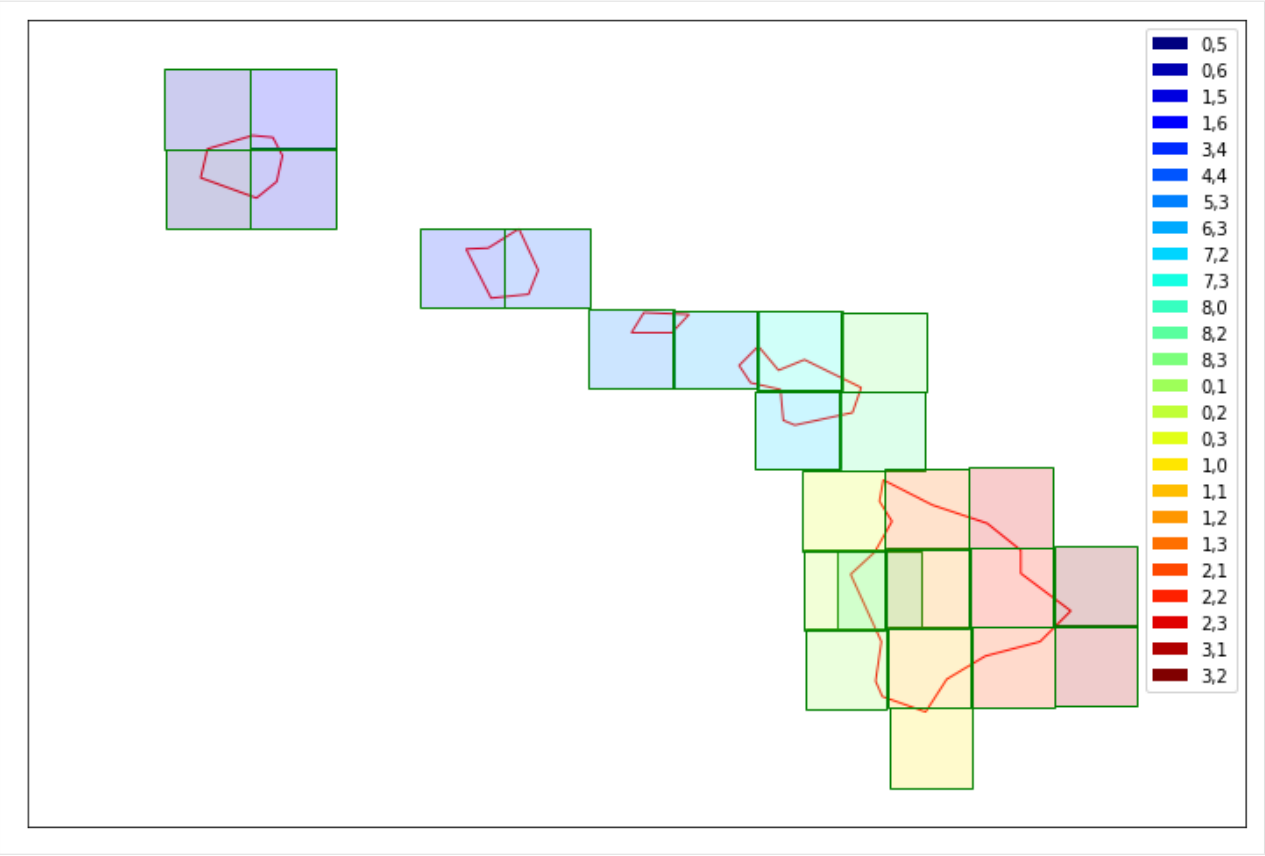


Splitting into UTM grid zones

When dealing with large areas such as countries or continents spanning multiple UTM zones, it might be convenient to split the area according to UTM zones or to the UTM Military Grid Reference System. For these use-cases, a `UtmZoneSplitter` and a `UtmGridSplitter` have been created. These splitters will generate a list of bounding boxes **in UTM CRS**. Each `BBox` will have the CRS corresponding to the UTM zone it belongs to.

To ensure consistency across zones and grid tiles, the size of the `BBox`es in metres is specified as input parameter. For this reason, the option `reduce_bbox_sizes` is not available for these splitters. The two splitters return consistent results between each other, with the exceptions of areas where the UTM grid tiles present exceptions, as in `31V` and `32V`, and `31X`, `33X`, `35X` and `37X`.

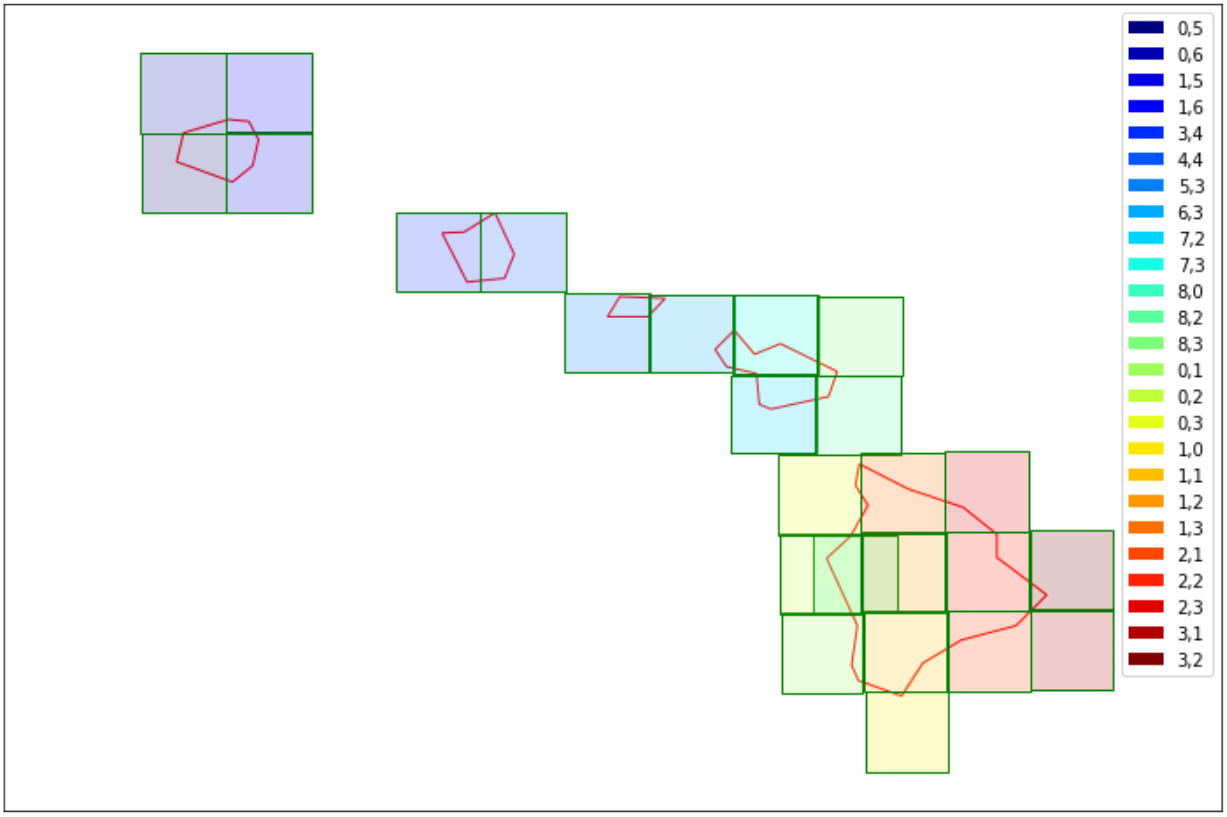
[20]:
```
utm_zone_splitter = UtmZoneSplitter([hawaii_area], CRS.WGS84, (50000, 50000))

show_splitter(utm_zone_splitter, show_legend=True)
```



[21]:
```
utm_zone_splitter.get_bbox_list()
```

[21]:
```
[BBox(((400000.0, 2400000.0), (450000.0, 2450000.0)), crs=CRS('32604')),
 BBox(((400000.0, 2450000.0), (450000.0, 2500000.0)), crs=CRS('32604')),
 BBox(((450000.0, 2400000.0), (500000.0, 2450000.0)), crs=CRS('32604')),
 BBox(((450000.0, 2450000.0), (500000.0, 2500000.0)), crs=CRS('32604')),
 BBox(((550000.0, 2350000.0), (600000.0, 2400000.0)), crs=CRS('32604')),
 BBox(((600000.0, 2350000.0), (650000.0, 2400000.0)), crs=CRS('32604')),
 BBox(((650000.0, 2300000.0), (700000.0, 2350000.0)), crs=CRS('32604')),
 BBox(((700000.0, 2300000.0), (750000.0, 2350000.0)), crs=CRS('32604')),
 BBox(((750000.0, 2250000.0), (800000.0, 2300000.0)), crs=CRS('32604')),
 BBox(((750000.0, 2300000.0), (800000.0, 2350000.0)), crs=CRS('32604')),
 BBox(((800000.0, 2150000.0), (850000.0, 2200000.0)), crs=CRS('32604')),
 BBox(((800000.0, 2250000.0), (850000.0, 2300000.0)), crs=CRS('32604')),
 BBox(((800000.0, 2300000.0), (850000.0, 2350000.0)), crs=CRS('32604')),
 BBox(((150000.0, 2100000.0), (200000.0, 2150000.0)), crs=CRS('32605')),
 BBox(((150000.0, 2150000.0), (200000.0, 2200000.0)), crs=CRS('32605')),
 BBox(((150000.0, 2200000.0), (200000.0, 2250000.0)), crs=CRS('32605')),
 BBox(((200000.0, 2050000.0), (250000.0, 2100000.0)), crs=CRS('32605')),
 BBox(((200000.0, 2100000.0), (250000.0, 2150000.0)), crs=CRS('32605')),
 BBox(((200000.0, 2150000.0), (250000.0, 2200000.0)), crs=CRS('32605')),
 BBox(((200000.0, 2200000.0), (250000.0, 2250000.0)), crs=CRS('32605')),
 BBox(((250000.0, 2100000.0), (300000.0, 2150000.0)), crs=CRS('32605')),
 BBox(((250000.0, 2150000.0), (300000.0, 2200000.0)), crs=CRS('32605')),
 BBox(((250000.0, 2200000.0), (300000.0, 2250000.0)), crs=CRS('32605')),
 BBox(((300000.0, 2100000.0), (350000.0, 2150000.0)), crs=CRS('32605')),
 BBox(((300000.0, 2150000.0), (350000.0, 2200000.0)), crs=CRS('32605'))]
```

[22]:
```
utm_grid_splitter = UtmGridSplitter([hawaii_area], CRS.WGS84, (50000, 50000))

show_splitter(utm_grid_splitter, show_legend=True)
```



3.5 Sentinel Hub Batch Processing

A tutorial about *Large area utilities* shows how to split a large area into smaller bounding boxes for which data can be requested using *Sentinel Hub Processing API*. This tutorial shows another way of doing that.

Sentinel Hub Batch Processing takes the geometry of a large area and divides it according to a specified tile grid. Next, it executes processing requests for each tile in the grid and stores results to a given location at AWS S3 storage. All this is efficiently executed on the server-side. Because of the optimized performance, it is significantly faster than running the same process locally.

More information about batch processing is available at Sentinel Hub documentation pages:

- How Batch API works
- Batch API service description

The tutorial will show a standard process of using Batch Processing with `sentinelhub-py`. The process can be divided into:

1. Define and create a batch request
2. Analyse a batch request before it is executed
3. Run a batch requests job and check the outcome

Imports

The tutorial requires packages `geopandas` and `descartes` which are not dependencies of `sentinelhub-py`.

[1]:
```
%matplotlib inline

import os
import datetime as dt

import geopandas as gpd

from sentinelhub import SentinelHubBatch, SentinelHubRequest, Geometry, CRS,␣
↪DataCollection, \
    MimeType, SHConfig, bbox_to_dimensions

from utils import plot_image
```

3.5.1 1. Create a batch request

To create a batch request we need to do the following:

- Define a Processing API request which we would like to execute on a large area.
- Select a tiling grid which will define how our area will be split into smaller tiles.
- Set up an S3 bucket where results will be saved.

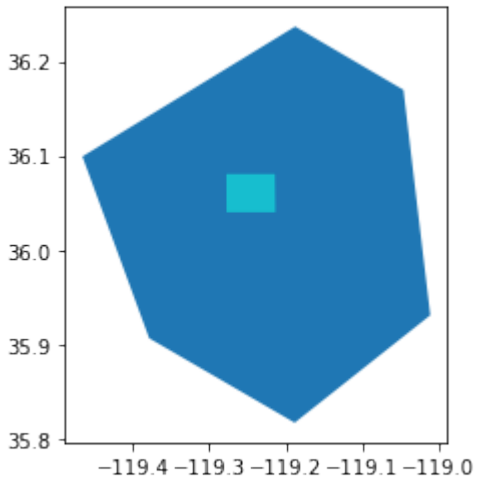
1.1 Define a Processing API request

First, let's set up the credentials the same way as in *Sentinel Hub Processing API tutorial*.

[2]:
```
CLIENT_ID = ''
CLIENT_SECRET = ''

config = SHConfig()

if CLIENT_ID and CLIENT_SECRET:
    config.sh_client_id = CLIENT_ID
    config.sh_client_secret = CLIENT_SECRET
```

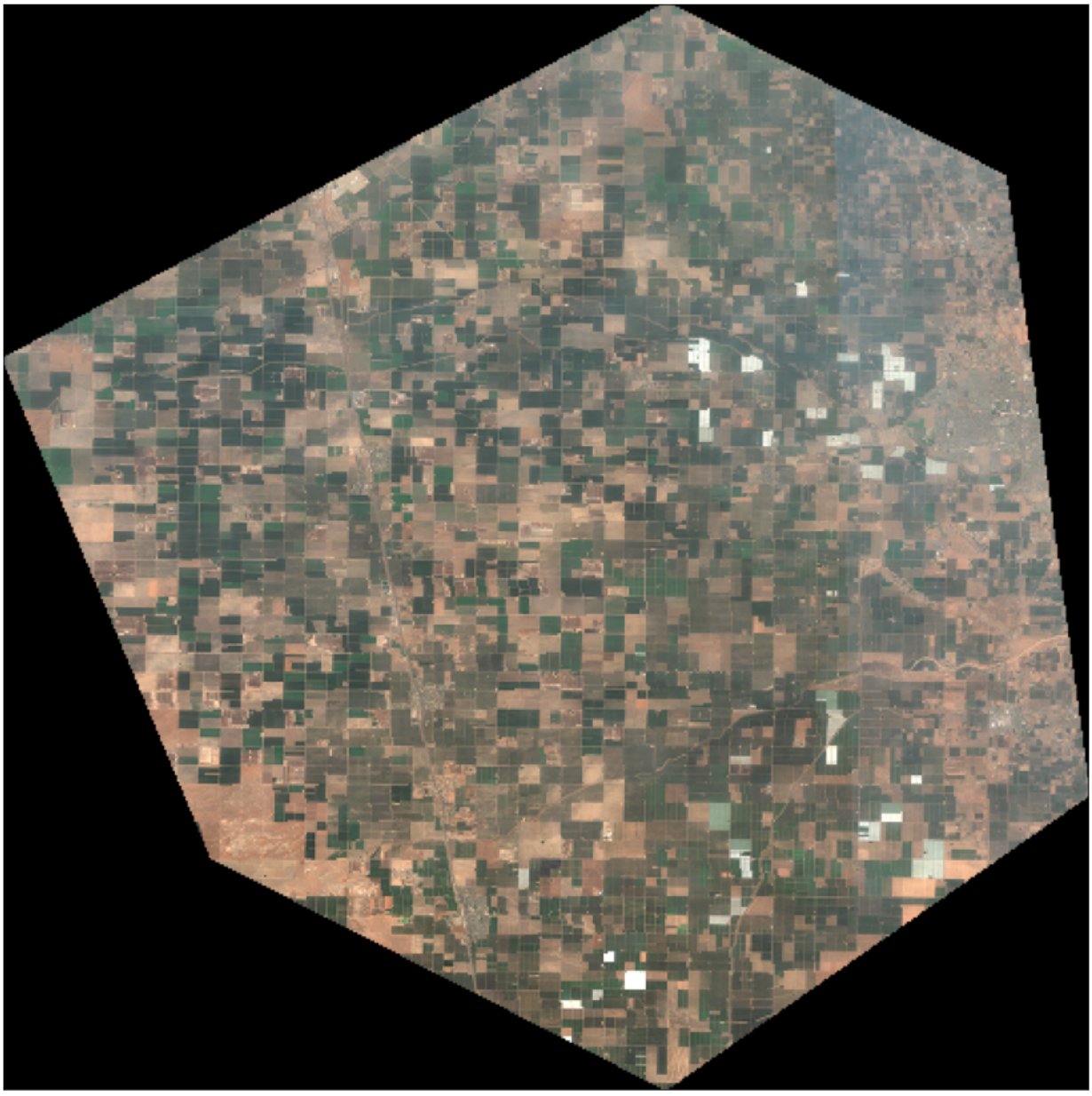
For our area of interest, we'll take an area of crop fields in California.

[3]:
```
SHAPE_PATH = os.path.join('.', 'data', 'california_crop_fields.geojson')
area_gdf = gpd.read_file(SHAPE_PATH)

# Geometry of an entire area
full_geometry = Geometry(area_gdf.geometry.values[0], crs=CRS.WGS84)
# Bounding box of a test sub-area
test_bbox = Geometry(area_gdf.geometry.values[1], crs=CRS.WGS84).bbox

area_gdf.plot(column='name');
```



Let's check a true-color satellite image of the entire area:

[4]:
```
evalscript_true_color = """
    //VERSION=3
    function setup() {
        return {
            input: [{
                bands: ["B02", "B03", "B04"]
            }],
            output: {
                bands: 3
            }
        };
    }
    function evaluatePixel(sample) {
        return [sample.B04, sample.B03, sample.B02];
    }
"""

request = SentinelHubRequest(
    evalscript=evalscript_true_color,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L2A,
        )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.PNG)
    ],
    geometry=full_geometry,
    size=(512, 512),
    config=config
)

image = request.get_data()[0]

plot_image(image, factor=3.5/255, clip_range=(0,1))
```



Next, let's define an evalscript and time range. To better demonstrate the power of batch processing we'll take an evalscript that returns a temporally-interpolated stack NDVI values.

Warning:

In the following cell parameters `evalscript` and `time_interval` are both defined for the same time interval. If you decide to change the time interval you have to change it both in the cell and in the evalscript code.

[5]:
```
EVALSCRIPT_PATH = os.path.join('.', 'data', 'interpolation_evalscript.js')

with open(EVALSCRIPT_PATH, 'r') as fp:
    evalscript = fp.read()
```

(continues on next page)

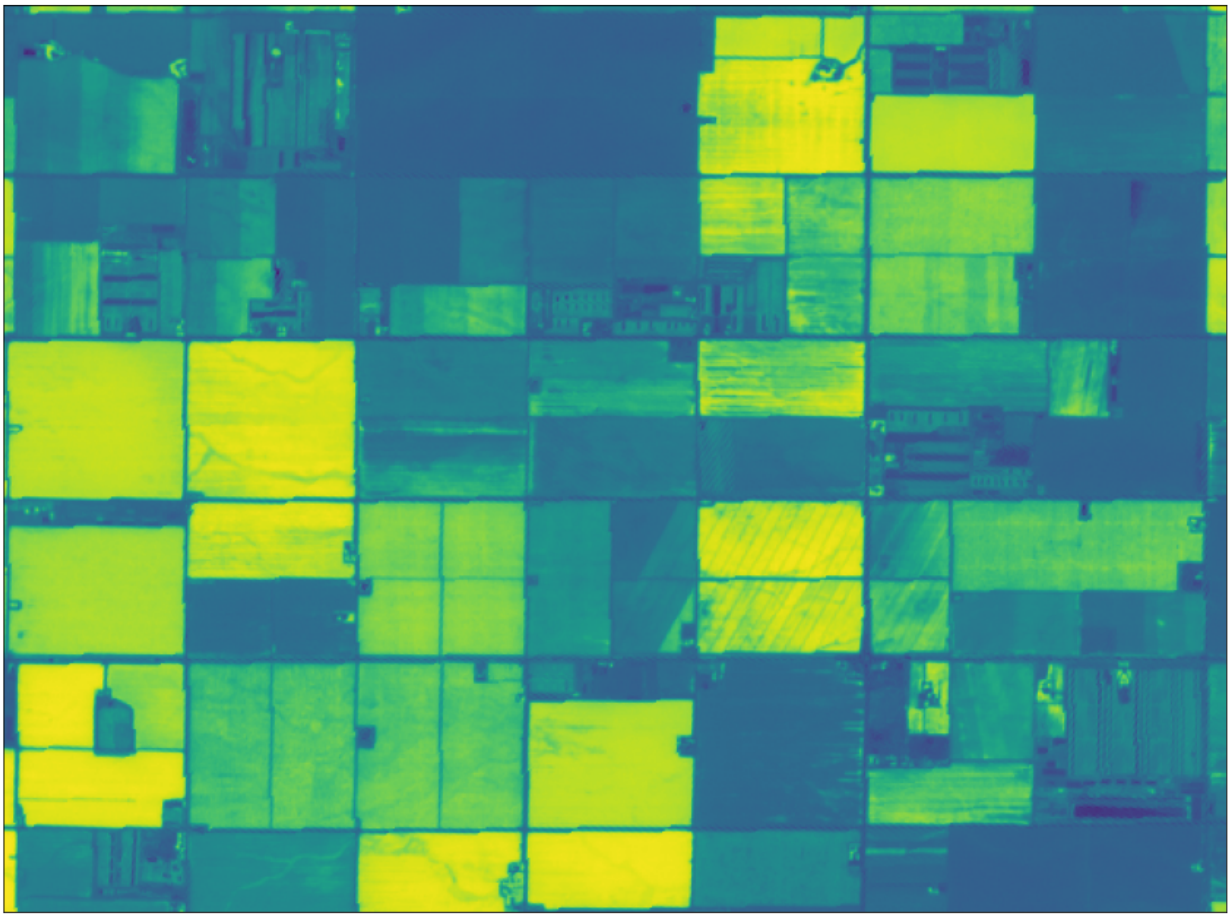
(continued from previous page)

```
time_interval = dt.date(year=2020, month=7, day=1), dt.date(year=2020, month=7,␣
↪day=30)
```

Now we can define a processing API request and test it on a smaller sub-area to make sure we get back desired data.

[6]:
```
%%time

sentinelhub_request = SentinelHubRequest(
    evalscript=evalscript,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L1C,
            time_interval=time_interval,
        )
    ],
    responses=[
        SentinelHubRequest.output_response('NDVI', MimeType.TIFF),
        SentinelHubRequest.output_response('data_mask', MimeType.TIFF)
    ],
    bbox=test_bbox,
    size=bbox_to_dimensions(test_bbox, 10),
    config=config
)

results = sentinelhub_request.get_data()[0]

print(f'Output data: {list(results)}')

plot_image(results['NDVI.tif'][..., 2])
```

```
Output data: ['NDVI.tif', 'data_mask.tif']
CPU times: user 175 ms, sys: 23.4 ms, total: 198 ms
Wall time: 11.7 s
```



We obtained stacks of NDVI values and data masks.

1.2 Select a tiling grid

Sentinel Hub Batch implements multiple different tiling grids. We can check currently available grids through a Python interface offered by `SentinelHubBatch` class.

```
[ ]: list(SentinelHubBatch.iter_tiling_grids(config=config))
```

Let's select a 10km grid, which is based on Sentinel-2 data tiling grid in UTM coordinate reference systems.

There is also an option to check a definition for a single grid:

```
[ ]: # Specify grid ID here:
GRID_ID = 1

SentinelHubBatch.get_tiling_grid(GRID_ID, config=config)
```

1.3 Set up an S3 bucket

For this step please follow instructions on how to configure an S3 bucket in a way that Sentinel Hub service will be able to write to it.

```
[ ]: # Write bucket name here:
BUCKET_NAME = ''
```

1.4 Join batch request definition

Now we are ready to create an entire batch request. This step won't trigger the actual processing. It will only save a batch request definition to the server-side.

```
[ ]: sentinelhub_request = SentinelHubRequest(
    evalscript=evalscript,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L1C,
            time_interval=time_interval,
        )
    ],
    responses=[
        SentinelHubRequest.output_response('NDVI', MimeType.TIFF),
        SentinelHubRequest.output_response('data_mask', MimeType.TIFF)
    ],
    geometry=full_geometry,
    # This time we don't specify size parameter
    config=config
)

batch_request = SentinelHubBatch.create(
    sentinelhub_request,
    tiling_grid=SentinelHubBatch.tiling_grid(
        grid_id=GRID_ID,
        resolution=10,
        buffer=(50, 50)
    ),
    bucket_name=BUCKET_NAME,
    # Check documentation for more about output configuration options:
    # output=SentinelHubBatch.output(...)
    description='sentinelhub-py tutorial batch job',
    config=config
)

batch_request
```

A batch request has been successfully created. From the object representation, we can see some of its main properties, such as `status`, which defines the current status of a batch request.

We can also check all request's details:

```
[ ]: batch_request.info
```

Any information about a batch request can be obtained from its info dictionary. There are a few properties available to help with extraction:

```
[ ]: print(batch_request.evalscript == evalscript)
print(batch_request.geometry == full_geometry)
```

At this point it is recommended that you write down your batch request ID. In case you restart your Python kernel or delete `batch_request` object you can always re-initialize it with the request ID:

```
[ ]: # Write your request ID here
REQUEST_ID = ''

batch_request_1 = SentinelHubBatch(REQUEST_ID)

batch_request_1
```

3.5.2 2. Analyse a batch request

Before we run a batch request job we can check currently defined batch requests and run an analysis to determine the outcome of a batch request. Important information we can obtain from this step are:

- the exact geometries of tiles from a tiling grid that will be processed,
- the number of processing units that a batch job will cost.

Note that this analysis paragraph is optional and is not required to run a batch request job.

2.1 Investigate past batch requests

We already have our current batch request definition in `batch_request` variable. However, if we would like to find it again we can search the history of all created batch requests:

```
[ ]: for request in SentinelHubBatch.iter_requests(config=config):
    print(request)
```

Alternatively, we can use a method that provides the latest created batch request:

```
[ ]: batch_request = SentinelHubBatch.get_latest_request(config=config)

batch_request
```

2.2 Run an analysis

At the moment we don't have information about tiles or processing units yet. But we can order the service to calculate it.

The following will start the analysis on the server-side:

```
[ ]: batch_request.start_analysis()
```

Depending on the size of our batch request it might take from a few seconds to a few minutes for analysis to finish. To determine if the analysis has finished we have to update batch request info and check the `status` information:

```
[ ]: batch_request.update_info()

batch_request
```

Once analysis is completed the `valueEstimate` tells us an estimated number of processing units the batch job will cost.

```
[ ]: processing_units = batch_request.info['valueEstimate']

print(f'Running this batch job will take about {processing_units:.4f} processing units
↪')
```

2.3 Check tile definitions

When the analysis is complete we can check information about tiles:

[ ]:
```
for tile_info in batch_request.iter_tiles():
    print(tile_info)
```

Optionally, we can request information about a single tile:

[ ]:
```
# Specify a tile ID
TILE_ID = ''

batch_request.get_tile(TILE_ID)
```

To interact with tiles we can also use a type of an `AreaSplitter` class which already parses geometries:

[ ]:
```
from sentinelhub import BatchSplitter

splitter = BatchSplitter(batch_request=batch_request)
splitter.get_bbox_list()
```

Let's plot the geometries:

[ ]:
```
def plot_batch_splitter(splitter):
    """ Plots tiles and area geometry from a splitter class
    """
    gdf = gpd.GeoDataFrame({
        'status': [info['status'] for info in splitter.get_info_list()],
        'geometry': splitter.get_bbox_list()
    }, crs=splitter.crs.pyproj_crs())
    ax = gdf.plot(column='status', legend=True, figsize=(10, 10))

    area_series = gpd.GeoSeries(
        [splitter.get_area_shape()],
        crs=splitter.crs.pyproj_crs()
    )
    area_series.plot(ax=ax, facecolor='none', edgecolor='black')


plot_batch_splitter(splitter)
```

3.5.3 3. Run a batch request job

Once we decide to run a batch request job we can trigger it with the following:

[ ]:
```
batch_request.start_job()
```

Again we can check if a job has finished by updating batch request info.

[ ]:
```
batch_request.update_info()

batch_request
```

Another option is to check which results have already been saved to the given S3 bucket.

When the job is running we can decide at any time to cancel it. Results that have already been produced will remain on the bucket.

```
[ ]: batch_request.cancel_job()
```

When a job has finished we can check the status in batch request info and statuses of each tile:

```
[ ]: splitter = BatchSplitter(batch_request=batch_request)

plot_batch_splitter(splitter)
```

In case processing for any tile fails we have an option to re-run the job again. This will only run the processing for the tiles that failed.

```
[ ]: batch_request.restart_job()
```

Alternatively, we can re-run processing only for a single tile:

```
[ ]: # Specify an ID of a tile that failed
FAILED_TILE_ID = ''

batch_request.reprocess_tile(FAILED_TILE_ID)
```

3.6 Sentinel Hub Feature Info Service (FIS)

One of the tools in `sentinelhub-py` package is a wrapper around Sentinel Hub **Feature (or Statistical) Info Service (FIS)**. The service is documented at Sentinel Hub webpage.

The main purpose of FIS service is to enable users obtaining statistics about satellite data without actually having to download large amounts of raster imagery. It is most effective when you have a sparse collection of bounding boxes or polygons spread over a large area and you would only like to know the aggregated values for each of them.

In order to use FIS it is required to have Sentinel Hub account and configured instance ID. For more about configuration details please check prerequisites of notebook about WMS and WCS.

```
[1]: %matplotlib inline

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import cm
from shapely.geometry import Polygon

from sentinelhub import FisRequest, BBox, Geometry, CRS, WcsRequest, CustomUrlParam, \
    DataCollection, HistogramType
from sentinelhub.time_utils import iso_to_datetime
```

Note that `pandas`, `seaborn` and `matplotlib` are not dependencies of `sentinelhub-py`, therefore you will have to install them manually.

```
[2]: from sentinelhub import SHConfig


INSTANCE_ID = ''  # In case you put instance ID into configuration file you can leave␣
↪this unchanged

if INSTANCE_ID:
```

(continues on next page)

(continued from previous page)

```
    config = SHConfig()
    config.instance_id = INSTANCE_ID
else:
    config = None
```

3.6.1 Exploring basic statistics

In the first example we will explore Sahara desert. There was a news about a sandstorm in March 2018 reported at Earth Observatory webpage. By a quick search in EO Browser we can see the same sandstorm was also observed by Sentinel-2: link.

Let's try to find out how to detect such sandstorm only by looking at some basic statistics of Sentinel-2 data. With FIS we can do that without even downloading any image!

First we set our area of interest - the same part of Sahara desert in Algeria, where the sandstorm should be. For time interval we will take a period of 3 months around that time.

[3]:
```
sahara_bbox = BBox((1.0, 26.0, 1.3, 25.7), CRS.WGS84)
time_interval = ('2018-02-01', '2018-05-01')
```

In order to obtain data from FIS service we first have to initialize `FisRequest` class. Just like `WmsRequest` or `WcsRequest` it belongs in the same family of `DataRequest` classes and therefore works in the same way.

We will set the following parameters:

- `layer` - name of the layer defined in Sentinel Hub Configurator. In this case we will use the layer with all Sentinel-2 L1C bands,
- `geometry_list` - list of geometry objects (`BBox` or `Geometry`), statistics will be calculated for each of them separately,
- `time` - statistics will be calculated for each acquisition in the give time interval separately,
- `resolution` - spatial resolution on which to calculate statistics
- `data_folder` - optional parameter for specifying location where the data should be saved locally

[4]:
```
fis_request = FisRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='BANDS-S2-L1C',
    geometry_list=[sahara_bbox],
    time=time_interval,
    resolution='60m',
    data_folder='./data',
    config=config
)
```

The request can be executed by calling the standard `get_data` method. This will concurrently make calls to FIS for each geometry in the list and collect responses.

[5]:
```
fis_data = fis_request.get_data(save_data=True)  # Takes about 30s, to avoid␣
↪redownloading we are saving results

fis_data[0]['C0'][0]
```

[5]:
```
{'date': '2018-04-30',
 'basicStats': {'min': 0.13920000195503235,
```

(continues on next page)

(continued from previous page)

```
  'max': 0.27570000290870667,
  'mean': 0.16790866606430574,
  'stDev': 0.00859753310640891}}
```

The obtained data is a list of FIS responses for each geometry. Each response is a dictionary with statistics for each channel separately. In our case that is 13 channels, 1 for each Sentinel-2 band. Statistics is then further divided per each acquisition.

Let's parse this data into a `pandas` data frame.

[6]:
```
def fis_data_to_dataframe(fis_data):
    """ Creates a DataFrame from list of FIS responses
    """
    COLUMNS = ['channel', 'date', 'min', 'max', 'mean', 'stDev']
    data = []

    for fis_response in fis_data:
        for channel, channel_stats in fis_response.items():
            for stat in channel_stats:
                row = [int(channel[1:]), iso_to_datetime(stat['date'])]

                for column in COLUMNS[2:]:
                    row.append(stat['basicStats'][column])

                data.append(row)

    return pd.DataFrame(data, columns=COLUMNS).sort_values(['channel', 'date'])


df = fis_data_to_dataframe(fis_data)

df
```

[6]:

	channel	date	min	max	mean	stDev
17	0	2018-02-04	0.1715	0.2577	0.185660	0.005714
16	0	2018-02-09	0.1478	0.2760	0.169501	0.007592
15	0	2018-02-14	0.1165	0.3047	0.170285	0.013389
14	0	2018-02-19	0.1327	0.2658	0.170829	0.015263
13	0	2018-02-24	0.4789	1.0189	0.730869	0.105041
..	...	...	...	...	...	...
220	12	2018-04-10	0.1020	0.5807	0.260120	0.095357
219	12	2018-04-15	0.3007	0.6025	0.460584	0.033551
218	12	2018-04-20	0.2926	0.5892	0.462507	0.035175
217	12	2018-04-25	0.1031	0.5932	0.447324	0.047799
216	12	2018-04-30	0.2652	0.6058	0.468056	0.036688

[234 rows x 6 columns]

The following code will plot timeseries of mean values (with standard deviation) for each band.

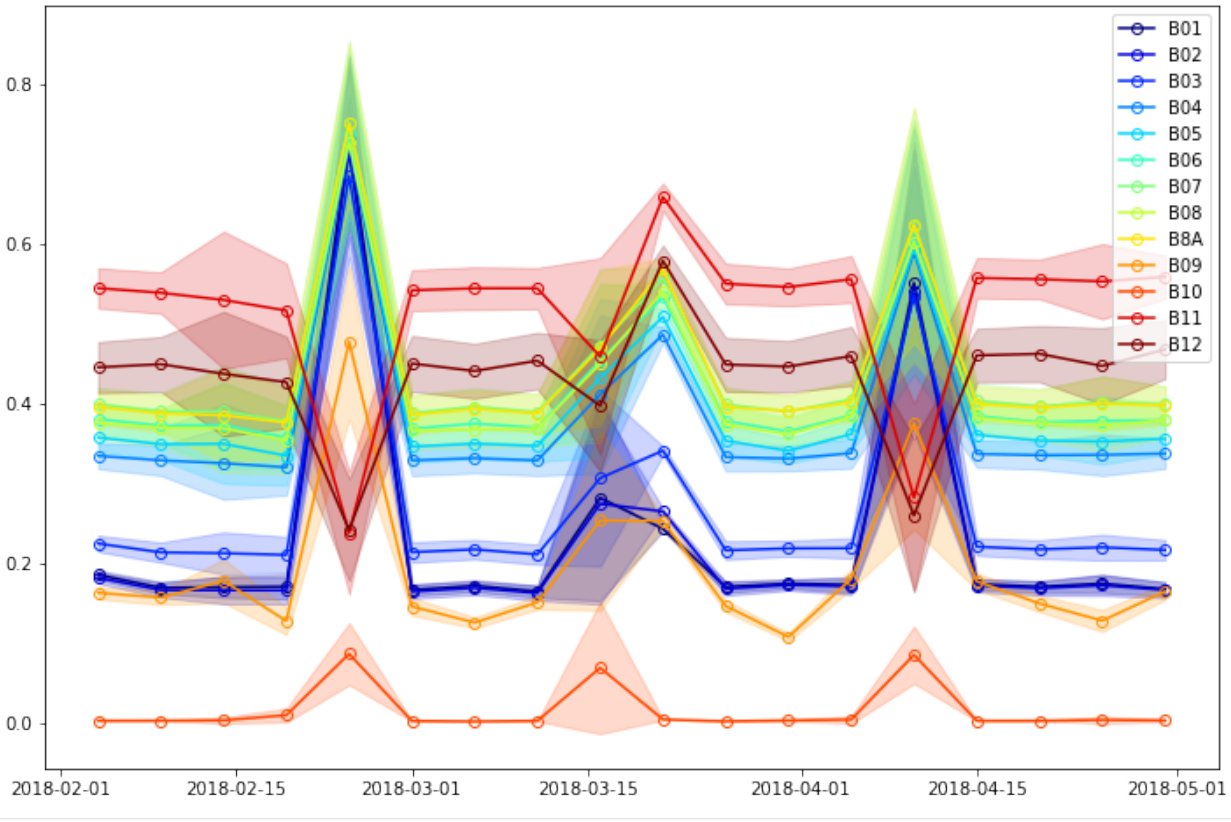
[7]:
```
BANDS = 'B01,B02,B03,B04,B05,B06,B07,B08,B8A,B09,B10,B11,B12'.split(',')

plt.figure(figsize=(12, 8))
for channel, (band, color) in enumerate(zip(BANDS, cm.jet(np.linspace(0, 1, 13)))):
    channel_df = df[df.channel == channel]
    plt.plot(channel_df.date, channel_df['mean'], '-o', markeredgewidth=1,
             color=color, markerfacecolor='None', label=band)
```

(continues on next page)

(continued from previous page)

```
    plt.fill_between(list(channel_df.date),  channel_df['mean'] - channel_df['stDev'],
                     channel_df['mean'] + channel_df['stDev'], alpha=0.2, color=color)

plt.legend(loc='upper right');
```

```
/home/matej/.local/share/virtualenvs/sentinelhub-py-k7ZMsJHq/lib/python3.7/site-
↪packages/pandas/plotting/_matplotlib/converter.py:103: FutureWarning: Using an␣
↪implicitly registered datetime converter for a matplotlib plotting method. The␣
↪converter was registered by pandas on import. Future versions of pandas will␣
↪require you to explicitly register matplotlib converters.

To register the converters:
        >>> from pandas.plotting import register_matplotlib_converters
        >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```



As there is no vegetation in the desert the reflectance values shouldn't change much over time. In our case it appears there are 4 acquisitions with different values that the rest - 5th, 9th, 10th and 14th acquisitions.

For all 4 acquisitions reflectance values are higher in visual spectra. However for 5th, 9th and 14th acquisition SWIR values (bands B11 and B12) are lover than average while for the 10th acquisition these values are higher than average. Hence this can lead us to conclusion that **5th, 9th and 14th acquisition contain clouds** while **10th acquisition is the one with the sandstorm**.

Just to be sure let's download true color images for these acquisitions and visually verify the results.

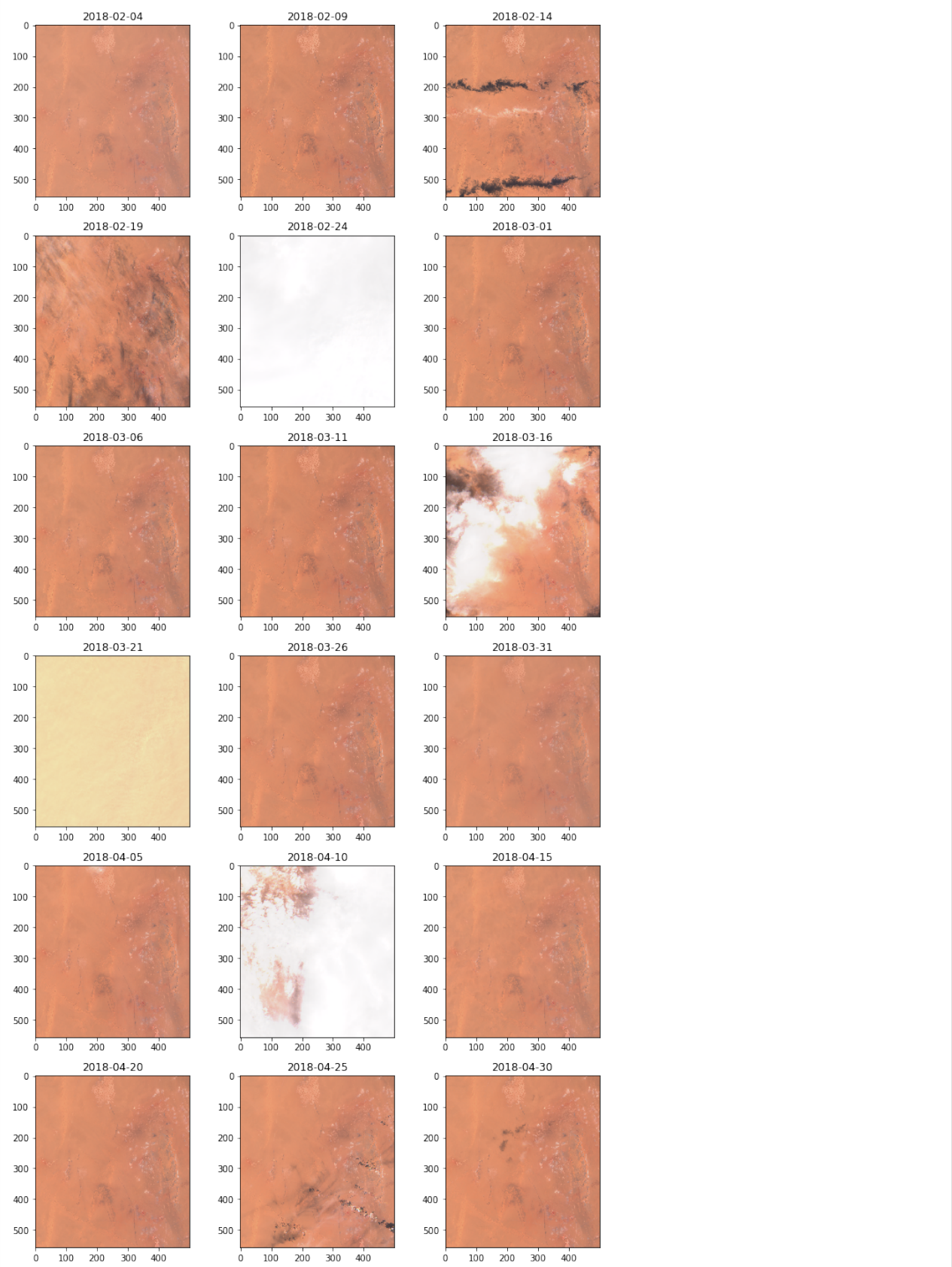
[8]:
```
wcs_request = WcsRequest(
    data_collection=DataCollection.SENTINEL2_L1C,
    layer='TRUE-COLOR-S2-L1C',
```

(continues on next page)

(continued from previous page)

```
    bbox= sahara_bbox,
    time=time_interval,
    resx='60m',
    resy='60m',
    custom_url_params={CustomUrlParam.SHOWLOGO: False},
    config=config
)

images = wcs_request.get_data()
```

[9]:
```
fig, axs = plt.subplots((len(images) + 2) // 3, 3, figsize=(10, 20))

for idx, (image, time) in enumerate(zip(images, wcs_request.get_dates())):
    axs.flat[idx].imshow(image)
    axs.flat[idx].set_title(time.date().isoformat())

fig.tight_layout()
```


2018-02-04
2018-02-09
2018-02-14
2018-02-19
2018-02-24
2018-03-01
2018-03-06
2018-03-11
2018-03-16
2018-03-21
2018-03-26
2018-03-31
2018-04-05
2018-04-10
2018-04-15
2018-04-20
2018-04-25
2018-04-30

3.6.2 Comparing histograms

Besides basic statistics FIS can also provide histograms of values split into specified number of bins. This allows us to analyze distribution of values without having to download entire images.

Let's compare NDVI value distributions of the following bounding boxes and polygons:

[10]:
```
bbox1 = BBox([46.16, -16.15, 46.51, -15.58], CRS.WGS84)
bbox2 = BBox((1292344.0, 5195920.0, 1310615.0, 5214191.0), CRS.POP_WEB)

geometry1 = Geometry(Polygon([(-5.13, 48),
                              (-5.23, 48.09),
                              (-5.13, 48.17),
                              (-5.03, 48.08),
                              (-5.13, 48)]),
                     CRS.WGS84)
geometry2 = Geometry(Polygon([(1292344.0, 5205055.5),
                              (1301479.5, 5195920.0),
                              (1310615.0, 5205055.5),
                              (1301479.5, 5214191.0),
                              (1292344.0, 5205055.5)]),
                     CRS.POP_WEB)
```

Histogram can only be obtained by specifying parameter `bins`, which is the number of bins into which values should be divided. There are multiple ways of dividing values into bins currently supported:

[11]:
```
list(HistogramType)
```

[11]:
```
[<HistogramType.EQUALFREQUENCY: 'equalfrequency'>,
 <HistogramType.EQUIDISTANT: 'equidistant'>,
 <HistogramType.STREAMING: 'streaming'>]
```

In our case we will divide values into `20` bins and the size of each bin will be the same. We will use Landsat 8 data. For simplification we select such time interval that there is only `1` acquisition available.

[12]:
```
ndvi_script = 'return [(B05 - B04) / (B05 + B04)]'

histogram_request = FisRequest(
    data_collection=DataCollection.LANDSAT8,
    layer='TRUE-COLOR-L8',
    geometry_list=[bbox1, bbox2, geometry1, geometry2],
    time=('2018-06-10', '2018-06-15'),
    resolution='100m',
    bins=20,
    histogram_type=HistogramType.EQUIDISTANT,
    custom_url_params={CustomUrlParam.EVALSCRIPT: ndvi_script},
    config=config
)

histogram_data = histogram_request.get_data()
```

[13]:
```
histogram_data[0]
```

[13]:
```
{'C0': [{'date': '2018-06-14',
   'basicStats': {'min': -0.7464028596878052,
    'max': 0.8366492390632629,
    'mean': 0.1811541665712424,
    'stDev': 0.42411094244425696},
```

(continues on next page)

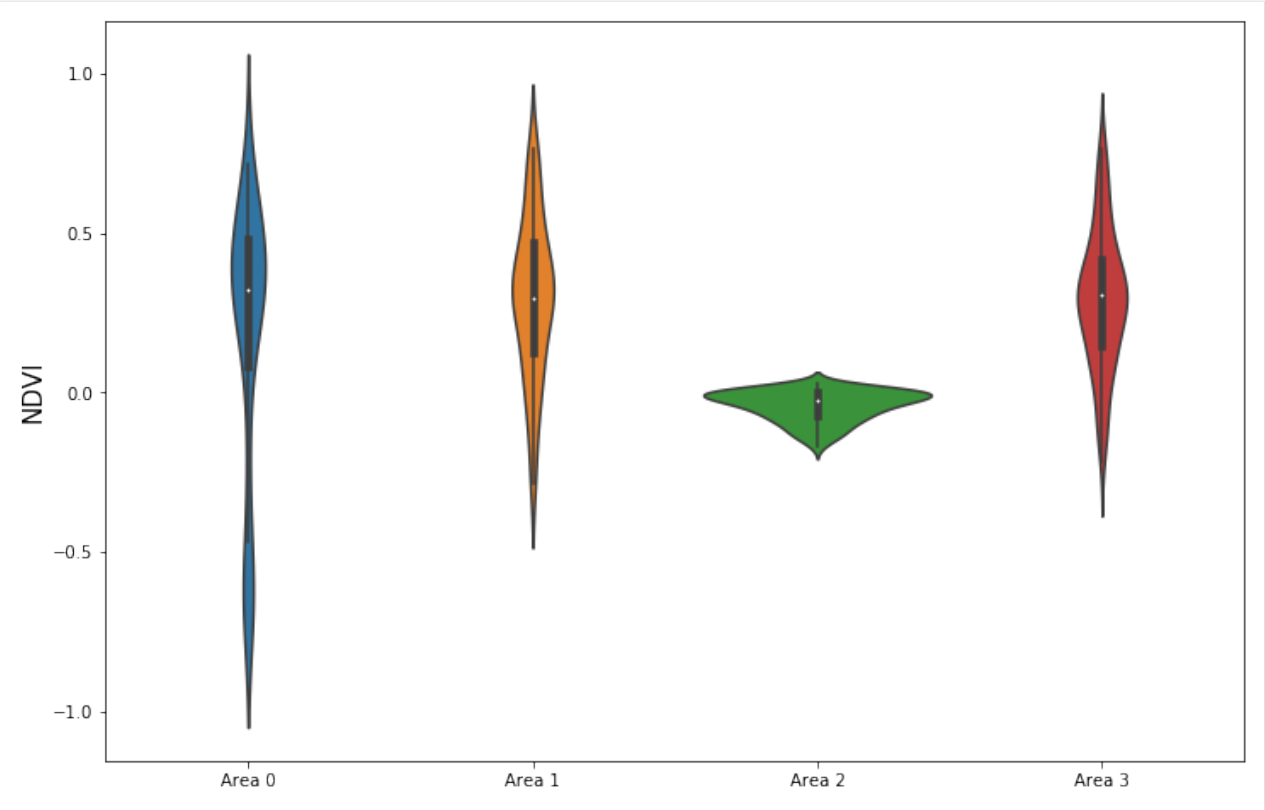
(continued from previous page)


```
   'histogram': {'bins': [{'lowEdge': -0.7464028596878052, 'count': 15504.0},
     {'lowEdge': -0.6672502547502518, 'count': 17223.0},
     {'lowEdge': -0.5880976498126984, 'count': 6786.0},
     {'lowEdge': -0.5089450448751449, 'count': 3744.0},
     {'lowEdge': -0.42979243993759153, 'count': 1972.0},
     {'lowEdge': -0.35063983500003815, 'count': 2149.0},
     {'lowEdge': -0.2714872300624847, 'count': 2305.0},
     {'lowEdge': -0.19233462512493127, 'count': 2342.0},
     {'lowEdge': -0.11318202018737789, 'count': 2145.0},
     {'lowEdge': -0.0340294152498245, 'count': 2188.0},
     {'lowEdge': 0.04512318968772888, 'count': 3874.0},
     {'lowEdge': 0.12427579462528238, 'count': 9674.0},
     {'lowEdge': 0.20342839956283576, 'count': 17459.0},
     {'lowEdge': 0.28258100450038914, 'count': 44026.0},
     {'lowEdge': 0.36173360943794264, 'count': 42916.0},
     {'lowEdge': 0.4408862143754959, 'count': 28010.0},
     {'lowEdge': 0.5200388193130494, 'count': 16899.0},
     {'lowEdge': 0.5991914242506029, 'count': 11092.0},
     {'lowEdge': 0.6783440291881562, 'count': 5956.0},
     {'lowEdge': 0.7574966341257097, 'count': 852.0}]}}]}
```


For each of the 4 areas we got back a FIS response. Each response contains statistics for NDVI channel and a single acquisition date. Besides basic statistics it also contains a histogram.

Let's plot them:

[14]:
```
plot_data = []
for idx, fis_response in enumerate(histogram_data):
    bins = fis_response['C0'][0]['histogram']['bins']

    counts = [value['count'] for value in bins]
    total_counts = sum(counts)
    counts = [round(100 * count / total_counts) for count in counts]

    bin_size = bins[1]['lowEdge'] - bins[0]['lowEdge']
    splits = [value['lowEdge'] + bin_size / 2 for value in bins]

    data = []
    for count, split in zip(counts, splits):
        data.extend([split] * count)
    plot_data.append(np.array(data))


fig, ax = plt.subplots(figsize=(12, 8))
ax = sns.violinplot(data=plot_data, ax=ax)
ax.set(xticklabels=['Area {}'.format(idx) for idx in range(len(histogram_data))])
plt.ylabel('NDVI', fontsize=15);
```



3.7 Accessing satellite data from AWS with Python

This example notebook shows how to obtain Sentinel-2 imagery and additional data from AWS S3 storage buckets. The data at AWS is the same as original S-2 data provided by ESA.

The `sentinelhub` package supports obtaining data by specifying products or by specifying tiles. It can download data either to the same file structure as it is at AWS or it can download data into original `.SAFE` file structure introduced by ESA.

Before testing any of the examples below please check Configuration paragraph for details about configuring AWS credentials and information about charges.

[1]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Note: `matplotlib` is not a dependency of `sentinelhub` and is used in these examples for visualizations.

3.7.1 Searching for available data

The archive of Sentinel-2 data at AWS consists of two buckets, one containing L1C and the other containing L2A data. There are multiple ways to search the archive for specific tiles and products:

- Manual search using aws_cli, e.g.:

```
aws s3 ls s3://sentinel-s2-l2a/tiles/33/U/WR/ --request-payer
```

- Manual search using service available at https://roda.sentinel-hub.com//, which does not require authentication, e.g.:

  https://roda.sentinel-hub.com/sentinel-s2-l1c/tiles/1/C/CV/2017/1/14/0/

- Automatic search by location and time interval using Sentinel Hub Web Feature Service (WFS):

(Note: For this functionality Sentinel Hub instance ID has to be configured according to Configuration paragraph.)

[3]:
```
from sentinelhub import WebFeatureService, BBox, CRS, DataCollection, SHConfig

INSTANCE_ID = ''  # In case you put instance ID into configuration file you can leave␣
↪this unchanged

if INSTANCE_ID:
    config = SHConfig()
    config.instance_id = INSTANCE_ID
else:
    config = None


search_bbox = BBox(bbox=[46.16, -16.15, 46.51, -15.58], crs=CRS.WGS84)
search_time_interval = ('2017-12-01T00:00:00', '2017-12-15T23:59:59')


wfs_iterator = WebFeatureService(
    search_bbox,
    search_time_interval,
    data_collection=DataCollection.SENTINEL2_L1C,
    maxcc=1.0,
    config=config
)

for tile_info in wfs_iterator:
    print(tile_info)
```

```
{'type': 'Feature', 'geometry': {'type': 'MultiPolygon', 'crs': {'type': 'name',
↪'properties': {'name': 'urn:ogc:def:crs:EPSG::4326'}}, 'coordinates': [[[[45.
↪93178396701427, -15.374656928849852], [46.95453856838988, -15.368029754563597], [46.
↪96412360581364, -16.360077552492225], [45.93635618696065, -16.3671551019236], [45.
↪93178396701427, -15.374656928849852]]]]}, 'properties': {'id': 'S2B_OPER_MSI_L1C_TL_
↪MTI__20171215T085654_A004050_T38LPH_N02.06', 'date': '2017-12-15', 'time': '07:12:03
↪', 'path': 's3://sentinel-s2-l1c/tiles/38/L/PH/2017/12/15/0', 'crs': 'EPSG:32738',
↪'mbr': '600000,8190220 709800,8300020', 'cloudCoverPercentage': 28.27}}
{'type': 'Feature', 'geometry': {'type': 'MultiPolygon', 'crs': {'type': 'name',
↪'properties': {'name': 'urn:ogc:def:crs:EPSG::4326'}}, 'coordinates': [[[[45.
↪93178396701427, -15.374656928849852], [46.95453856838988, -15.368029754563597], [46.
↪96412360581364, -16.360077552492225], [45.93635618696065, -16.3671551019236], [45.
↪93178396701427, -15.374656928849852]]]]}, 'properties': {'id': 'S2A_OPER_MSI_L1C_TL_
↪SGS__20171210T103113_A012887_T38LPH_N02.06', 'date': '2017-12-10', 'time': '07:12:10
↪', 'path': 's3://sentinel-s2-l1c/tiles/38/L/PH/2017/12/10/0', 'crs': 'EPSG:32738',
↪'mbr': '600000,8190220 709800,8300020', 'cloudCoverPercentage': 94.02}}
{'type': 'Feature', 'geometry': {'type': 'MultiPolygon', 'crs': {'type': 'name',
↪'properties': {'name': 'urn:ogc:def:crs:EPSG::4326'}}, 'coordinates': [[[[45.
↪93178396701427, -15.374656928849852], [46.95453856838988, -15.368029754563597], [46.
↪96412360581364, -16.360077552492225], [45.93635618696065, -16.3671551019236], [45.
↪93178396701427, -15.374656928849852]]]]}, 'properties': {'id': 'S2B_OPER_MSI_L1C_TL_
↪SGS__20171205T102636_A003907_T38LPH_N02.06', 'date': '2017-12-05', 'time': '07:13:30
↪', 'path': 's3://sentinel-s2-l1c/tiles/38/L/PH/2017/12/5/0', 'crs': 'EPSG:32738',
↪'mbr': '600000,8190220 709800,8300020', 'cloudCoverPercentage': 91.74}}
```

(continues on next page)

(continued from previous page)

From obtained WFS iterator we can extract info which uniquely defines each tile.

[4]:
```
wfs_iterator.get_tiles()
```

[4]:
```
[('38LPH', '2017-12-15', 0),
 ('38LPH', '2017-12-10', 0),
 ('38LPH', '2017-12-5', 0)]
```

- Automatic search with functions from sentinelhub.opensearch module (no authentication required):

[5]:
```
from sentinelhub import get_area_info

for tile_info in get_area_info(search_bbox, search_time_interval, maxcc=0.5):
    print(tile_info)
```

```
{'type': 'Feature', 'id': '985b7c0c-5d4a-5105-a37b-ef41f4092392', 'geometry': {'type':
↪ 'MultiPolygon', 'coordinates': [[[[45.931783967, -15.374656929], [46.954538568, -
↪15.368029755], [46.964123606, -16.360077552], [45.936356187, -16.367155102], [45.
↪931783967, -15.374656929]]]]}, 'properties': {'collection': 'Sentinel2', 'license':
↪{'licenseId': 'unlicensed', 'hasToBeSigned': 'never', 'grantedCountries': None,
↪'grantedOrganizationCountries': None, 'grantedFlags': None, 'viewService': 'public',
↪ 'signatureQuota': -1, 'description': {'shortName': 'No license'}},
↪'productIdentifier': 'S2B_OPER_MSI_L1C_TL_MTI__20171215T085654_A004050_T38LPH_N02.06
↪', 'parentIdentifier': None, 'title': 'S2B_OPER_MSI_L1C_TL_MTI__20171215T085654_
↪A004050_T38LPH_N02.06', 'description': None, 'organisationName': None, 'startDate':
↪'2017-12-15T07:12:03Z', 'completionDate': '2017-12-15T07:12:03Z', 'productType':
↪'S2MSI1C', 'processingLevel': '1C', 'platform': 'Sentinel-2', 'instrument': 'MSI',
↪'resolution': 10, 'sensorMode': None, 'orbitNumber': 4050, 'quicklook': None,
↪'thumbnail': None, 'updated': '2017-12-15T14:03:04.356331Z', 'published': '2017-12-
↪15T14:03:04.356331Z', 'snowCover': 0, 'cloudCover': 28.27, 'keywords': [], 'centroid
↪': {'type': 'Point', 'coordinates': [23.482061803, -15.8675924285]}, 's3Path':
↪'tiles/38/L/PH/2017/12/15/0', 'spacecraft': 'S2B', 'sgsId': 3440459, 's3URI': 's3://
↪sentinel-s2-l1c/tiles/38/L/PH/2017/12/15/0/', 'services': {'download': {'url':
↪'http://sentinel-s2-l1c.s3-website.eu-central-1.amazonaws.com#tiles/38/L/PH/2017/12/
↪15/0/', 'mimeType': 'text/html'}}, 'links': [{'rel': 'self', 'type': 'application/
↪json', 'title': 'GeoJSON link for 985b7c0c-5d4a-5105-a37b-ef41f4092392', 'href':
↪'http://opensearch.sentinel-hub.com/resto/collections/Sentinel2/985b7c0c-5d4a-5105-
↪a37b-ef41f4092392.json?&lang=en'}]}}
```

3.7.2 Download data

Once we have found correct tiles or products we can download them and explore the data.

Aws Tile

Sentinel-2 tile can be uniquely defined either with ESA tile ID (e.g. `L1C_T01WCV_A012011_20171010T003615`) or with tile name (e.g. `T38TML` or `38TML`), sensing time and AWS index. The AWS index is the last number in tile AWS path (e.g. https://roda.sentinel-hub.com/sentinel-s2-l1c/tiles/1/C/CV/2017/1/14/0/ → `0`).

The package works with the second tile definition. To transform tile ID to `(tile_name, time, aws_index)` do the following:

[6]:
```
from sentinelhub import AwsTile

tile_id = 'S2A_OPER_MSI_L1C_TL_MTI__20151219T100121_A002563_T38TML_N02.01'
tile_name, time, aws_index = AwsTile.tile_id_to_tile(tile_id)
tile_name, time, aws_index
```

[6]:
```
('38TML', '2015-12-19', 1)
```

Now we are ready to download the data. Let's download only bands `B8A` and `B10`, meta data files `tileInfo.json`, `preview.jp2` and pre-calculated cloud mask `qi/MSK_CLOUDS_B00`. We will save everything into folder `./AwsData`.

[7]:
```
from sentinelhub import AwsTileRequest

bands = ['B8A', 'B10']
metafiles = ['tileInfo', 'preview', 'qi/MSK_CLOUDS_B00']
data_folder = './AwsData'

request = AwsTileRequest(
    tile=tile_name,
    time=time,
    aws_index=aws_index,
    bands=bands,
    metafiles=metafiles,
    data_folder=data_folder,
    data_collection=DataCollection.SENTINEL2_L1C
)

request.save_data()  # This is where the download is triggered
```

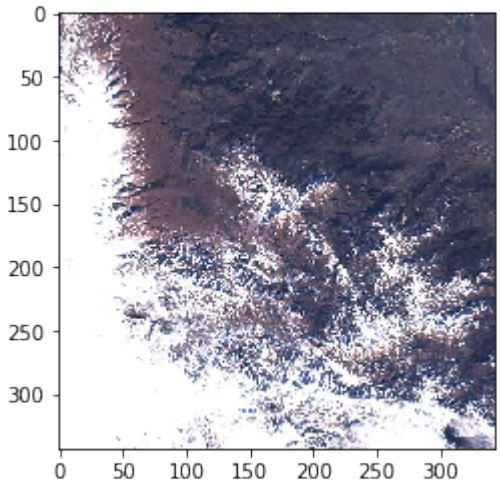
Note that upon calling this method again the data won't be re-downloaded unless we set the parameter `redownload=True`.

To obtain downloaded data we can simply do:

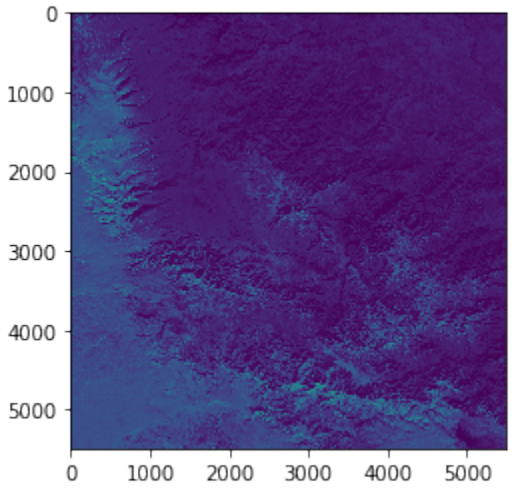
[8]:
```
data_list = request.get_data()  # This will not redownload anything because data is␣
↪already stored on disk

b8a, b10, tile_info, preview, cloud_mask = data_list
```

Download and reading could also be done in a single call `request.get_data(save_data=True)`.

[9]:
```
_ = plt.imshow(preview)
```



[10]:
```
_ = plt.imshow(b8a)
```



Aws Product

Sentinel-2 product is uniquely defined by ESA product ID. We can obtain data for the whole product

[11]:
```
from sentinelhub import AwsProductRequest

product_id = 'S2A_MSIL1C_20171010T003621_N0205_R002_T01WCV_20171010T003615'

request = AwsProductRequest(product_id=product_id, data_folder=data_folder)

# Uncomment the the following line to download the data:
# data_list = request.get_data(save_data=True)
```

If `bands` parameter is not defined all bands will be downloaded. If `metafiles` parameter is not defined no additional metadata files will be downloaded.

Data into .SAFE structure

The data can also be downloaded into .SAFE structure by specifying `safe_format=True`. The following code will download data from upper example again because now data will be stored in different folder structure.

[12]:
```
tile_request = AwsTileRequest(
    tile=tile_name,
    time=time,
    aws_index=aws_index,
    data_collection=DataCollection.SENTINEL2_L1C,
    bands=bands,
    metafiles=metafiles,
    data_folder=data_folder,
    safe_format=True
)

# Uncomment the the following line to download the data:
# tile_request.save_data()
```

[13]:
```
product_id = 'S2A_OPER_PRD_MSIL1C_PDMC_20160121T043931_R069_V20160103T171947_
↪20160103T171947'

product_request = AwsProductRequest(
    product_id=product_id,
    bands=['B01'],
    data_folder=data_folder,
    safe_format=True
)

# Uncomment the the following line to download the data:
# product_request.save_data()
```

Older products contain multiple tiles. In case would like to download only some tiles it is also possible to specify a list of tiles to download.

[14]:
```
product_request = AwsProductRequest(
    product_id=product_id,
    tile_list=['T14PNA', 'T13PHT'],
    data_folder=data_folder,
    safe_format=True
)

# Uncomment the the following line to download the data:
# product_request.save_data()
```

3.8 Downloading satellite data from AWS with command line

The following examples show how to download Sentinel-2 data from AWS S3 storage bucket and store them into original ESA .SAFE format. Before testing any of the examples below please check Configuration paragraph for details about configuring AWS credentials and information about charges.

3.8.1 Sentinel-2 products

To download a Sentinel-2 product and save it to present working directory only ESA product ID has to be specified:

```
$ sentinelhub.aws --product S2A_MSIL1C_20170414T003551_N0204_R016_T54HVH_
↪20170414T003551
```

Download of L2A products works in the same way, as the difference between L1C and L2A can be determined from product ID:

```
$ sentinelhub.aws --product S2A_MSIL2A_20180402T151801_N0207_R068_T33XWJ_
↪20180402T202222
```

If certain file has already been downloaded and exists in the expected folder it by default won't be redownloaded. However redownload can be specified with the following flag:

```
$ sentinelhub.aws --product S2A_MSIL1C_20170414T003551_N0204_R016_T54HVH_
↪20170414T003551 -r
```

It is possible to get only information about .SAFE structure (without charge) and not download the product:

```
$ sentinelhub.aws --product S2A_MSIL1C_20170414T003551_N0204_R016_T54HVH_
↪20170414T003551 -i
```

Download product and save it to specific file directory:

```
$ sentinelhub.aws --product S2A_MSIL1C_20170414T003551_N0204_R016_T54HVH_
↪20170414T003551 -f /home/ESA_Products
```

Download specified bands only:

```
$ sentinelhub.aws --product S2A_MSIL1C_20170414T003551_N0204_R016_T54HVH_
↪20170414T003551 --bands B08,B11
```

3.8.2 Sentinel-2 tiles

It is possible to download only a specific tile within the product. The tile can be specified with tile name (i.e. tile's spatial location) and sensing date:

```
$ sentinelhub.aws --tile T54HVH 2017-04-14
```

By default L1C tile data will be downloaded. In order to download L2A tile data a flag has to be used:

```
$ sentinelhub.aws --tile T33XWJ 2018-04-02 --l2a
```

Download entire product corresponding to tile:

```
$ sentinelhub.aws --tile T54HVH 2017-04-14 -e
```

3.8.3 .SAFE format details

Because files in AWS bucket are stored differently the *sentinelhub* package has to reconstruct .SAFE format following the rules of ESA naming convention. Reconstructed format may differ from the original only in the following:

- Folders HTML and rep_info inside main product folder do not contain any data.
- Auxiliary file inside AUX_DATA folder of every tile in old .SAFE format does not have original name. Instead it is named AUX_ECMWFT which is the same as in compact .SAFE format.

- Some products created in October 2017 might miss quality report files (*FORMAT_CORRECTNESS.xml*, *GENERAL_QUALITY.xml*, *GEOMETRIC_QUALITY.xml*, *RADIOMETRIC_QUALITY.xml* and *SENSOR_QUALITY.xml*).

If you notice any other difference please raise an issue at GitHub page.

CHAPTER 4

Package content

Content of the **sentinelhub** package.

4.1 Modules

4.1.1 areas

Module for working with large geographical areas

sentinelhub.areas.**AreaSplitter**

class sentinelhub.areas.**BBoxSplitter**(*shape_list*, *crs*, *split_shape*, ***kwargs*)

A tool that splits the given area into smaller parts. Given the area it calculates its bounding box and splits it into smaller bounding boxes of equal size. Then it filters out the bounding boxes that do not intersect the area. If specified by user it can also reduce the sizes of the remaining bounding boxes to best fit the area.

Parameters

- **shape_list** (*list(shapely.geometry.multipolygon.MultiPolygon or shapely.geometry.polygon.Polygon)*) – A list of geometrical shapes describing the area of interest
- **crs** (CRS) – Coordinate reference system of the shapes in *shape_list*
- **split_shape** (*int or (int, int)*) – Parameter that describes the shape in which the area bounding box will be split. It can be a tuple of the form *(n, m)* which means the area bounding box will be split into *n* columns and *m* rows. It can also be a single integer *n* which is the same as *(n, n)*.
- **reduce_bbox_sizes** (*bool*) – If *True* it will reduce the sizes of bounding boxes so that they will tightly fit the given area geometry from *shape_list*.

class sentinelhub.areas.**OsmSplitter**(*shape_list*, *crs*, *zoom_level*, ***kwargs*)

A tool that splits the given area into smaller parts. For the splitting it uses Open Street Map (OSM) grid on the

specified zoom level. It calculates bounding boxes of all OSM tiles that intersect the area. If specified by user it can also reduce the sizes of the remaining bounding boxes to best fit the area.

Parameters

- **shape_list** (*list(shapely.geometry.multipolygon.MultiPolygon or shapely.geometry.polygon.Polygon)*) – A list of geometrical shapes describing the area of interest
- **crs** (CRS) – Coordinate reference system of the shapes in *shape_list*
- **zoom_level** (*int*) – A zoom level defined by OSM. Level 0 is entire world, level 1 splits the world into 4 parts, etc.
- **reduce_bbox_sizes** (*bool*) – If *True* it will reduce the sizes of bounding boxes so that they will tightly fit the given area geometry from *shape_list*.

get_world_bbox()

Creates a bounding box of the entire world in EPSG: 3857

Returns Bounding box of entire world

Return type *BBox*

class sentinelhub.areas.**TileSplitter**(*shape_list*, *crs*, *time_interval*, *tile_split_shape=1*, *data_collection=None*, *config=None*, *data_source=None*, ***kwargs*)

A tool that splits the given area into smaller parts. Given the area, time interval and data collection it collects info from Sentinel Hub WFS service about all satellite tiles intersecting the area. For each of them it calculates bounding box and if specified it splits these bounding boxes into smaller bounding boxes. Then it filters out the ones that do not intersect the area. If specified by user it can also reduce the sizes of the remaining bounding boxes to best fit the area.

Parameters

- **shape_list** (*list(shapely.geometry.multipolygon.MultiPolygon or shapely.geometry.polygon.Polygon)*) – A list of geometrical shapes describing the area of interest
- **crs** (CRS) – Coordinate reference system of the shapes in *shape_list*
- **time_interval** (*(str, str)*) – Interval with start and end date of the form YYYY-MM-DDThh:mm:ss or YYYY-MM-DD
- **tile_split_shape** – Parameter that describes the shape in which the satellite tile bounding boxes will be split. It can be a tuple of the form *(n, m)* which means the tile bounding boxes will be split into *n* columns and *m* rows. It can also be a single integer *n* which is the same as *(n, n)*.
- **data_collection** (DataCollection) – A satellite data collection
- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.
- **reduce_bbox_sizes** (*bool*) – If *True* it will reduce the sizes of bounding boxes so that they will tightly fit the given area geometry from *shape_list*.
- **data_source** (DataCollection) – A deprecated alternative of data_collection

get_tile_dict()

Returns the dictionary of satellite tiles intersecting the area geometry. For each tile they contain info about their bounding box and lists of acquisitions and geometries

Returns Dictionary containing info about tiles intersecting the area

Return type dict

class sentinelhub.areas.**CustomGridSplitter**(*shape_list*, *crs*, *bbox_grid*, *bbox_split_shape=1*, ***kwargs*)

Splitting class which can split according to given custom collection of bounding boxes

Parameters

- **shape_list** (*list(shapely.geometry.multipolygon.MultiPolygon or shapely.geometry.polygon.Polygon)*) – A list of geometrical shapes describing the area of interest
- **crs** (CRS) – Coordinate reference system of the shapes in *shape_list*
- **bbox_grid** (*list(BBox) or BBoxCollection*) – A collection of bounding boxes defining a grid of splitting. All of them have to be in the same CRS.
- **bbox_split_shape** (*int or (int, int)*) – Parameter that describes the shape in which each of the bounding boxes in the given grid will be split. It can be a tuple of the form *(n, m)* which means the tile bounding boxes will be split into *n* columns and *m* rows. It can also be a single integer *n* which is the same as *(n, n)*.
- **reduce_bbox_sizes** (*bool*) – If *True* it will reduce the sizes of bounding boxes so that they will tightly fit the given geometry in *shape_list*.

4.1.2 aws

Module for obtaining data from Amazon Web Service

class sentinelhub.aws.**AwsService**(*parent_folder=''*, *bands=None*, *metafiles=None*, *config=None*)

Bases: abc.ABC

Amazon Web Service (AWS) base class

Parameters

- **parent_folder** (*str*) – Folder where the fetched data will be saved.
- **bands** (*list(str) or None*) – List of Sentinel-2 bands for request. If parameter is set to *None* all bands will be used.
- **metafiles** (*list(str) or None*) – List of additional metafiles available on AWS (e.g. `['metadata', 'tileInfo', 'preview/B01', 'TCI']`). If parameter is set to *None* the list will be set automatically.
- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

get_requests()

Abstract class for joining together download requests

get_base_url(*force_http=False*)

Creates base URL path

Parameters **force_http** (*str*) – *True* if HTTP base URL should be used and *False* otherwise

Returns base url string

Return type str

get_safe_type()
Determines the type of ESA product.

In 2016 ESA changed structure and naming of data. Therefore the class must distinguish between old product type and compact (new) product type.

Returns type of ESA product

Return type *constants.EsaSafeType*

Raises ValueError

get_baseline()
Determines the baseline number (i.e. version) of ESA .SAFE product

Returns baseline number

Return type str

Raises ValueError

static url_to_tile(*url*)
Extracts tile name, date and AWS index from tile url on AWS.

Parameters **url** (*str*) – class input parameter 'metafiles'

Returns Name of tile, date and AWS index which uniquely identifies tile on AWS

Return type (str, str, int)

sort_download_list()
Method for sorting the list of download requests. Band images have priority before metadata files. If bands images or metadata files are specified with a list they will be sorted in the same order as in the list. Otherwise they will be sorted alphabetically (band B8A will be between B08 and B09).

structure_recursion(*struct*, *folder*)
From nested dictionaries representing .SAFE structure it recursively extracts all the files that need to be downloaded and stores them into class attribute *download_list*.

Parameters

- **struct** (*dict*) – nested dictionaries representing a part of .SAFE structure
- **folder** (*str*) – name of folder where this structure will be saved

static add_file_extension(*filename*, *data_format=None*, *remove_path=False*)
Joins filename and corresponding file extension if it has one.

Parameters

- **filename** (*str*) – Name of the file without extension
- **data_format** (*constants.MimeType or None*) – format of file, if *None* it will be set automatically
- **remove_path** (*bool*) – *True* if the path in filename string should be removed

Returns Name of the file with extension

Return type str

has_reports()
Products created with baseline 2.06 and greater (and some products with baseline 2.05) should have quality report files

Returns *True* if the product has report xml files and *False* otherwise

Return type bool

is_early_compact_l2a()
Check if product is early version of compact L2A product

Returns *True* if product is early version of compact L2A product and *False* otherwise

Return type bool

class sentinelhub.aws.**AwsProduct**(*product_id*, *tile_list=None*, ***kwargs*)
Bases: *sentinelhub.aws.AwsService*

Service class for Sentinel-2 product on AWS

Parameters

- **product_id** (*str*) – ESA ID of the product
- **tile_list** (*list(str) or None*) – list of tile names
- **parent_folder** (*str*) – location of the directory where the fetched data will be saved.
- **bands** (*list(str) or None*) – List of Sentinel-2 bands for request. If parameter is set to *None* all bands will be used.
- **metafiles** (*list(str) or None*) – List of additional metafiles available on AWS (e.g. `['metadata', 'tileInfo', 'preview/B01', 'TCI']`). If parameter is set to *None* the list will be set automatically.
- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

static parse_tile_list(*tile_input*)
Parses class input and verifies band names.

Parameters **tile_input** (*str or list(str)*) – class input parameter *tile_list*

Returns parsed list of tiles

Return type list(str) or None

get_requests()
Creates product structure and returns list of files for download.

Returns List of download requests and list of empty folders that need to be created

Return type (list(download.DownloadRequest), list(str))

get_data_collection()
The method determines data collection from product ID.

Returns Data collection of the product

Return type *DataCollection*

Raises ValueError

get_date()
Collects sensing date of the product.

Returns Sensing date

Return type str

get_url(*filename*, *data_format=None*)
Creates url of file location on AWS.

Parameters

- **filename** (*str*) – name of file
- **data_format** (*constants.MimeType or None*) – format of file, if *None* it will be set automatically

Returns url of file location

Return type str

get_product_url(*force_http=False*)
Creates base url of product location on AWS.

Parameters **force_http** (*str*) – *True* if HTTP base URL should be used and *False* otherwise

Returns url of product location

Return type str

get_tile_url(*tile_info*)
Collects tile url from *productInfo.json* file.

Parameters **tile_info** (*dict*) – information about tile from *productInfo.json*

Returns url of tile location

Return type str

get_filepath(*filename*)
Creates file path for the file.

Parameters **filename** (*str*) – name of the file

Returns filename with path on disk

Return type str

class sentinelhub.aws.**AwsTile**(*tile_name*, *time*, *aws_index=None*, *data_collection=<DataCollection.SENTINEL2_L1C: DataCollectionDefinition(api_id: S2L1C wfs_id: DSS1 collection_type: Sentinel-2 sensor_type: MSI processing_level: L1C bands: ('B01', 'B02', 'B03', 'B04', 'B05', 'B06', 'B07', 'B08', 'B8A', 'B09', 'B10', 'B11', 'B12') is_timeless: False)>*, ***kwargs*)
Bases: *sentinelhub.aws.AwsService*

Service class for Sentinel-2 product on AWS

Parameters

- **tile** (*str*) – Tile name (e.g. 'T10UEV')
- **time** (*str*) – Tile sensing time in ISO8601 format
- **aws_index** (*int or None*) – There exist Sentinel-2 tiles with the same tile and time parameter. Therefore each tile on AWS also has an index which is visible in their url path. If `aws_index` is set to *None* the class will try to find the index automatically. If there will be multiple choices it will choose the lowest index and inform the user.
- **data_collection** (*DataCollection*) – A collection of requested AWS data. Supported collections are Sentinel-2 L1C and Sentinel-2 L2A, default is Sentinel-2 L1C data.
- **parent_folder** (*str*) – folder where the fetched data will be saved.
- **bands** (*list(str) or None*) – List of Sentinel-2 bands for request. If parameter is set to *None* all bands will be used.

- **metafiles** (*list(str) or None*) – List of additional metafiles available on AWS (e.g. `['metadata', 'tileInfo', 'preview/B01', 'TCI']`). If parameter is set to *None* the list will be set automatically.
- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

static parse_tile_name(*name*)
Parses and verifies tile name.

Parameters **name** (*str*) – class input parameter *tile_name*

Returns parsed tile name

Return type str

static parse_datetime(*time*)
Parses and verifies tile sensing time.

Parameters **time** (*str*) – tile sensing time

Returns tile sensing time in ISO8601 format

Return type str

get_requests()
Creates tile structure and returns list of files for download.

Returns List of download requests and list of empty folders that need to be created

Return type (list(download.DownloadRequest), list(str))

get_aws_index()
Returns tile index on AWS. If *tile_index* was not set during class initialization it will be determined according to existing tiles on AWS.

Returns Index of tile on AWS

Return type int

tile_is_valid()
Checks if tile has tile info and valid timestamp

Returns *True* if tile is valid and *False* otherwise

Return type bool

get_tile_info()
Collects basic info about tile from tileInfo.json.

Returns dictionary with tile information

Return type dict

get_url(*filename*)
Creates url of file location on AWS.

Parameters **filename** (*str*) – name of file

Returns url of file location

Return type str

get_tile_url(*force_http=False*)
Creates base url of tile location on AWS.

Parameters **force_http** (*str*) – *True* if HTTP base URL should be used and *False* otherwise

Returns url of tile location

Return type str

get_qi_url(*metafile*)
Returns url of tile metadata products

Parameters **metafile** (*str*) – Name of metadata product at AWS

Returns url location of metadata product at AWS

Return type str

get_gml_url(*qi_type*, *band='B00'*)

Parameters

- **qi_type** (*str*) – type of quality indicator
- **band** (*str*) – band name

Returns location of gml file on AWS

Return type str

get_preview_url(*data_type='L1C'*)
Returns url location of full resolution L1C preview :return:

get_filepath(*filename*)
Creates file path for the file.

Parameters **filename** (*str*) – name of the file

Returns filename with path on disk

Return type str

get_product_id()
Obtains ESA ID of product which contains the tile.

Returns ESA ID of the product

Return type str

static tile_id_to_tile(*tile_id*)

Parameters **tile_id** (*str*) – original tile identification string provided by ESA (e.g. 'S2A_OPER_MSI_L1C_TL_SGS__20160109T230542_A002870_T10UEV_N02.01')

Returns tile name, sensing date and AWS index

Return type (str, str, int)

4.1.3 aws_safe

Module for creating .SAFE structure with data from AWS

class sentinelhub.aws_safe.**SafeProduct**(*product_id*, *tile_list=None*, ***kwargs*)
Bases: *sentinelhub.aws.AwsProduct*

Class implementing transformation of satellite products from AWS into .SAFE structure

Parameters

- **product_id** (*str*) – ESA ID of the product
- **tile_list** (*list(str) or None*) – list of tile names
- **parent_folder** (*str*) – location of the directory where the fetched data will be saved.
- **bands** (*list(str) or None*) – List of Sentinel-2 bands for request. If parameter is set to *None* all bands will be used.
- **metafiles** (*list(str) or None*) – List of additional metafiles available on AWS (e.g. `['metadata', 'tileInfo', 'preview/B01', 'TCI']`). If parameter is set to *None* the list will be set automatically.
- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

get_requests()

Creates product structure and returns list of files for download

Returns list of download requests

Return type list(download.DownloadRequest)

get_safe_struct()

Describes a structure inside tile folder of ESA product .SAFE structure

Returns nested dictionaries representing .SAFE structure

Return type dict

get_main_folder()

Returns name of main folder

Return type str

get_datastrip_list()

Returns list of datastrips folder names and urls from *productInfo.json* file

Return type list((str, str))

get_datastrip_name(*datastrip*)

Parameters **datastrip** (*str*) – name of datastrip

Returns name of datastrip folder

Return type str

get_datastrip_metadata_name(*datastrip_folder*)

Parameters **datastrip_folder** (*str*) – name of datastrip folder

Returns name of datastrip metadata file

Return type str

get_product_metadata_name()

Returns name of product metadata file

Return type str

get_report_name()

Returns name of the report file of L2A products

Return type str

get_report_time()
Returns time when the L2A processing started and reports was created. :return: String in a form YYYYMMDDTHHMMSS :rtype: str

class sentinelhub.aws_safe.**SafeTile**(**args*, ***kwargs*)
Bases: *sentinelhub.aws.AwsTile*

Class implementing transformation of satellite tiles from AWS into .SAFE structure

Initialization parameters are inherited from parent class

get_requests()
Creates tile structure and returns list of files for download.

Returns list of download requests for

Return type list(download.DownloadRequest)

get_safe_struct()
Describes a structure inside tile folder of ESA product .SAFE structure.

Returns nested dictionaries representing .SAFE structure

Return type dict

get_tile_id()
Creates ESA tile ID

Returns ESA tile ID

Return type str

get_sensing_time()

Returns Exact tile sensing time

Return type str

get_datastrip_time()

Returns Exact datastrip time

Return type str

get_datatake_time()

Returns Exact time of datatake

Return type str

get_main_folder()

Returns name of tile folder

Return type str

get_tile_metadata_name()

Returns name of tile metadata file

Return type str

get_aux_data_name()

Returns name of auxiliary data file

Return type str

get_img_name(*band*, *resolution=None*)

Parameters

- **band** (*str*) – band name
- **resolution** (*str or None*) – Specifies the resolution in case of Sentinel-2 L2A products

Returns name of band image file

Return type str

get_qi_name(*qi_type*, *band='B00'*, *data_format=<MimeType.GML: 'gml'>*)

Parameters

- **qi_type** (*str*) – type of quality indicator
- **band** (*str*) – band name
- **data_format** (MimeType) – format of the file

Returns name of gml file

Return type str

get_preview_name()

Returns .SAFE name of full resolution L1C preview :return: name of preview file :rtype: str

4.1.4 config

Module for managing configuration data from *config.json*

class sentinelhub.config.**SHConfig**

Bases: object

A sentinelhub-py package configuration class.

The class reads during its first initialization the configurable settings from `./config.json` file:

- *instance_id*: An instance ID for Sentinel Hub service used for OGC requests.
- *sh_client_id*: User's OAuth client ID for Sentinel Hub service
- *sh_client_secret*: User's OAuth client secret for Sentinel Hub service
- ***sh_base_url*: There exist multiple deployed instances of Sentinel Hub service, this parameter defines the** location of a specific service instance.
- *geopedia_wms_url*: Base url for Geopedia WMS services.
- *geopedia_rest_url*: Base url for Geopedia REST services.
- *aws_access_key_id*: Access key for AWS Requester Pays buckets.
- *aws_secret_access_key*: Secret access key for AWS Requester Pays buckets.
- *aws_metadata_url*: Base url for publicly available metadata files
- *aws_s3_l1c_bucket*: Name of Sentinel-2 L1C bucket at AWS s3 service.
- *aws_s3_l2a_bucket*: Name of Sentinel-2 L2A bucket at AWS s3 service.
- *opensearch_url*: Base url for Sentinelhub Opensearch service.
- *max_wfs_records_per_query*: Maximum number of records returned for each WFS query.
- *max_opensearch_records_per_query*: Maximum number of records returned for each Opensearch query.

- *max_download_attempts*: Maximum number of download attempts from a single URL until an error will be raised.
- *download_sleep_time*: Number of seconds between the failed download attempt and the next attempt.
- *download_timeout_seconds*: Maximum number of seconds before download attempt is canceled.
- *number_of_download_processes*: Number of download processes, used to calculate rate-limit sleep time.

Usage in the code:

- `SHConfig().sh_base_url`
- `SHConfig().instance_id`

save()
Method that saves configuration parameter changes from instance of SHConfig class to global config class and to *config.json* file.

Example `my_config = SHConfig()`

```
my_config.instance_id = '<new instance id>'
my_config.save()
```

reset(*params=Ellipsis*)
Resets configuration class to initial values. Use *SHConfig.save()* method in order to save this change.

Parameters **params** (*Ellipsis or list(str) or str*) – Parameters which will be reset. Parameters can be specified with a list of names, e.g. `['instance_id', 'aws_access_key_id', 'aws_secret_access_key']`, or as a single name, e.g. `'sh_base_url'`. By default all parameters will be reset and default value is `Ellipsis`.

get_params()
Returns a list of parameter names

Returns List of parameter names

Return type list(str)

get_config_dict()
Get a dictionary representation of *SHConfig* class

Returns A dictionary with configuration parameters

Return type dict

get_config_location()
Returns location of configuration file on disk

Returns File path of *config.json* file

Return type str

has_eocloud_url()
Checks if base Sentinel Hub URL is set to eocloud URL

Returns *True* if 'eocloud' string is in base OGC URL else *False*

Return type bool

get_sh_oauth_url()
Provides URL for Sentinel Hub authentication endpoint

Returns An URL endpoint

Return type str

get_sh_processing_api_url()
Provides URL for Sentinel Hub processing API endpoint

Returns An URL endpoint

Return type str

get_sh_ogc_url()
Provides URL for Sentinel Hub OGC endpoint

Returns An URL endpoint

Return type str

get_sh_rate_limit_url()
Provides URL for Sentinel Hub rate limiting endpoint

Returns An URL endpoint

Return type str

4.1.5 constants

Module defining constants and enumerate types used in the package

class sentinelhub.constants.**PackageProps**
Bases: object

Class for obtaining package properties. Currently it supports obtaining package version.

static get_version()
Returns package version

Returns package version

Return type str

class sentinelhub.constants.**ServiceUrl**
Bases: object

Most commonly used Sentinel Hub service URLs

class sentinelhub.constants.**ServiceType**
Bases: enum.Enum

Enum constant class for type of service

Supported types are WMS, WCS, WFS, AWS, IMAGE

class sentinelhub.constants.**CRSMeta**
Bases: enum.EnumMeta

Metaclass used for building CRS Enum class

This is executed at the beginning of runtime when CRS class is created

class sentinelhub.constants.**CRS**
Bases: enum.Enum

Coordinate Reference System enumerate class

Available CRS constants are WGS84, POP_WEB (i.e. Popular Web Mercator) and constants in form UTM_<zone><direction>, where zone is an integer from [1, 60] and direction is either N or S (i.e. northern or southern hemisphere)

epsg
EPSG code property

Returns EPSG code of given CRS

Return type int

ogc_string()
Returns a string of the form authority:id representing the CRS.

Parameters **self** (CRS) – An enum constant representing a coordinate reference system.

Returns A string representation of the CRS.

Return type str

opengis_string
Returns an URL to OGC webpage where the CRS is defined

Returns An URL with CRS definition

Return type str

is_utm()
Checks if crs is one of the 64 possible UTM coordinate reference systems.

Parameters **self** (CRS) – An enum constant representing a coordinate reference system.

Returns *True* if crs is UTM and *False* otherwise

Return type bool

projection
Returns a projection in form of pyproj class. For better time performance it will cache results of 5 most recently used CRS classes.

Returns pyproj projection class

Return type pyproj.Proj

pyproj_crs
Returns a pyproj CRS class. For better time performance it will cache results of 5 most recently used CRS classes.

Returns pyproj CRS class

Return type pyproj.CRS

get_transform_function
Returns a function for transforming geometrical objects from one CRS to another. The function will support transformations between any objects that pyproj supports. For better time performance this method will cache results of 10 most recently used pairs of CRS classes.

Parameters

- **self** (CRS) – Initial CRS
- **other** (CRS) – Target CRS

Returns A projection function obtained from pyproj package

Return type function

class sentinelhub.constants.**CustomUrlParam**
Bases: enum.Enum

Enum class to represent supported custom url parameters of OGC services

Supported parameters are *SHOWLOGO*, *ATMFILTER*, *EVALSCRIPT*, *EVALSCRIPTURL*, *PREVIEW*, *QUALITY*, *UPSAMPLING*, *DOWNSAMPLING*, *TRANSPARENT*, *BGCOLOR* and *GEOMETRY*.

See http://sentinel-hub.com/develop/documentation/api/custom-url-parameters and https://www.sentinel-hub.com/develop/documentation/api/ogc_api/wms-parameters for more information.

class sentinelhub.constants.**HistogramType**
Bases: enum.Enum

Enum class for types of histogram supported by Sentinel Hub FIS service

Supported histogram types are EQUALFREQUENCY, EQUIDISTANT and STREAMING

class sentinelhub.constants.**MimeType**
Bases: enum.Enum

Enum class to represent supported image file formats

Supported file formats are TIFF 8-bit, TIFF 16-bit, TIFF 32-bit float, PNG, JPEG, JPEG2000, JSON, CSV, ZIP, HDF5, XML, GML, RAW

extension
Returns file extension of the MimeType object

Returns A file extension string

Return type str

is_image_format()
Checks whether file format is an image format

Example: `MimeType.PNG.is_image_format()` or `MimeType.is_image_format(MimeType.PNG)`

Parameters **self** (MimeType) – File format

Returns *True* if file is in image format, *False* otherwise

Return type bool

is_api_format()
Checks if mime type is supported by Sentinel Hub API

Returns True if API supports this format and False otherwise

Return type bool

is_tiff_format()
Checks whether file format is a TIFF image format

Example: `MimeType.TIFF.is_tiff_format()` or `MimeType.is_tiff_format(MimeType.TIFF)`

Parameters **self** (MimeType) – File format

Returns *True* if file is in image format, *False* otherwise

Return type bool

get_string()
Get file format as string

Returns String describing the file format

Return type str

get_sample_type()
Returns sampleType used in Sentinel-Hub evalscripts.

Returns sampleType

Return type str

Raises ValueError

get_expected_max_value()
Returns max value of image *MimeType* format and raises an error if it is not an image format

Note: For *MimeType.TIFF_d32f* it will return `1.0` as that is expected maximum for an image even though it could be higher.

Returns A maximum value of specified image format

Return type int or float

Raises ValueError

class sentinelhub.constants.**RequestType**
Bases: enum.Enum

Enum constant class for GET/POST request type

class sentinelhub.constants.**SHConstants**
Bases: object

Initialisation of constants used by OGC request.

Constants are LATEST

class sentinelhub.constants.**AwsConstants**
Bases: object

Initialisation of every constant used by AWS classes

For each supported data collection it contains lists of all possible bands and all possible metadata files:

- S2_L1C_BANDS and S2_L1C_METAFILES
- S2_L2A_BANDS and S2_L2A_METAFILES

It also contains dictionary of all possible files and their formats: AWS_FILES

class sentinelhub.constants.**EsaSafeType**
Bases: enum.Enum

Enum constants class for ESA .SAFE type.

Types are OLD_TYPE and COMPACT_TYPE

4.1.6 data_collections

Module defining data collections

class sentinelhub.data_collections.**OrbitDirection**
Bases: object

Orbit directions

class sentinelhub.data_collections.**DataCollectionDefinition**(*api_id: str = None*, *wfs_id: str = None*, *service_url: str = None*, *collection_type: str = None*, *sensor_type: str = None*, *processing_level: str = None*, *swath_mode: str = None*, *polarization: str = None*, *resolution: str = None*, *orbit_direction: str = None*, *timeliness: str = None*, *bands: Tuple[str, ...] = None*, *collection_id: str = None*, *is_timeless: bool = False*)

Bases: object

An immutable definition of a data collection

Check *DataCollection.define* for more info about attributes of this class

derive(***params*)

Create a new data collection definition from current definition and parameters that override current parameters

Parameters **params** – Any of DataCollectionDefinition attributes

Returns A new data collection definition

Return type *DataCollectionDefinition*

class sentinelhub.data_collections.**DataCollection**

Bases: enum.Enum

An enum class for data collections

It contains a number of predefined data collections, which are the most commonly used with Sentinel Hub service. Additionally it also allows defining new data collections by specifying data collection parameters relevant for the service. Check *DataCollection.define* and similar methods for more.

define_from(*name*, ***params*)

Define a new data collection from an existing one

Parameters

- **name** (*str*) – A name of a new data collection
- **params** – Any parameter to override current data collection parameters

Returns A new data collection

Return type *DataCollection*

api_id

Provides a Sentinel Hub Processing API identifier or raises an error if it is not defined

Returns An identifier

Return type str

Raises ValueError

wfs_id
Provides a Sentinel Hub WFS identifier or raises an error if it is not defined

Returns An identifier

Return type str

Raises ValueError

bands
Provides band names available for the data collection

Returns A tuple of band names

Return type tuple(str)

Raises ValueError

is_sentinel1
Checks if data collection is a Sentinel-1 collection type

Example: `DataCollection.SENTINEL1_IW.is_sentinel1`

Returns *True* if collection is Sentinel-1 collection type and *False* otherwise

Return type bool

contains_orbit_direction(*orbit_direction*)
Checks if a data collection contains given orbit direction

Parameters **orbit_direction** (*string*) – An orbit direction

Returns *True* if data collection contains the orbit direction

Returns bool

sentinelhub.data_collections.**DataSource**
alias of *sentinelhub.data_collections.DataCollection*

sentinelhub.data_collections.**handle_deprecated_data_source**(*data_collection*, *data_source*, *default=None*)
Joins parameters used to specify a data collection. In case data_source is given it raises a warning. In case both are given it raises an error. In case neither are given but there is a default collection it raises another warning.

Note that this function is only temporary and will be removed in future package versions

4.1.7 data_request

Main module for collecting data

class sentinelhub.data_request.**DataRequest**(*download_client_class*, ***, *data_folder=None*, *config=None*, *instance_id=None*)
Bases: abc.ABC

Abstract class for all Sentinel Hub data requests.

Every data request type can write the fetched data to disk and then read it again (and hence avoid the need to download the same data again).

Parameters

- **download_client_class** (*type*) – A class implementing a download client
- **data_folder** (*str*) – location of the directory where the fetched data will be saved.
- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

create_request()
An abstract method for logic of creating download requests

get_download_list()
Returns a list of download requests for requested data.

Returns List of data to be downloaded

Return type list(sentinelhub.DownloadRequest)

get_filename_list()
Returns a list of file names (or paths relative to *data_folder*) where the requested data will be saved or read from, if it has already been downloaded and saved.

Returns A list of filenames

Return type list(str)

get_url_list()
Returns a list of urls for requested data.

Returns List of URLs from where data will be downloaded.

Return type list(str)

is_valid_request()
Checks if initialized class instance successfully prepared a list of items to download

Returns *True* if request is valid and *False* otherwise

Return type bool

get_data(*, *save_data=False*, *redownload=False*, *data_filter=None*, *max_threads=None*, *decode_data=True*, *raise_download_errors=True*)
Get requested data either by downloading it or by reading it from the disk (if it was previously downloaded and saved).

Parameters

- **save_data** (*bool*) – flag to turn on/off saving of data to disk. Default is *False*.
- **redownload** (*bool*) – if *True*, download again the requested data even though it's already saved to disk. Default is *False*, do not download if data is already available on disk.
- **data_filter** (*list(int) or None*) – Used to specify which items will be returned by the method and in which order. E.g. with `data_filter=[0, 2, -1]` the method will return only 1st, 3rd and last item. Default filter is *None*.
- **max_threads** (*int or None*) – Maximum number of threads to be used for download in parallel. The default is *max_threads=None* which will use the number of processors on the system multiplied by 5.
- **decode_data** (*bool*) – If *True* (default) it decodes data (e.g., returns image as an array of numbers); if *False* it returns binary data.
- **raise_download_errors** (*bool*) – If *True* any error in download process should be raised as `DownloadFailedException`. If *False* failed downloads will only raise

warnings and the method will return list with *None* values in places where the results of failed download requests should be.

Returns requested images as numpy arrays, where each array corresponds to a single acquisition and has shape `[height, width, channels]`.

Return type list of numpy arrays

save_data(*, *data_filter=None*, *redownload=False*, *max_threads=None*, *raise_download_errors=False*)

Saves data to disk. If `redownload=True` then the data is redownloaded using `max_threads` workers.

Parameters

- **data_filter** (*list(int) or None*) – Used to specify which items will be returned by the method and in which order. E.g. with *data_filter=[0, 2, -1]* the method will return only 1st, 3rd and last item. Default filter is *None*.
- **redownload** (*bool*) – data is redownloaded if `redownload=True`. Default is *False*
- **max_threads** (*int or None*) – Maximum number of threads to be used for download in parallel. The default is *max_threads=None* which will use the number of processors on the system multiplied by 5.
- **raise_download_errors** (*bool*) – If *True* any error in download process should be raised as `DownloadFailedException`. If *False* failed downloads will only raise warnings.

class sentinelhub.data_request.**OgcRequest**(*layer*, *bbox*, *, *time='latest'*, *service_type=None*, *data_collection=None*, *size_x=None*, *size_y=None*, *maxcc=1.0*, *image_format=<MimeType.PNG: 'png'>*, *custom_url_params=None*, *time_difference=datetime.timedelta(days=-1, seconds=86399)*, *data_source=None*, ***kwargs*)

Bases: *sentinelhub.data_request.DataRequest*

The base class for OGC-type requests (WMS and WCS) where all common parameters are defined

Parameters

- **layer** (*str*) – An ID of a layer configured in Sentinel Hub Configurator. It has to be configured for the same instance ID which will be used for this request. Also the satellite collection of the layer in Configurator must match the one given by *data_collection* parameter
- **bbox** (`geometry.BBox`) – Bounding box of the requested image. Coordinates must be in the specified coordinate reference system.
- **time** (*str or (str, str) or datetime.date or (datetime.date, datetime.date) or datetime.datetime or (datetime.datetime, datetime.datetime)*) – time or time range for which to return the results, in ISO8601 format (year-month-date, for example: `2016-01-01`, or year-month-dateThours:minutes:seconds format, i.e. `2016-01-01T16:31:21`). When a single time is specified the request will return data for that specific date, if it exists. If a time range is specified the result is a list of all scenes between the specified dates conforming to the cloud coverage criteria. Most recent acquisition being first in the list. For the latest acquisition use `latest`. Examples: `latest`, `'2016-01-01'`, or `('2016-01-01', ' 2016-01-31')`
- **service_type** (`constants.ServiceType`) – type of OGC service (WMS or WCS)

- **data_collection** (DataCollection) – A collection of requested satellite data. It has to be the same as defined in Sentinel Hub Configurator for the given layer.
- **size_x** (*int or str*) – number of pixels in x or resolution in x (i.e. 512 or 10m)
- **size_y** (*int or str*) – number of pixels in x or resolution in y (i.e. 512 or 10m)
- **maxcc** (*float*) – maximum accepted cloud coverage of an image. Float between 0.0 and 1.0. Default is 1.0.
- **image_format** (constants.MimeType) – format of the returned image by the Sentinel Hub's WMS getMap service. Default is PNG, but in some cases 32-bit TIFF is required, i.e. if requesting unprocessed raw bands. Default is constants.MimeType.PNG.
- **custom_url_params** (*Dict[CustomUrlParameter, object]*) – A dictionary of CustomUrlParameters and their values supported by Sentinel Hub's WMS and WCS services. All available parameters are described at http://www.sentinel-hub.com/develop/documentation/api/custom-url-parameters. Note: in case of *CustomUrlParam.EVALSCRIPT* the dictionary value must be a string of Javascript code that is not encoded into base64.
- **time_difference** (*datetime.timedelta*) – The time difference below which dates are deemed equal. That is, if for the given set of OGC parameters the images are available at datestimes *d1<=d2<=...<=dn* then only those with *dk-dj>time_difference* will be considered. The default time difference is negative (*-1s*), meaning that all dates are considered by default.
- **data_folder** (*str*) – location of the directory where the fetched data will be saved.
- **config** (SHConfig or None) – A custom instance of config class to override parameters from the saved configuration.
- **data_source** (DataCollection) – A deprecated alternative to data_collection

create_request(*reset_wfs_iterator=False*)

Set download requests

Create a list of DownloadRequests for all Sentinel-2 acquisitions within request's time interval and acceptable cloud coverage.

Parameters **reset_wfs_iterator** (*bool*) – When re-running the method this flag is used to reset/keep existing wfs_iterator (i.e. instance of WebFeatureService class). If the iterator is not reset you don't have to repeat a service call but tiles and dates will stay the same.

get_dates()

Get list of dates

List of all available Sentinel-2 acquisitions for given bbox with max cloud coverage and the specified time interval. When a single time is specified the request will return that specific date, if it exists. If a time range is specified the result is a list of all scenes between the specified dates conforming to the cloud coverage criteria. Most recent acquisition being first in the list.

Returns list of all available Sentinel-2 acquisition times within request's time interval and acceptable cloud coverage.

Return type list(datetime.datetime) or [None]

get_tiles()

Returns iterator over info about all satellite tiles used for the OgcRequest

Returns Iterator of dictionaries containing info about all satellite tiles used in the request. In case of DataCollection.DEM it returns None.

Return type Iterator[dict] or None

class sentinelhub.data_request.**WmsRequest**(*, *width=None*, *height=None*, ***kwargs*)

Bases: *sentinelhub.data_request.OgcRequest*

Web Map Service request class

Creates an instance of Sentinel Hub WMS (Web Map Service) GetMap request, which provides access to Sentinel-2's unprocessed bands (B01, B02, ..., B08, B8A, ..., B12) or processed products such as true color imagery, NDVI, etc. The only difference is that in the case of WMS request the user specifies the desired image size instead of its resolution.

It is required to specify at least one of *width* and *height* parameters. If only one of them is specified the the other one will be calculated to best fit the bounding box ratio. If both of them are specified they will be used no matter the bounding box ratio.

More info available at: https://www.sentinel-hub.com/develop/documentation/api/ogc_api/wms-parameters

Parameters

- **width** (*int or None*) – width (number of columns) of the returned image (array)
- **height** (*int or None*) – height (number of rows) of the returned image (array)
- **layer** (*str*) – An ID of a layer configured in Sentinel Hub Configurator. It has to be configured for the same instance ID which will be used for this request. Also the satellite collection of the layer in Configurator must match the one given by *data_collection* parameter
- **bbox** (geometry.BBox) – Bounding box of the requested image. Coordinates must be in the specified coordinate reference system.
- **time** (*str or (str, str) or datetime.date or (datetime.date, datetime.date) or datetime.datetime or (datetime.datetime, datetime.datetime)*) – time or time range for which to return the results, in ISO8601 format (year-month-date, for example: `2016-01-01`, or year-month-dateThours:minutes:seconds format, i.e. `2016-01-01T16:31:21`). When a single time is specified the request will return data for that specific date, if it exists. If a time range is specified the result is a list of all scenes between the specified dates conforming to the cloud coverage criteria. Most recent acquisition being first in the list. For the latest acquisition use `latest`. Examples: `latest`, `'2016-01-01'`, or `('2016-01-01', ' 2016-01-31')`
- **data_collection** (DataCollection) – A collection of requested satellite data. It has to be the same as defined in Sentinel Hub Configurator for the given layer. Default is Sentinel-2 L1C.
- **size_x** (*int or str*) – number of pixels in x or resolution in x (i.e. `512` or `10m`)
- **size_y** (*int or str*) – number of pixels in x or resolution in y (i.e. `512` or `10m`)
- **maxcc** (*float*) – maximum accepted cloud coverage of an image. Float between 0.0 and 1.0. Default is `1.0`.
- **image_format** (constants.MimeType) – format of the returned image by the Sentinel Hub's WMS getMap service. Default is PNG, but in some cases 32-bit TIFF is required, i.e. if requesting unprocessed raw bands. Default is `constants.MimeType.PNG`.
- **custom_url_params** (*Dict[CustomUrlParameter, object]*) – A dictionary of CustomUrlParameters and their values supported by Sentinel Hub's WMS and WCS services. All available parameters are described at http://www.sentinel-hub.

com/develop/documentation/api/custom-url-parameters. Note: in case of *CustomUrlParam.EVALSCRIPT* the dictionary value must be a string of Javascript code that is not encoded into base64.

- **time_difference** (*datetime.timedelta*) – The time difference below which dates are deemed equal. That is, if for the given set of OGC parameters the images are available at datestimes *d1<=d2<=...<=dn* then only those with *dk-dj>time_difference* will be considered. The default time difference is negative (*-1s*), meaning that all dates are considered by default.
- **data_folder** (*str*) – location of the directory where the fetched data will be saved.
- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.
- **data_source** (DataCollection) – A deprecated alternative to data_collection

class sentinelhub.data_request.**WcsRequest**(*, *resx='10m'*, *resy='10m'*, ***kwargs*)

Bases: *sentinelhub.data_request.OgcRequest*

Web Coverage Service request class

Creates an instance of Sentinel Hub WCS (Web Coverage Service) GetCoverage request, which provides access to Sentinel-2's unprocessed bands (B01, B02, ..., B08, B8A, ..., B12) or processed products such as true color imagery, NDVI, etc., as the WMS service. The only difference is that in the case of WCS request the user specifies the desired resolution of the image instead of its size.

More info available at: https://www.sentinel-hub.com/develop/documentation/api/ogc_api/wcs-request

Parameters

- **resx** (*str*) – resolution in x (resolution of a column) given in meters in the format (examples `10m`, `20m`, ...). Default is `10m`, which is the best native resolution of some Sentinel-2 bands.
- **resy** (*str*) – resolution in y (resolution of a row) given in meters in the format (examples `10m`, `20m`, ...). Default is `10m`, which is the best native resolution of some Sentinel-2 bands.
- **layer** (*str*) – An ID of a layer configured in Sentinel Hub Configurator. It has to be configured for the same instance ID which will be used for this request. Also the satellite collection of the layer in Configurator must match the one given by *data_collection* parameter
- **bbox** (geometry.BBox) – Bounding box of the requested image. Coordinates must be in the specified coordinate reference system.
- **time** (*str or (str, str) or datetime.date or (datetime.date, datetime.date) or datetime.datetime or (datetime.datetime, datetime.datetime)*) – time or time range for which to return the results, in ISO8601 format (year-month-date, for example: `2016-01-01`, or year-month-dateThours:minutes:seconds format, i.e. `2016-01-01T16:31:21`). When a single time is specified the request will return data for that specific date, if it exists. If a time range is specified the result is a list of all scenes between the specified dates conforming to the cloud coverage criteria. Most recent acquisition being first in the list. For the latest acquisition use `latest`. Examples: `latest`, `'2016-01-01'`, or `('2016-01-01', ' 2016-01-31')`
- **data_collection** (DataCollection) – A collection of requested satellite data. It has to be the same as defined in Sentinel Hub Configurator for the given layer. Default is Sentinel-2 L1C.

- **size_x** (*int or str*) – number of pixels in x or resolution in x (i.e. `512` or `10m`)
- **size_y** (*int or str*) – number of pixels in x or resolution in y (i.e. `512` or `10m`)
- **maxcc** (*float*) – maximum accepted cloud coverage of an image. Float between 0.0 and 1.0. Default is `1.0`.
- **image_format** (`constants.MimeType`) – format of the returned image by the Sentinel Hub's WMS getMap service. Default is PNG, but in some cases 32-bit TIFF is required, i.e. if requesting unprocessed raw bands. Default is `constants.MimeType.PNG`.
- **custom_url_params** (*Dict[CustomUrlParameter, object]*) – A dictionary of CustomUrlParameters and their values supported by Sentinel Hub's WMS and WCS services. All available parameters are described at http://www.sentinel-hub.com/develop/documentation/api/custom-url-parameters. Note: in case of *CustomUrlParam.EVALSCRIPT* the dictionary value must be a string of Javascript code that is not encoded into base64.
- **time_difference** (*datetime.timedelta*) – The time difference below which dates are deemed equal. That is, if for the given set of OGC parameters the images are available at datestimes *d1<=d2<=...<=dn* then only those with *dk-dj>time_difference* will be considered. The default time difference is negative (*-1s*), meaning that all dates are considered by default.
- **data_folder** (*str*) – location of the directory where the fetched data will be saved.
- **config** (`SHConfig` *or None*) – A custom instance of config class to override parameters from the saved configuration.
- **data_source** (`DataCollection`) – A deprecated alternative to data_collection

class sentinelhub.data_request.**FisRequest**(*layer*, *time*, *geometry_list*, ***, *resolution='10m'*, *bins=None*, *histogram_type=None*, ***kwargs*)

Bases: *sentinelhub.data_request.OgcRequest*

The Statistical info (or feature info service, abbreviated FIS) request class

The Statistical info (or feature info service, abbreviated FIS), performs elementary statistical computations—such as mean, standard deviation, and histogram approximating the distribution of reflectance values—on remotely sensed data for a region specified in a given spatial reference system across different bands and time ranges.

A quintessential usage example would be querying the service for basic statistics and the distribution of NDVI values for a polygon representing an agricultural unit over a time range.

More info available at: https://www.sentinel-hub.com/develop/documentation/api/ogc_api/wcs-request

Parameters

- **layer** (*str*) – An ID of a layer configured in Sentinel Hub Configurator. It has to be configured for the same instance ID which will be used for this request. Also the satellite collection of the layer in Configurator must match the one given by *data_collection* parameter
- **time** (*str or (str, str) or datetime.date or (datetime.date, datetime.date) or datetime.datetime or (datetime.datetime, datetime.datetime)*) – time or time range for which to return the results, in ISO8601 format (year-month-date, for example: `2016-01-01`, or year-month-dateThours:minutes:seconds format, i.e. `2016-01-01T16:31:21`). Examples: `'2016-01-01'`, or `('2016-01-01', ' 2016-01-31')`

- **geometry_list** (*list, [geometry.Geometry or geometry.Bbox]*) – A WKT representation of a geometry describing the region of interest. Note that WCS 1.1.1 standard is used here, so for EPSG:4326 coordinates should be in latitude/longitude order.
- **resolution** (*str*) – Specifies the spatial resolution, in meters per pixel, of the image from which the statistics are to be estimated. When using CRS=EPSG:4326 one has to add the "m" suffix to enforce resolution in meters per pixel (e.g. RESOLUTION=10m).
- **bins** (*str*) – The number of bins (a positive integer) in the histogram. If this parameter is absent no histogram is computed.
- **histogram_type** (HistogramType) – type of histogram
- **data_collection** (DataCollection) – A collection of requested satellite data. It has to be the same as defined in Sentinel Hub Configurator for the given layer. Default is Sentinel-2 L1C.
- **maxcc** (*float*) – maximum accepted cloud coverage of an image. Float between 0.0 and 1.0. Default is `1.0`.
- **custom_url_params** (*Dict[CustomUrlParameter, object]*) – Dictionary of CustomUrlParameters and their values supported by Sentinel Hub's WMS and WCS services. All available parameters are described at http://www.sentinel-hub.com/develop/documentation/api/custom-url-parameters. Note: in case of constants.CustomUrlParam.EVALSCRIPT the dictionary value must be a string of Javascript code that is not encoded into base64.
- **data_folder** (*str*) – location of the directory where the fetched data will be saved.
- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.
- **data_source** (DataCollection) – A deprecated alternative to data_collection

create_request()
Set download requests

Create a list of DownloadRequests for all Sentinel-2 acquisitions within request's time interval and acceptable cloud coverage.

get_dates()
This method is not supported for FIS request

get_tiles()
This method is not supported for FIS request

class sentinelhub.data_request.**GeopediaRequest**(*layer*, *service_type*, ***, *bbox=None*, *theme=None*, *image_format=<MimeType.PNG: 'png'>*, ***kwargs*)

Bases: *sentinelhub.data_request.DataRequest*

The base class for Geopedia requests where all common parameters are defined.

Parameters

- **layer** (*str*) – Geopedia layer which contains requested data
- **service_type** (constants.ServiceType) – Type of the service, supported are `ServiceType.WMS` and `ServiceType.IMAGE`
- **bbox** (geometry.BBox) – Bounding box of the requested data

- **theme** (*str*) – Geopedia's theme endpoint string for which the layer is defined. Only required by WMS service.
- **image_format** (constants.MimeType) – Format of the returned image by the Sentinel Hub's WMS getMap service. Default is constants.MimeType.PNG.
- **data_folder** (*str*) – Location of the directory where the fetched data will be saved.
- **config** (SHConfig *or None*) – A custom instance of config class to override parameters from the saved configuration.

create_request()
An abstract method for logic of creating download requests

class sentinelhub.data_request.**GeopediaWmsRequest**(*layer*, *theme*, *bbox*, *, *width=None*, *height=None*, *custom_url_params=None*, ***kwargs*)
Bases: *sentinelhub.data_request.GeopediaRequest*

Web Map Service request class for Geopedia

Creates an instance of Geopedia's WMS (Web Map Service) GetMap request, which provides access to WMS layers in Geopedia.

Parameters

- **layer** (*str*) – Geopedia layer which contains requested data
- **theme** (*str*) – Geopedia's theme endpoint string for which the layer is defined.
- **bbox** (geometry.BBox) – Bounding box of the requested data
- **width** (*int or None*) – width (number of columns) of the returned image (array)
- **height** (*int or None*) – height (number of rows) of the returned image (array)
- **custom_url_params** (*Dict[CustomUrlParameter, object]*) – dictionary of CustomUrlParameters and their values supported by Geopedia's WMS services. At the moment only the transparency is supported (CustomUrlParam.TRANSPARENT).
- **image_format** (constants.MimeType) – Format of the returned image by the Sentinel Hub's WMS getMap service. Default is constants.MimeType.PNG.
- **data_folder** (*str*) – Location of the directory where the fetched data will be saved.
- **config** (SHConfig *or None*) – A custom instance of config class to override parameters from the saved configuration.

create_request()
Set download requests

Create a list of DownloadRequests for all Sentinel-2 acquisitions within request's time interval and acceptable cloud coverage.

class sentinelhub.data_request.**GeopediaImageRequest**(*, *image_field_name*, *keep_image_names=True*, *gpd_session=None*, ***kwargs*)
Bases: *sentinelhub.data_request.GeopediaRequest*

Request to access data in a Geopedia vector / raster layer.

Parameters

- **image_field_name** (*str*) – Name of the field in the data table which holds images

- **keep_image_names** (*bool*) – If *True* images will be saved with the same names as in Geopedia otherwise Geopedia hashes will be used as names. If there are multiple images with the same names in the Geopedia layer this parameter should be set to *False* to prevent images being overwritten.
- **layer** (*str*) – Geopedia layer which contains requested data
- **bbox** (geometry.BBox) – Bounding box of the requested data
- **image_format** (constants.MimeType) – Format of the returned image by the Sentinel Hub's WMS getMap service. Default is constants.MimeType.PNG.
- **gpd_session** (GeopediaSession *or None*) – Optional parameter for specifying a custom Geopedia session, which can also contain login credentials. This can be used for accessing private Geopedia layers. By default it is set to *None* and a basic Geopedia session without credentials will be created.
- **data_folder** (*str*) – Location of the directory where the fetched data will be saved.
- **config** (SHConfig *or None*) – A custom instance of config class to override parameters from the saved configuration.

create_request(*reset_gpd_iterator=False*)

Set a list of download requests

Set a list of DownloadRequests for all images that are under the given property of the Geopedia's Vector layer.

Parameters **reset_gpd_iterator** (*bool*) – When re-running the method this flag is used to reset/keep existing gpd_iterator (i.e. instance of GeopediaFeatureIterator class). If the iterator is not reset you don't have to repeat a service call but tiles and dates will stay the same.

get_items()

Returns iterator over info about data used for this request

Returns Iterator of dictionaries containing info about data used in this request.

Return type Iterator[dict] or None

class sentinelhub.data_request.**AwsRequest**(*, *bands=None*, *metafiles=None*, *safe_format=False*, ***kwargs*)

Bases: *sentinelhub.data_request.DataRequest*

The base class for Amazon Web Service request classes. Common parameters are defined here.

Collects and provides data from AWS.

AWS database is available at: http://sentinel-s2-l1c.s3-website.eu-central-1.amazonaws.com/

Parameters

- **bands** (*list(str) or None*) – List of Sentinel-2 bands for request. If *None* all bands will be obtained
- **metafiles** (*list(str)*) – list of additional metafiles available on AWS (e.g. ['metadata', 'tileInfo', 'preview/B01', 'TCI'])
- **safe_format** (*bool*) – flag that determines the structure of saved data. If *True* it will be saved in .SAFE format defined by ESA. If *False* it will be saved in the same structure as the structure at AWS.
- **data_folder** (*str*) – location of the directory where the fetched data will be saved.

- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

create_request()
An abstract method for logic of creating download requests

get_aws_service()

Returns initialized AWS service class

Return type *aws.AwsProduct* or *aws.AwsTile* or *aws_safe.SafeProduct* or *aws_safe.SafeTile*

class sentinelhub.data_request.**AwsProductRequest**(*product_id*, *, *tile_list=None*, ***kwargs*)

Bases: *sentinelhub.data_request.AwsRequest*

AWS Service request class for an ESA product

List of available products: http://sentinel-s2-l1c.s3-website.eu-central-1.amazonaws.com/#products/

Parameters

- **product_id** (*str*) – original ESA product identification string (e.g. `'S2A_MSIL1C_20170414T003551_N0204_R016_T54HVH_20170414T003551'`)
- **tile_list** (*list(str) or None*) – list of tiles inside the product to be downloaded. If parameter is set to *None* all tiles inside the product will be downloaded.
- **bands** (*list(str) or None*) – List of Sentinel-2 bands for request. If *None* all bands will be obtained
- **metafiles** (*list(str)*) – list of additional metafiles available on AWS (e.g. `['metadata', 'tileInfo', 'preview/B01', 'TCI']`)
- **safe_format** (*bool*) – flag that determines the structure of saved data. If *True* it will be saved in .SAFE format defined by ESA. If *False* it will be saved in the same structure as the structure at AWS.
- **data_folder** (*str*) – location of the directory where the fetched data will be saved.
- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

create_request()
An abstract method for logic of creating download requests

class sentinelhub.data_request.**AwsTileRequest**(*, *tile=None*, *time=None*, *aws_index=None*, *data_collection=None*, *data_source=None*, ***kwargs*)

Bases: *sentinelhub.data_request.AwsRequest*

AWS Service request class for an ESA tile

List of available products: http://sentinel-s2-l1c.s3-website.eu-central-1.amazonaws.com/#tiles/

Parameters

- **tile** (*str*) – tile name (e.g. `'T10UEV'`)
- **time** (*str*) – tile sensing time in ISO8601 format
- **aws_index** (*int or None*) – there exist Sentinel-2 tiles with the same tile and time parameter. Therefore each tile on AWS also has an index which is visible in their url path. If aws_index is set to *None* the class will try to find the index automatically. If there will be multiple choices it will choose the lowest index and inform the user.

- **data_collection** (DataCollection) – A collection of requested AWS data. Supported collections are Sentinel-2 L1C and Sentinel-2 L2A.
- **bands** (*list(str) or None*) – List of Sentinel-2 bands for request. If *None* all bands will be obtained
- **metafiles** (*list(str)*) – list of additional metafiles available on AWS (e.g. `['metadata', 'tileInfo', 'preview/B01', 'TCI']`)
- **safe_format** (*bool*) – flag that determines the structure of saved data. If *True* it will be saved in .SAFE format defined by ESA. If *False* it will be saved in the same structure as the structure at AWS.
- **data_folder** (*str*) – location of the directory where the fetched data will be saved.
- **config** (SHConfig *or None*) – A custom instance of config class to override parameters from the saved configuration.
- **data_source** (DataCollection) – A deprecated alternative to data_collection

create_request()

An abstract method for logic of creating download requests

sentinelhub.data_request.**get_safe_format**(*product_id=None*, *tile=None*, *entire_product=False*, *bands=None*, *data_collection=None*, *data_source=None*)

Returns .SAFE format structure in form of nested dictionaries. Either `product_id` or `tile` must be specified.

Parameters

- **product_id** (*str*) – original ESA product identification string. Default is *None*
- **tile** ((*str, str*)) – tuple containing tile name and sensing time/date. Default is *None*
- **entire_product** (*bool*) – in case tile is specified this flag determines if it will be place inside a .SAFE structure of the product. Default is *False*
- **bands** (*list(str) or None*) – list of bands to download. If *None* all bands will be downloaded. Default is *None*
- **data_collection** (DataCollection) – In case of tile request the collection of satellite data has to be specified.
- **data_source** (DataCollection) – A deprecated alternative to data_collection

Returns Nested dictionaries representing .SAFE structure.

Return type dict

sentinelhub.data_request.**download_safe_format**(*product_id=None*, *tile=None*, *folder='.'*, *redownload=False*, *entire_product=False*, *bands=None*, *data_collection=None*, *data_source=None*)

Downloads .SAFE format structure in form of nested dictionaries. Either `product_id` or `tile` must be specified.

Parameters

- **product_id** (*str*) – original ESA product identification string. Default is *None*
- **tile** ((*str, str*)) – tuple containing tile name and sensing time/date. Default is *None*
- **folder** (*str*) – location of the directory where the fetched data will be saved. Default is `'.'`

- **redownload** (*bool*) – if *True*, download again the requested data even though it's already saved to disk. If *False*, do not download if data is already available on disk. Default is *False*
- **entire_product** (*bool*) – in case tile is specified this flag determines if it will be place inside a .SAFE structure of the product. Default is *False*
- **bands** (*list(str) or None*) – list of bands to download. If *None* all bands will be downloaded. Default is *None*
- **data_collection** (DataCollection) – In case of tile request the collection of satellite data has to be specified.
- **data_source** (DataCollection) – A deprecated alternative to data_collection

Returns Nested dictionaries representing .SAFE structure.

Return type dict

4.1.8 download.aws_client

Module implementing download client that is adjusted to download from AWS

class sentinelhub.download.aws_client.**AwsDownloadClient**(*, *redownload=False*, *raise_download_errors=True*, *config=None*)

Bases: *sentinelhub.download.client.DownloadClient*

An AWS download client class

Parameters

- **redownload** (*bool*) – If *True* the data will always be downloaded again. By default this is set to *False* and the data that has already been downloaded and saved to an expected location will be read from the location instead of being downloaded again.
- **raise_download_errors** (*bool*) – If *True* any error in download process will be raised as *DownloadFailedException*. If *False* failed downloads will only raise warnings.
- **config** (SHConfig) – An instance of configuration class

static is_s3_request(*request*)

Checks if data has to be downloaded from AWS s3 bucket

Returns *True* if url describes location at AWS s3 bucket and *False* otherwise

Return type bool

4.1.9 download.client

Module implementing the main download client class

class sentinelhub.download.client.**DownloadClient**(*, *redownload=False*, *raise_download_errors=True*, *config=None*)

Bases: object

A basic download client object

It does the following:

- downloads the data with multiple threads in parallel,
- handles any exceptions that occur during download,

- decodes downloaded data,
- reads and writes locally stored/cached data

Parameters

- **redownload** (*bool*) – If *True* the data will always be downloaded again. By default this is set to *False* and the data that has already been downloaded and saved to an expected location will be read from the location instead of being downloaded again.
- **raise_download_errors** (*bool*) – If *True* any error in download process will be raised as *DownloadFailedException*. If *False* failed downloads will only raise warnings.
- **config** (SHConfig) – An instance of configuration class

download(*download_requests*, *max_threads=None*, *decode_data=True*)
Download one or multiple requests, provided as a request list.

Parameters

- **download_requests** (*List[DownloadRequest] or DownloadRequest*) – A list of requests or a single request to be executed.
- **max_threads** (*int or None*) – Maximum number of threads to be used for download in parallel. The default is *max_threads=None* which will use the number of processors on the system multiplied by 5.
- **decode_data** (*bool*) – If *True* it will decode data otherwise it will return it in binary format.

Returns A list of results or a single result, depending on input parameter *download_requests*

Return type list(object) or object

get_json(*url*, *post_values=None*, *headers=None*, *request_type=None*, ***kwargs*)
Download request as JSON data type

Parameters

- **url** (*str*) – An URL from where the data will be downloaded
- **post_values** (*dict or None*) – A dictionary of parameters for a POST request
- **headers** (*dict*) – A dictionary of additional request headers
- **request_type** (*RequestType or None*) – A type of HTTP request to make. If not specified, then it will be a GET request if *post_values=None* and a POST request otherwise
- **kwargs** – Any other parameters that are passed to DownloadRequest class

Returns JSON data parsed into Python objects

Return type dict or list or str or None

get_xml(*url*, ***kwargs*)
Download request as XML data type

Parameters

- **url** (*str*) – url to Sentinel Hub's services or other sources from where the data is downloaded
- **kwargs** – Any other parameters that are passed to DownloadRequest class

Returns request response as XML instance

Return type XML instance or None

sentinelhub.download.client.**get_json**(*url*, *post_values=None*, *headers=None*, *request_type=None*, *download_client_class=<class 'sentinelhub.download.client.DownloadClient'>*, ***kwargs*)

Download request as JSON data type

Parameters

- **url** (*str*) – An URL from where the data will be downloaded
- **post_values** (*dict or None*) – A dictionary of parameters for a POST request
- **headers** (*dict*) – A dictionary of additional request headers
- **request_type** (*RequestType or None*) – A type of HTTP request to make. If not specified, then it will be a GET request if *post_values=None* and a POST request otherwise
- **download_client_class** (*object*) – A class that implements a download client
- **kwargs** – Parameters that are passed to a DownloadRequest class

Returns JSON data parsed into Python objects

Return type object or None

sentinelhub.download.client.**get_xml**(*url*, *download_client_class=<class 'sentinelhub.download.client.DownloadClient'>*)

Download request as XML data type

Parameters

- **url** (*str*) – url to Sentinel Hub's services or other sources from where the data is downloaded
- **download_client_class** (*object*) – A class that implements a download client

Returns request response as XML instance

Return type XML instance or None

Raises RunTimeError

4.1.10 download.request

Module implementing DownloadRequest class

class sentinelhub.download.request.**DownloadRequest**(***, *url=None*, *headers=None*, *request_type=<RequestType.GET: 'GET'>*, *post_values=None*, *use_session=False*, *data_type=<MimeType.RAW: 'raw'>*, *save_response=False*, *data_folder=None*, *filename=None*, *return_data=True*, ***properties*)

Bases: object

A class defining a single download request.

The class is a container with all parameters needed to execute a single download request and save or return downloaded data.

Parameters

- **url** (*str or None*) – An URL from where to download
- **headers** (*dict or None*) – Headers of a HTTP request
- **request_type** (*str or RequestType*) – Type of request, either GET or POST. Default is *RequestType.GET*
- **post_values** (*dict or None*) – A dictionary of values that will be sent with a POST request. Default is *None*
- **use_session** (*bool*) – A flag that specifies if the download request will require a session header
- **data_type** (*MimeType or str*) – An expected file format of downloaded data. Default is *MimeType.RAW*
- **save_response** (*bool*) – A flag defining if the downloaded data will be saved to disk. Default is *False*.
- **data_folder** (*str or None*) – A folder path where the fetched data will be (or already is) saved. Default is *None*
- **filename** (*str or None*) – A custom filename where the data will be saved. By default data will be saved in a folder which name are hashed request parameters.
- **return_data** (*bool*) – A flag defining if the downloaded data will be returned as an output of download procedure. Default is *True*.
- **properties** – Any additional parameters.

raise_if_invalid()
Method that raises an error if something is wrong with request parameters

Raises ValueError

get_request_params(*include_metadata=False*)
Provides parameters that define the request in form of a dictionary

Parameters **include_metadata** (*bool*) – A flag defining if also metadata parameters should be included, such as headers and current time

Returns A dictionary of parameters

Return type dict

get_hashed_name()
It takes request url and payload and calculates a unique hashed string from them.

Returns A hashed string

Return type str

get_relative_paths()
A method that calculates file paths relative to *data_folder*

Returns Returns a pair of file paths, a request payload path and a response path. If request path is not defined it returns *None*.

Return type (str or None, str)

get_storage_paths()
A method that calculates file paths where request payload and response will be saved.

Returns Returns a pair of file paths, a request payload path and a response path. Each of them can also be *None* if it is not defined.

Return type (str or None, str or None)

4.1.11 download.sentinelhub_client

Module implementing a rate-limited multi-threaded download client for downloading from Sentinel Hub service

class sentinelhub.download.sentinelhub_client.**SentinelHubDownloadClient**(*, *session=None*, ***kwargs*)

Bases: *sentinelhub.download.client.DownloadClient*

Download client specifically configured for download from Sentinel Hub service

Parameters

- **session** (*SentinelHubSession or None*) – An OAuth2 session with Sentinel Hub service
- **kwargs** – Optional parameters from DownloadClient

4.1.12 fis

Module for working with Sentinel Hub FIS service

class sentinelhub.fis.**FisService**(***kwargs*)

Bases: *sentinelhub.ogc.OgcImageService*

Sentinel Hub OGC services class for providing FIS data

Intermediate layer between FIS requests and the Sentinel Hub FIS services.

Parameters **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

get_request(*request*)

Get download requests

Create a list of DownloadRequests for all Sentinel-2 acquisitions within request's time interval and acceptable cloud coverage.

Parameters **request** (*OgcRequest or GeopediaRequest*) – OGC-type request with specified bounding box, time interval, and cloud coverage for specific product.

Returns list of DownloadRequests

4.1.13 geo_utils

Module for manipulation of geographical information

sentinelhub.geo_utils.**bbox_to_dimensions**(*bbox*, *resolution*)

Calculates width and height in pixels for a given bbox of a given pixel resolution (in meters). The result is rounded to nearest integers

Parameters

- **bbox** (*geometry.BBox*) – bounding box

- **resolution** (*float or (float, float)*) – Resolution of desired image in meters. It can be a single number or a tuple of two numbers - resolution in horizontal and resolution in vertical direction.

Returns width and height in pixels for given bounding box and pixel resolution

Return type int, int

sentinelhub.geo_utils.**bbox_to_resolution**(*bbox*, *width*, *height*, *meters=True*)
Calculates pixel resolution for a given bbox of a given width and height. By default it returns result in meters.

Parameters

- **bbox** (geometry.BBox) – bounding box
- **width** (*int*) – width of bounding box in pixels
- **height** (*int*) – height of bounding box in pixels
- **meters** (*bool*) – If *True* result will be given in meters, otherwise it will be given in units of current CRS

Returns resolution east-west at north and south, and resolution north-south for given CRS

Return type float, float

Raises ValueError if CRS is not supported

sentinelhub.geo_utils.**get_image_dimension**(*bbox*, *width=None*, *height=None*)
Given bounding box and one of the parameters width or height it will return the other parameter that will best fit the bounding box dimensions

Parameters

- **bbox** (geometry.BBox) – bounding box
- **width** (*int or None*) – image width or *None* if height is unknown
- **height** (*int or None*) – image height or *None* if height is unknown

Returns width or height rounded to integer

Return type int

sentinelhub.geo_utils.**to_utm_bbox**(*bbox*)
Transform bbox into UTM CRS

Parameters **bbox** (geometry.BBox) – bounding box

Returns bounding box in UTM CRS

Return type *geometry.BBox*

sentinelhub.geo_utils.**get_utm_bbox**(*img_bbox*, *transform*)
Get UTM coordinates given a bounding box in pixels and a transform

Parameters

- **img_bbox** (*list*) – boundaries of bounding box in pixels as *[row1, col1, row2, col2]*
- **transform** (*tuple or list*) – georeferencing transform of the image, e.g. *(x_upper_left, res_x, 0, y_upper_left, 0, -res_y)*

Returns UTM coordinates as [east1, north1, east2, north2]

Return type list

sentinelhub.geo_utils.**wgs84_to_utm**(*lng*, *lat*, *utm_crs=None*)
Convert WGS84 coordinates to UTM. If UTM CRS is not set it will be calculated automatically.

Parameters

- **lng** (*float*) – longitude in WGS84 system
- **lat** (*float*) – latitude in WGS84 system
- **utm_crs** (*constants.CRS or None*) – UTM coordinate reference system enum constants

Returns east, north coordinates in UTM system

Return type float, float

sentinelhub.geo_utils.**to_wgs84**(*east*, *north*, *crs*)
Convert any CRS with (east, north) coordinates to WGS84

Parameters

- **east** (*float*) – east coordinate
- **north** (*float*) – north coordinate
- **crs** (*constants.CRS*) – CRS enum constants

Returns latitude and longitude coordinates in WGS84 system

Return type float, float

sentinelhub.geo_utils.**utm_to_pixel**(*east*, *north*, *transform*, *truncate=True*)
Convert UTM coordinate to image coordinate given a transform

Parameters

- **east** (*float*) – east coordinate of point
- **north** (*float*) – north coordinate of point
- **transform** (*tuple or list*) – georeferencing transform of the image, e.g. *(x_upper_left, res_x, 0, y_upper_left, 0, -res_y)*
- **truncate** (*bool*) – Whether to truncate pixel coordinates. Default is *True*

Returns row and column pixel image coordinates

Return type float, float or int, int

sentinelhub.geo_utils.**pixel_to_utm**(*row*, *column*, *transform*)
Convert pixel coordinate to UTM coordinate given a transform

Parameters

- **row** (*int or float*) – row pixel coordinate
- **column** (*int or float*) – column pixel coordinate
- **transform** (*tuple or list*) – georeferencing transform of the image, e.g. *(x_upper_left, res_x, 0, y_upper_left, 0, -res_y)*

Returns east, north UTM coordinates

Return type float, float

sentinelhub.geo_utils.**wgs84_to_pixel**(*lng*, *lat*, *transform*, *utm_epsg=None*, *truncate=True*)
Convert WGS84 coordinates to pixel image coordinates given transform and UTM CRS. If no CRS is given it will be calculated it automatically.

Parameters

- **lng** (*float*) – longitude of point
- **lat** (*float*) – latitude of point
- **transform** (*tuple or list*) – georeferencing transform of the image, e.g. *(x_upper_left, res_x, 0, y_upper_left, 0, -res_y)*
- **utm_epsg** (constants.CRS *or None*) – UTM coordinate reference system enum constants
- **truncate** (*bool*) – Whether to truncate pixel coordinates. Default is *True*

Returns row and column pixel image coordinates

Return type float, float or int, int

sentinelhub.geo_utils.**get_utm_crs**(*lng*, *lat*, *source_crs=CRS('4326')*)

Get CRS for UTM zone in which (lat, lng) is contained.

Parameters

- **lng** (*float*) – longitude
- **lat** (*float*) – latitude
- **source_crs** (constants.CRS) – source CRS

Returns CRS of the zone containing the lat,lon point

Return type *constants.CRS*

sentinelhub.geo_utils.**transform_point**(*point*, *source_crs*, *target_crs*)

Maps point form src_crs to tgt_crs

Parameters

- **point** (*(float, float)*) – a tuple *(x, y)*
- **source_crs** (constants.CRS) – source CRS
- **target_crs** (constants.CRS) – target CRS

Returns point in target CRS

Return type (float, float)

4.1.14 geometry

Module implementing geometry classes

class sentinelhub.geometry.**BaseGeometry**(*crs*)

Bases: abc.ABC

Base geometry class

Parameters **crs** (constants.CRS) – Coordinate reference system of the geometry

crs

Returns the coordinate reference system (CRS)

Returns Coordinate reference system Enum

Return type *constants.CRS*

geometry
An abstract property - ever subclass must implement geometry property

geojson
Returns representation in a GeoJSON format. Use json.dump for writing it to file.

Returns A dictionary in GeoJSON format

Return type dict

get_geojson()
Returns representation in a GeoJSON format. Use json.dump for writing it to file.

Returns A dictionary in GeoJSON format

Return type dict

wkt
Transforms geometry object into *Well-known text* format

Returns string in WKT format

Return type str

class sentinelhub.geometry.**BBox**(*bbox*, *crs*)
Bases: *sentinelhub.geometry.BaseGeometry*

Class representing a bounding box in a given CRS.

Throughout the sentinelhub package this class serves as the canonical representation of a bounding box. It can initialize itself from multiple representations:

1) `((min_x,min_y),(max_x,max_y))`,
2) `(min_x,min_y,max_x,max_y)`,
3) `[min_x,min_y,max_x,max_y]`,
4) `[[min_x, min_y],[max_x,max_y]]`,
5) `[(min_x, min_y),(max_x,max_y)]`,
6) `([min_x, min_y],[max_x,max_y])`,
7) `'min_x,min_y,max_x,max_y'`,
8) `{'min_x':min_x, 'max_x':max_x, 'min_y':min_y, 'max_y':max_y}`,
9) `bbox`, where `bbox` is an instance of `BBox`.

Note that BBox coordinate system depends on `crs` parameter:

- In case of `constants.CRS.WGS84` axis x represents longitude and axis y represents latitude
- In case of `constants.CRS.POP_WEB` axis x represents easting and axis y represents northing
- In case of `constants.CRS.UTM_*` axis x represents easting and axis y represents northing

Parameters

- **bbox** – A bbox in any valid representation
- **crs** (`constants.CRS`) – Coordinate reference system of the bounding box

lower_left
Returns the lower left vertex of the bounding box

Returns min_x, min_y

Return type (float, float)

upper_right
Returns the upper right vertex of the bounding box

Returns max_x, max_y

Return type (float, float)

middle
Returns the middle point of the bounding box

Returns middle point

Return type (float, float)

reverse()
Returns a new BBox object where x and y coordinates are switched

Returns New BBox object with switched coordinates

Return type *BBox*

transform(*crs*)
Transforms BBox from current CRS to target CRS

Parameters **crs** (constants.CRS) – target CRS

Returns Bounding box in target CRS

Return type *BBox*

buffer(*buffer*)
Changes both BBox dimensions (width and height) by a percentage of size of each dimension. If number is negative, the size will decrease. Returns a new instance of BBox object.

Parameters **buffer** (*float*) – A percentage of BBox size change

Returns A new bounding box of buffered size

Return type *BBox*

get_polygon(*reverse=False*)
Returns a tuple of coordinates of 5 points describing a polygon. Points are listed in clockwise order, first point is the same as the last.

Parameters **reverse** (*bool*) – *True* if x and y coordinates should be switched and *False* otherwise

Returns *((x_1, y_1), . . . , (x_5, y_5))*

Return type tuple(tuple(float))

geometry
Returns polygon geometry in shapely format

Returns A polygon in shapely format

Return type shapely.geometry.polygon.Polygon

get_partition(*num_x=None*, *num_y=None*, *size_x=None*, *size_y=None*)
Partitions bounding box into smaller bounding boxes of the same size.

If *num_x* and *num_y* are specified, the total number of BBoxes is know but not the size. If *size_x* and *size_y* are provided, the BBox size is fixed but the number of BBoxes is not known in advance. In the latter case, the generated bounding boxes might cover an area larger than the parent BBox.

Parameters

- **num_x** (*int or None*) – Number of parts BBox will be horizontally divided into.
- **num_y** (*int or None*) – Number of parts BBox will be vertically divided into.
- **size_x** (*float or None*) – Physical dimension of BBox along easting coordinate
- **size_y** (*float or None*) – Physical dimension of BBox along northing coordinate

Returns Two-dimensional list of smaller bounding boxes. Their location is

Return type list(list(*BBox*))

get_transform_vector(*resx*, *resy*)

Given resolution it returns a transformation vector

Parameters

- **resx** (*float or int*) – Resolution in x direction
- **resy** (*float or int*) – Resolution in y direction

Returns A tuple with 6 numbers representing transformation vector

Return type tuple(float)

class sentinelhub.geometry.**Geometry**(*geometry*, *crs*)

Bases: *sentinelhub.geometry.BaseGeometry*

A class that combines shapely geometry with coordinate reference system. It currently supports polygons and multipolygons.

It can be initialize with any of the following geometry representations: - *shapely.geometry.Polygon* or *shapely.geometry.MultiPolygon* - A GeoJSON dictionary with (multi)polygon coordinates - A WKT string with (multi)polygon coordinates

Parameters

- **geometry** (*shapely.geometry.Polygon or shapely.geometry.MultiPolygon or dict or str*) – A polygon or multipolygon in any valid representation
- **crs** (constants.CRS) – Coordinate reference system of the geometry

reverse()

Returns a new Geometry object where x and y coordinates are switched

Returns New Geometry object with switched coordinates

Return type *Geometry*

transform(*crs*)

Transforms Geometry from current CRS to target CRS

Parameters **crs** (constants.CRS) – target CRS

Returns Geometry in target CRS

Return type *Geometry*

geometry

Returns shapely object representing geometry in this class

Returns A polygon or a multipolygon in shapely format

Return type shapely.geometry.Polygon or shapely.geometry.MultiPolygon

bbox
Returns BBox object representing bounding box around the geometry

Returns A bounding box, with same CRS

Return type *BBox*

class sentinelhub.geometry.**BBoxCollection**(*bbox_list*)
Bases: *sentinelhub.geometry.BaseGeometry*

A collection of bounding boxes

Parameters **bbox_list** (*list(BBox)*) – A list of BBox objects which have to be in the same CRS

bbox_list
Returns the list of bounding boxes from collection

Returns The list of bounding boxes

Return type list(*BBox*)

geometry
Returns shapely object representing geometry

Returns A multipolygon of bounding boxes

Return type shapely.geometry.MultiPolygon

bbox
Returns BBox object representing bounding box around the geometry

Returns A bounding box, with same CRS

Return type *BBox*

reverse()
Returns a new BBoxCollection object where all x and y coordinates are switched

Returns New Geometry object with switched coordinates

Return type *BBoxCollection*

transform(*crs*)
Transforms BBoxCollection from current CRS to target CRS

Parameters **crs** (constants.CRS) – target CRS

Returns BBoxCollection in target CRS

Return type *BBoxCollection*

4.1.15 geopedia

Module for working with Geopedia

class sentinelhub.geopedia.**GeopediaService**(*config=None*)
Bases: object

The class for Geopedia OGC services

Parameters **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

class sentinelhub.geopedia.**GeopediaSession**(*, *username=None*, *password=None*, *password_md5=None*, *is_global=False*, ***kwargs*)

Bases: *sentinelhub.geopedia.GeopediaService*

For retrieving data from Geopedia vector and raster layers it is required to make a session. This class handles starting and renewing of session and login (optional). It provides session headers required by Geopedia REST requests. Session duration is hardcoded to 1 hour with class attribute SESSION_DURATION. After that this class will automatically renew the session and login.

Parameters

- **username** (*str or None*) – Optional parameter in case of login with Geopedia credentials
- **password** (*str or None*) – Optional parameter in case of login with Geopedia credentials
- **password_md5** (*str or None*) – Password can optionally also be provided as already encoded md5 hexadecimal string
- **is_global** (*bool*) – If *True* this session will be shared among all instances of this class, otherwise it will be used only with the single instance. Default is *False*.
- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

session_info

All information that Geopedia provides about the current session

Returns A dictionary with session info

Return type dict

session_id

A public property of this class which provides a Geopedia session ID

Returns A session ID string

Return type str

session_headers

Headers which have to be used when accessing any data from Geopedia with this session

Returns A dictionary containing session headers

Return type dict

user_info

Information that this session has about user

Returns A dictionary with user info

Return type dict

user_id

Geopedia user ID. If no login was done during session this will return *'NO_USER'*.

Returns User ID string

Return type str

restart()
Method that restarts Geopedia Session

Returns It returns the object itself, with new session

Return type *GeopediaSession*

provide_session(*start_new=False*)
Makes sure that session is still valid and provides session info

Parameters **start_new** (*bool*) – If *True* it will always create a new session. Otherwise it will create a new session only if no session exists or the previous session timed out.

Returns Current session info

Return type dict

class sentinelhub.geopedia.**GeopediaWmsService**(***kwargs*)
Bases: *sentinelhub.geopedia.GeopediaService*, *sentinelhub.ogc.OgcImageService*

Geopedia OGC services class for providing image data. Most of the methods are inherited from *sentinelhub.ogc.OgcImageService* class.

Parameters **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

get_request(*request*)
Get a list of DownloadRequests for all data that are under the given field in the table of a Geopedia layer.

Returns list of items which have to be downloaded

Return type list(*DownloadRequest*)

get_dates(*request*)
Geopedia does not support date queries

Parameters **request** (*WmsRequest or WcsRequest*) – OGC-type request

Returns Undefined date

Return type [None]

get_wfs_iterator()
This method is inherited from OgcImageService but is not implemented.

class sentinelhub.geopedia.**GeopediaImageService**(***kwargs*)
Bases: *sentinelhub.geopedia.GeopediaService*

Service class that provides images from a Geopedia vector layer.

Parameters **base_url** (*str or None*) – Base url of Geopedia REST services. If *None*, the url specified in the configuration file is taken.

get_request(*request*)
Get a list of DownloadRequests for all data that are under the given field in the table of a Geopedia layer.

Returns list of items which have to be downloaded

Return type list(*DownloadRequest*)

get_gpd_iterator()
Returns iterator over info about data used for the *GeopediaVectorRequest*

Returns Iterator of dictionaries containing info about data used in the request.

Return type Iterator[dict] or None

class sentinelhub.geopedia.**GeopediaFeatureIterator**(*layer*, *bbox=None*, *query_filter=None*, *gpd_session=None*, ***kwargs*)

Bases: *sentinelhub.geopedia.GeopediaService*

Iterator for Geopedia Vector Service

Parameters

- **layer** (*str*) – Geopedia layer which contains requested data
- **bbox** (BBox) – Bounding box of the requested image. Its coordinates must be in the CRS.POP_WEB (EPSG:3857) coordinate system.
- **query_filter** (*str*) – Query string used for filtering returned features.
- **gpd_session** (GeopediaSession *or None*) – Optional parameter for specifying a custom Geopedia session, which can also contain login credentials. This can be used for accessing private Geopedia layers. By default it is set to *None* and a basic Geopedia session without credentials will be created.
- **config** (SHConfig *or None*) – A custom instance of config class to override parameters from the saved configuration.

get_geometry_iterator()
Iterator over Geopedia feature geometries

get_field_iterator(*field*)
Iterator over the specified field of Geopedia features

get_size()
Provides number of features which can be obtained. It has to fetch at least one feature from Geopedia services to get this information.

Returns Size of Geopedia layer with applied filters

Return type int

4.1.16 io_utils

Utility functions to read/write image data from/to file

sentinelhub.io_utils.**read_data**(*filename*, *data_format=None*)
Read image data from file

This function reads input data from file. The format of the file can be specified in `data_format`. If not specified, the format is guessed from the extension of the filename.

Parameters

- **filename** (*str*) – filename to read data from
- **data_format** (MimeType) – format of filename. Default is *None*

Returns data read from filename

Raises exception if filename does not exist

sentinelhub.io_utils.**read_tar**(*filename*)
Read a tar from file

sentinelhub.io_utils.**read_tiff_image**(*filename*)
Read data from TIFF file

Parameters **filename** (*str*) – name of TIFF file to be read

Returns data stored in TIFF file

sentinelhub.io_utils.**read_jp2_image**(*filename*)
Read data from JPEG2000 file

Parameters **filename** (*str*) – name of JPEG2000 file to be read

Returns data stored in JPEG2000 file

sentinelhub.io_utils.**read_image**(*filename*)
Read data from PNG or JPG file

Parameters **filename** (*str*) – name of PNG or JPG file to be read

Returns data stored in JPG file

sentinelhub.io_utils.**read_text**(*filename*)
Read data from text file

Parameters **filename** (*str*) – name of text file to be read

Returns data stored in text file

sentinelhub.io_utils.**read_csv**(*filename*, *delimiter=';'*)
Read data from CSV file

Parameters

- **filename** (*str*) – name of CSV file to be read
- **delimiter** (*str*) – type of CSV delimiter. Default is **;**

Returns data stored in CSV file as list

sentinelhub.io_utils.**read_json**(*filename*)
Read data from JSON file

Parameters **filename** (*str*) – name of JSON file to be read

Returns data stored in JSON file

sentinelhub.io_utils.**read_xml**(*filename*)
Read data from XML or GML file

Parameters **filename** (*str*) – name of XML or GML file to be read

Returns data stored in XML file

sentinelhub.io_utils.**read_numpy**(*filename*)
Read data from numpy file

Parameters **filename** (*str*) – name of numpy file to be read

Returns data stored in file as numpy array

sentinelhub.io_utils.**write_data**(*filename*, *data*, *data_format=None*, *compress=False*, *add=False*)
Write image data to file

Function to write image data to specified file. If file format is not provided explicitly, it is guessed from the filename extension. If format is TIFF, geo information and compression can be optionally added.

Parameters

- **filename** (*str*) – name of file to write data to
- **data** (*numpy array*) – image data to write to file

- **data_format** (MimeType) – format of output file. Default is *None*
- **compress** (*bool*) – whether to compress data or not. Default is *False*
- **add** (*bool*) – whether to append to existing text file or not. Default is *False*

Raises exception if numpy format is not supported or file cannot be written

sentinelhub.io_utils.**write_tiff_image**(*filename*, *image*, *compress=False*)
Write image data to TIFF file

Parameters

- **filename** (*str*) – name of file to write data to
- **image** (*numpy array*) – image data to write to file
- **compress** (*bool*) – whether to compress data. If *True*, lzma compression is used. Default is *False*

sentinelhub.io_utils.**write_jp2_image**(*filename*, *image*)
Write image data to JPEG2000 file

Parameters

- **filename** (*str*) – name of JPEG2000 file to write data to
- **image** (*numpy array*) – image data to write to file

Returns jp2k object

sentinelhub.io_utils.**write_image**(*filename*, *image*)
Write image data to PNG, JPG file

Parameters

- **filename** (*str*) – name of PNG or JPG file to write data to
- **image** (*numpy array*) – image data to write to file

sentinelhub.io_utils.**write_text**(*filename*, *data*, *add=False*)
Write image data to text file

Parameters

- **filename** (*str*) – name of text file to write data to
- **data** (*numpy array*) – image data to write to text file
- **add** (*bool*) – whether to append to existing file or not. Default is *False*

sentinelhub.io_utils.**write_csv**(*filename*, *data*, *delimiter=';'*)
Write image data to CSV file

Parameters

- **filename** (*str*) – name of CSV file to write data to
- **data** (*numpy array*) – image data to write to CSV file
- **delimiter** (*str*) – delimiter used in CSV file. Default is **;**

sentinelhub.io_utils.**write_json**(*filename*, *data*)
Write data to JSON file

Parameters

- **filename** (*str*) – name of JSON file to write data to

- **data** (*list, tuple*) – data to write to JSON file

sentinelhub.io_utils.**write_xml**(*filename*, *element_tree*)
Write data to XML or GML file

Parameters

- **filename** (*str*) – name of XML or GML file to write data to
- **element_tree** (*xmlElementTree*) – data as ElementTree object

sentinelhub.io_utils.**write_numpy**(*filename*, *data*)
Write data as numpy file

Parameters

- **filename** (*str*) – name of numpy file to write data to
- **data** (*numpy array*) – data to write to numpy file

sentinelhub.io_utils.**write_bytes**(*filename*, *data*)
Write binary data into a file

Parameters

- **filename** –
- **data** –

Returns

4.1.17 ogc

Module for working with Sentinel Hub OGC services

class sentinelhub.ogc.**OgcService**(*config=None*)
Bases: object

The base class for Sentinel Hub OGC services

Parameters **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

class sentinelhub.ogc.**OgcImageService**(***kwargs*)
Bases: *sentinelhub.ogc.OgcService*

Sentinel Hub OGC services class for providing image data

Intermediate layer between QGC-type requests (WmsRequest and WcsRequest) and the Sentinel Hub OGC (WMS and WCS) services.

Parameters **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

get_request(*request*)
Get download requests

Create a list of DownloadRequests for all Sentinel-2 acquisitions within request's time interval and acceptable cloud coverage.

Parameters **request** (*OgcRequest or GeopediaRequest*) – QGC-type request with specified bounding box, time interval, and cloud coverage for specific product.

Returns list of DownloadRequests

get_url(*request*, *, *date=None*, *size_x=None*, *size_y=None*, *geometry=None*)
Returns url to Sentinel Hub's OGC service for the product specified by the OgcRequest and date.

Parameters

- **request** (*OgcRequest or GeopediaRequest*) – OGC-type request with specified bounding box, cloud coverage for specific product.
- **date** (*datetime.datetime or None*) – acquisition date or None
- **size_x** (*int or str*) – horizontal image dimension
- **size_y** (*int or str*) – vertical image dimension

Returns dictionary with parameters

Returns url to Sentinel Hub's OGC service for this product.

Return type str

get_base_url(*request*)
Creates base url string.

Parameters **request** (*OgcRequest or GeopediaRequest*) – OGC-type request with specified bounding box, cloud coverage for specific product.

Returns base string for url to Sentinel Hub's OGC service for this product.

Return type str

get_dates(*request*)
Get available Sentinel-2 acquisitions at least time_difference apart

List of all available Sentinel-2 acquisitions for given bbox with max cloud coverage and the specified time interval. When a single time is specified the request will return that specific date, if it exists. If a time range is specified the result is a list of all scenes between the specified dates conforming to the cloud coverage criteria. Most recent acquisition being first in the list.

When a time_difference threshold is set to a positive value, the function filters out all datetimes which are within the time difference. The oldest datetime is preserved, all others all deleted.

Parameters **request** (*WmsRequest or WcsRequest*) – OGC-type request

Returns List of dates of existing acquisitions for the given request

Return type list(datetime.datetime) or [None]

static get_image_dimensions(*request*)
Verifies or calculates image dimensions.

Parameters **request** (*WmsRequest or WcsRequest*) – OGC-type request

Returns horizontal and vertical dimensions of requested image

Return type (int or str, int or str)

get_wfs_iterator()
Returns iterator over info about all satellite tiles used for the request

Returns Iterator of dictionaries containing info about all satellite tiles used in the request. In case of DataCollection.DEM it returns None.

Return type Iterator[dict] or None

class sentinelhub.ogc.**WebFeatureService**(*bbox*, *time_interval*, *, *data_collection=None*, *maxcc=1.0*, *data_source=None*, ***kwargs*)
Bases: *sentinelhub.ogc.OgcService*

Class for interaction with Sentinel Hub WFS service

The class is an iterator over info data of all available satellite tiles for requested parameters. It collects data from Sentinel Hub service only during the first iteration. During next iterations it returns already obtained data. The data is in the order returned by Sentinel Hub WFS service.

Parameters

- **bbox** (geometry.BBox) – Bounding box of the requested image. Coordinates must be in the specified coordinate reference system.
- **time_interval** (*(str, str)*) – interval with start and end date of the form YYYY-MM-DDThh:mm:ss or YYYY-MM-DD
- **data_collection** (DataCollection) – A collection of requested satellite data
- **maxcc** (*float*) – Maximum accepted cloud coverage of an image. Float between 0.0 and 1.0. Default is 1.0.
- **config** (SHConfig *or* *None*) – A custom instance of config class to override parameters from the saved configuration.
- **data_source** (DataCollection) – A deprecated alternative to data_collection

get_dates()
Returns a list of acquisition times from tile info data

Returns List of acquisition times in the order returned by WFS service.

Return type list(datetime.datetime)

get_geometries()
Returns a list of geometries from tile info data

Returns List of multipolygon geometries in the order returned by WFS service.

Return type list(shapely.geometry.MultiPolygon)

get_tiles()
Returns list of tiles with tile name, date and AWS index

Returns List of tiles in form of (tile_name, date, aws_index)

Return type list((str, str, int))

4.1.18 opensearch

Module for communication with http://opensearch.sentinel-hub.com/resto/api

For more search parameters check: http://opensearch.sentinel-hub.com/resto/api/collections/Sentinel2/describe.xml

exception sentinelhub.opensearch.**TileMissingException**
Bases: Exception

This exception is raised when requested tile is missing at Sentinel Hub Opensearch service

sentinelhub.opensearch.**get_tile_info_id**(*tile_id*)
Get basic information about image tile

Parameters **tile_id** (*str*) – original tile identification string provided by ESA (e.g. 'S2A_OPER_MSI_L1C_TL_SGS__20160109T230542_A002870_T10UEV_N02.01')

Returns dictionary with info provided by Opensearch REST service or *None* if such tile does not exist on AWS.

Return type dict or None

Raises TileMissingException if no tile with tile ID *tile_id* exists

sentinelhub.opensearch.**get_tile_info**(*tile*, *time*, *aws_index=None*, *all_tiles=False*)
Get basic information about image tile

Parameters

- **tile** (*str*) – tile name (e.g. `'T10UEV'`)
- **time** (*str or (str, str)*) – A single date or a time interval, times have to be in ISO 8601 string
- **aws_index** (*int or None*) – index of tile on AWS
- **all_tiles** (*bool*) – If *True* it will return list of all tiles otherwise only the first one

Returns dictionary with info provided by Opensearch REST service or *None* if such tile does not exist on AWS.

Return type dict or None

sentinelhub.opensearch.**get_area_info**(*bbox*, *date_interval*, *maxcc=None*)
Get information about all images from specified area and time range

Parameters

- **bbox** (geometry.BBox) – bounding box of requested area
- **date_interval** (*tuple(str)*) – a pair of time strings in ISO8601 format
- **maxcc** (*float in range [0, 1] or None*) – filter images by maximum percentage of cloud coverage

Returns list of dictionaries containing info provided by Opensearch REST service

Return type list(dict)

sentinelhub.opensearch.**get_area_dates**(*bbox*, *date_interval*, *maxcc=None*)
Get list of times of existing images from specified area and time range

Parameters

- **bbox** (geometry.BBox) – bounding box of requested area
- **date_interval** (*tuple(str)*) – a pair of time strings in ISO8601 format
- **maxcc** (*float in range [0, 1] or None*) – filter images by maximum percentage of cloud coverage

Returns list of time strings in ISO8601 format

Return type list[datetime.datetime]

sentinelhub.opensearch.**reduce_by_maxcc**(*result_list*, *maxcc*)
Filter list image tiles by maximum cloud coverage

Parameters

- **result_list** (*list(dict)*) – list of dictionaries containing info provided by Opensearch REST service
- **maxcc** (*float in range [0, 1]*) – filter images by maximum percentage of cloud coverage

Returns list of dictionaries containing info provided by Opensearch REST service

Return type list(dict)

sentinelhub.opensearch.**search_iter**(*tile_id=None*, *bbox=None*, *start_date=None*, *end_date=None*, *absolute_orbit=None*, *config=None*)

A generator function that implements OpenSearch search queries and returns results

All parameters for search are optional.

Parameters

- **tile_id** (*str*) – original tile identification string provided by ESA (e.g. 'S2A_OPER_MSI_L1C_TL_SGS__20160109T230542_A002870_T10UEV_N02.01')
- **bbox** (geometry.BBox) – bounding box of requested area
- **start_date** (*str*) – beginning of time range in ISO8601 format
- **end_date** (*str*) – end of time range in ISO8601 format
- **absolute_orbit** (*int*) – An absolute orbit number of Sentinel-2 L1C products as defined by ESA
- **config** (SHConfig *or None*) – A custom instance of config class to override parameters from the saved configuration.

Returns An iterator returning dictionaries with info provided by Sentinel Hub OpenSearch REST service

Return type Iterator[dict]

4.1.19 os_utils

Module for managing files and folders

sentinelhub.os_utils.**get_content_list**(*folder='.'*)

Get list of contents in input folder

Parameters **folder** (*str*) – input folder to list contents. Default is `'.'`

Returns list of folder contents

Return type list(str)

sentinelhub.os_utils.**get_folder_list**(*folder='.'*)

Get list of sub-folders contained in input folder

Parameters **folder** (*str*) – input folder to list sub-folders. Default is `'.'`

Returns list of sub-folders

Return type list(str)

sentinelhub.os_utils.**get_file_list**(*folder='.'*)

Get list of files contained in input folder

Parameters **folder** (*str*) – input folder to list files only. Default is `'.'`

Returns list of files

Return type list(str)

sentinelhub.os_utils.**create_parent_folder**(*filename*)

Create parent folder for input filename recursively

Parameters **filename** (*str*) – input filename

Raises error if folder cannot be created

sentinelhub.os_utils.**make_folder**(*path*)
Create folder at input path recursively

Create a folder specified by input path if one does not exist already

Parameters **path** (*str*) – input path to folder to be created

Raises os.error if folder cannot be created

sentinelhub.os_utils.**rename**(*old_path*, *new_path*, *edit_folders=True*)
Rename files or folders

Parameters

- **old_path** (*str*) – name of file or folder to rename
- **new_path** (*str*) – name of new file or folder
- **edit_folders** (*bool*) – flag to allow recursive renaming of folders. Default is *True*

sentinelhub.os_utils.**size**(*pathname*)
Returns size of a file or folder in Bytes

Parameters **pathname** (*str*) – path to file or folder to be sized

Returns size of file or folder in Bytes

Return type int

Raises os.error if file is not accessible

sentinelhub.os_utils.**sys_is_windows**()
Check if user is running the code on Windows machine

Returns *True* if OS is Windows and *False* otherwise

Return type bool

4.1.20 sentinelhub_batch

Module implementing an interface with Sentinel Hub Batch service

class sentinelhub.sentinelhub_batch.**SentinelHubBatch**(*request_id=None*, ***, *request_info=None*, *config=None*)
Bases: object

An interface class for Sentinel Hub Batch API

For more info check Batch API reference.

Parameters

- **request_id** (*str or None*) – A batch request ID
- **request_info** (*dict or None*) – Information about batch request parameters obtained from the service. This parameter can be given instead of *request_id*
- **config** (*SHConfig or None*) – A configuration object

classmethod create(*sentinelhub_request*, *tiling_grid*, *output=None*, *bucket_name=None*, *description=None*, *config=None*)
Create a new batch request

Parameters

- **sentinelhub_request** (*SentinelHubRequest or dict*) – An instance of SentinelHubRequest class containing all request parameters. Alternatively, it can also be just a payload dictionary for Processing API request
- **tiling_grid** (*dict*) – A dictionary with tiling grid parameters. It can be built with *tiling_grid* method
- **output** (*dict or None*) – A dictionary with output parameters. It can be built with *output* method. Alternatively, one can set *bucket_name* parameter instead.
- **bucket_name** (*str or None*) – A name of an s3 bucket where to save data. Alternatively, one can set *output* parameter to specify more output parameters.
- **description** (*str or None*) – A description of a batch request
- **config** (*SHConfig or None*) – A configuration object

static tiling_grid(*grid_id*, *resolution*, *buffer=None*, ***kwargs*)

A helper method to build a dictionary with tiling grid parameters

Parameters

- **grid_id** (*int*) – An ID of a tiling grid
- **resolution** (*float or int*) – A grid resolution
- **buffer** (*(int, int) or None*) – Optionally, a buffer around each tile can be defined. It can be defined with a tuple of integers *(buffer_x, buffer_y)*, which specifies a number of buffer pixels in horizontal and vertical directions.
- **kwargs** – Any other arguments to be added to a dictionary of parameters

Returns A dictionary with parameters

Return type dict

static output(*, *default_tile_path=None*, *cog_output=None*, *cog_parameters=None*, *create_collection=None*, *collection_id=None*, *responses=None*, ***kwargs*)

A helper method to build a dictionary with tiling grid parameters

Parameters

- **default_tile_path** (*str or None*) – A path or a template on an s3 bucket where to store results. More info at Batch API documentation
- **cog_output** (*bool or None*) – A flag specifying if outputs should be written in COGs (cloud-optimized GeoTIFFs)or normal GeoTIFFs
- **cog_parameters** (*dict or None*) – A dictionary specifying COG creation parameters
- **create_collection** (*bool or None*) – If True the results will be written in COGs and a batch collection will be created
- **collection_id** (*str or None*) – If True results will be added to an existing collection
- **responses** (*list or None*) – Specification of path template for individual outputs/responses
- **kwargs** – Any other arguments to be added to a dictionary of parameters

Returns A dictionary of output parameters

Return type dict

static iter_tiling_grids(*search=None*, *sort=None*, *config=None*, ***kwargs*)
An iterator over tiling grids

Parameters

- **search** (*str*) – A search parameter
- **sort** (*str*) – A sort parameter
- **config** (SHConfig) – A configuration object
- **kwargs** – Any other request query parameters

Returns An iterator over tiling grid definitions

Return type Iterator[dict]

static get_tiling_grid(*grid_id*, *config=None*)
Provides a single tiling grid

Parameters

- **grid_id** (*str or int*) – An ID of a requested tiling grid
- **config** (SHConfig) – A configuration object

Returns A tiling grid definition

Return type dict

info
A dictionary with a Batch request information. It loads a new dictionary only if one doesn't exist yet.

Returns Batch request info

Return type dict

update_info()
Updates information about a batch request

Returns Batch request info

Return type dict

evalscript
Provides an evalscript used by a batch request

Returns An evalscript

Return type str

bbox
Provides a bounding box used by a batch request

Returns An area bounding box together with CRS

Return type *BBox*

Raises ValueError

geometry
Provides a geometry used by a batch request

Returns An area geometry together with CRS

Return type *Geometry*

Raises ValueError

static iter_requests(*search=None*, *sort=None*, *user_id=None*, *config=None*, ***kwargs*)
Iterate existing batch requests

Parameters

- **search** (*str or None*) – Filter requests by a search query
- **sort** (*str or None*) – Sort obtained batch requests in a specific order
- **user_id** (*str or None*) – Filter requests by a user id who defined a request
- **config** (*SHConfig or None*) – A configuration object
- **kwargs** – Any additional parameters to include in a request query

Returns An iterator over existing batch requests

Return type Iterator[*SentinelHubBatch*]

static get_latest_request(*config=None*)
Provides a batch request that has been created the latest

delete()
Delete a batch job request

start_analysis()
Starts analysis of a batch job request

start_job()
Starts running a batch job

cancel_job()
Cancels a batch job

restart_job()
Restarts only those parts of a job that failed

iter_tiles(*status=None*, ***kwargs*)
Iterate over info about batch request tiles

Parameters

- **status** (*str or None*) – A filter to obtain only tiles with a certain status
- **kwargs** – Any additional parameters to include in a request query

Returns An iterator over information about each tile

Return type Iterator[dict]

get_tile(*tile_id*)
Provides information about a single batch request tile

Parameters **tile_id** (*int or None*) – An ID of a tile

Returns Information about a tile

Return type dict

reprocess_tile(*tile_id*)
Reprocess a single failed tile

Parameters **tile_id** (*int or None*) – An ID of a tile

4.1.21 sentinelhub_rate_limit

Module implementing rate limiting logic for Sentinel Hub service

class sentinelhub.sentinelhub_rate_limit.**SentinelHubRateLimit**(*num_processes=1*, *minimum_wait_time=0.05*, *maximum_wait_time=60.0*)

Bases: object

Class implementing rate limiting logic of Sentinel Hub service

It has 2 public methods:

- register_next - tells if next download can start or if not, what is the wait before it can be asked again
- update - updates expectations according to headers obtained from download

The rate limiting object is collecting information about the status of rate limiting policy buckets from Sentinel Hub service. According to this information and a feedback from download requests it adapts expectations about when the next download attempt will be possible.

Parameters

- **num_processes** (*int*) – Number of parallel download processes running.
- **minimum_wait_time** (*float*) – Minimum wait time between two consecutive download requests in seconds.
- **maximum_wait_time** (*float*) – Maximum wait time between two consecutive download requests in seconds.

register_next()

Determines if next download request can start or not by returning the waiting time in seconds.

update(*headers*)

Update the next possible download time if the service has responded with the rate limit

class sentinelhub.sentinelhub_rate_limit.**PolicyBucket**(*policy_type*, *policy_payload*)

Bases: object

A class representing Sentinel Hub policy bucket

Parameters

- **policy_type** (PolicyType *or* *str*) – A type of policy
- **policy_payload** (*dict*) – A dictionary of policy parameters

content

Variable *content* can be accessed as a property

count_cost_per_second(*elapsed_time*, *new_content*)

Calculates the cost per second for the bucket given the elapsed time and the new content.

In the calculation it assumes that during the elapsed time bucket was being filled all the time - i.e. it assumes the bucket has never been full for a non-zero amount of time in the elapsed time period.

get_wait_time(*elapsed_time*, *process_num*, *cost_per_request*, *requests_completed*, *buffer_cost=0.5*)

Expected time a user would have to wait for this bucket

is_request_bucket()

Checks if bucket counts requests

is_fixed()
Checks if bucket has a fixed number of requests

class sentinelhub.sentinelhub_rate_limit.**PolicyType**
Bases: enum.Enum

Enum defining different types of policies

4.1.22 sentinelhub_request

SentinelHubRequest for the Processing API

Documentation: https://docs.sentinel-hub.com/api/latest/reference/

class sentinelhub.sentinelhub_request.**SentinelHubRequest**(*evalscript*, *input_data*, *responses*, *bbox=None*, *geometry=None*, *size=None*, *resolution=None*, ***kwargs*)
Bases: *sentinelhub.data_request.DataRequest*

Sentinel Hub API request class

For details of certain parameters check the Processing API reference.

Parameters

- **evalscript** (*str*) – Evalscript.
- **input_data** (*List[dict or InputDataDict]*) – A list of input dictionary objects as described in the API reference. It can be generated with the helper function *SentinelHubRequest.input_data*
- **responses** (*List[dict]*) – A list of *output.responses* objects as described in the API reference. It can be generated with the helper function *SentinelHubRequest.output_response*
- **bbox** (*sentinelhub.BBox*) – Bounding box describing the area of interest.
- **geometry** (*sentinelhub.Geometry*) – Geometry describing the area of interest.
- **size** (*Tuple[int, int]*) – Size of the image.
- **resolution** (*Tuple[float, float]*) – Resolution of the image. It has to be in units compatible with the given CRS.
- **data_folder** (*str*) – location of the directory where the fetched data will be saved.
- **config** (*SHConfig or None*) – A custom instance of config class to override parameters from the saved configuration.

create_request()
Prepares a download request

static input_data(*data_collection=None*, *time_interval=None*, *maxcc=None*, *mosaicking_order=None*, *upsampling=None*, *downsampling=None*, *other_args=None*, *data_source=None*)
Generate the *input* part of the Processing API request body

Parameters

- **data_collection** (DataCollection) – One of supported ProcessingAPI data collections.

- **time_interval** (*(str, str) or (datetime, datetime)*) – interval with start and end date of the form YYYY-MM-DDThh:mm:ss or YYYY-MM-DD
- **maxcc** (*float or None*) – Maximum accepted cloud coverage of an image. Float between 0.0 and 1.0. Default is 1.0.
- **mosaicking_order** (*str or None*) – Mosaicking order, which has to be either 'mostRecent', 'leastRecent' or 'leastCC'.
- **upsampling** (*str*) – A type of upsampling to apply on data
- **downsampling** (*str*) – A type of downsampling to apply on data
- **other_args** – Additional dictionary of arguments. If provided, the resulting dictionary will get updated by it.
- **other_args** – dict
- **data_source** (DataCollection) – A deprecated alternative of data_collection

Returns A dictionary-like object that also contains additional attributes

Return type *InputDataDict*

static body(*request_bounds*, *request_data*, *evalscript*, *request_output=None*, *other_args=None*)

Generate the body the Processing API request body

Parameters

- **request_bounds** (*dict*) – A dictionary as generated by *SentinelHubRequest.bounds* helper method.
- **request_data** (*List[dict]*) – A list of dictionaries as generated by *SentinelHubRequest.input_data* helper method.
- **evalscript** (*str*) – Evalscript (https://docs.sentinel-hub.com/api/latest/#/Evalscript/)
- **request_output** (*dict*) – A dictionary as generated by *SentinelHubRequest.output* helper method.
- **other_args** – Additional dictionary of arguments. If provided, the resulting dictionary will get updated by it.
- **other_args** – dict

static output_response(*identifier*, *response_format*, *other_args=None*)

Generate an element of *output.responses* as described in the Processing API reference.

Parameters

- **identifier** (*str*) – Identifier of the output response.
- **response_format** (*str or sentinelhub.MimeType*) – A mime type of one of 'png', 'json', 'jpeg', 'tiff'.
- **other_args** – Additional dictionary of arguments. If provided, the resulting dictionary will get updated by it.
- **other_args** – dict

static output(*responses*, *size=None*, *resolution=None*, *other_args=None*)

Generate an *output* part of the request as described in the Processing API reference

Parameters

- **responses** (*List[dict]*) – A list of objects in *output.responses* as generated by *SentinelHubRequest.output_response*.

- **size** (*Tuple[int, int]*) – Size of the image.
- **resolution** (*Tuple[float, float]*) – Resolution of the image. It has to be in units compatible with the given CRS.
- **other_args** – Additional dictionary of arguments. If provided, the resulting dictionary will get updated by it.
- **other_args** – dict

static bounds(*bbox=None*, *geometry=None*, *other_args=None*)
Generate a *bound* part of the Processing API request

Parameters

- **bbox** (*sentinelhub.BBox*) – Bounding box describing the area of interest.
- **geometry** (*sentinelhub.Geometry*) – Geometry describing the area of interest.
- **other_args** – Additional dictionary of arguments. If provided, the resulting dictionary will get updated by it.
- **other_args** – dict

class sentinelhub.sentinelhub_request.**InputDataDict**(*input_data_dict*, ***, *service_url=None*)
Bases: dict

An input data dictionary which also holds additional attributes

Parameters

- **input_data_dict** (*dict*) – A normal dictionary with input parameters
- **service_url** (*str*) – A service URL defined by a data collection

4.1.23 sentinelhub_session

Module implementing Sentinel Hub session object

class sentinelhub.sentinelhub_session.**SentinelHubSession**(*config=None*)
Bases: object

Sentinel Hub authentication class

The class will do OAuth2 authentication with Sentinel Hub service and store the token. It will make sure that the token is never expired by automatically refreshing it if expiry time is close.

Parameters **config** (SHConfig) – An instance of package configuration class

token
Always up-to-date session's token

Returns A token in a form of dictionary of parameters

Return type dict

session_headers
Provides session authorization headers

Returns A dictionary with authorization headers

Return type dict

4.1.24 testing_utils

Utility tools for writing unit tests for packages which rely on *sentinelhub-py*

class sentinelhub.testing_utils.**TestSentinelHub**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Class implementing common functionalities of unit tests for working with *sentinelhub-py* package:

- reading configuration parameters from environmental variables and saving them to *config.json*,
- setting logger,
- handling input and output data folders,
- method for testing statistics of a numpy data array.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

classmethod setUpClass()
A general set up class

Use `super().setUpClass()` in every class which inherits `TestSentinelHub`

classmethod tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_numpy_data(*data=None*, *exp_shape=None*, *exp_dtype=None*, *exp_min=None*, *exp_max=None*, *exp_mean=None*, *exp_median=None*, *delta=None*, *test_name=''*)
Validates basic statistics of data array

Parameters

- **data** (*numpy.ndarray*) – Data array
- **exp_shape** (*tuple(int)*) – Expected shape
- **exp_dtype** (*numpy.dtype*) – Expected dtype
- **exp_min** (*float*) – Expected minimal value
- **exp_max** (*float*) – Expected maximal value
- **exp_mean** (*float*) – Expected mean value
- **exp_median** (*float*) – Expected median value
- **delta** (*float*) – Precision of validation. If not set, it will be set automatically
- **test_name** (*str*) – Name of the test case

class sentinelhub.testing_utils.**TestCaseContainer**(*name*, *request*, ***stats*)
Bases: object

Class for storing expected statistics for a single test case

Parameters

- **name** (*str*) – Name of a test case
- **request** (*object*) – A class which provides the data for testing
- **stats** – Any other parameters

4.1.25 time_utils

Module with useful time/date functions

sentinelhub.time_utils.**get_dates_in_range**(*start_date*, *end_date*)
Get all dates within input start and end date in ISO 8601 format

Parameters

- **start_date** (*str*) – start date in ISO 8601 format
- **end_date** (*str*) – end date in ISO 8601 format

Returns list of dates between start_date and end_date in ISO 8601 format

Return type list of str

sentinelhub.time_utils.**next_date**(*date*)
Get date of day after input date in ISO 8601 format

For instance, if input date is `'2017-03-12'`, the function returns `'2017-03-13'`

Parameters **date** (*str*) – input date in ISO 8601 format

Returns date after input date in ISO 8601 format

Return type str

sentinelhub.time_utils.**prev_date**(*date*)
Get date of day previous to input date in ISO 8601 format

For instance, if input date is `'2017-03-12'`, the function returns `'2017-03-11'`

Parameters **date** (*str*) – input date in ISO 8601 format

Returns date previous to input date in ISO 8601 format

Return type str

sentinelhub.time_utils.**iso_to_datetime**(*date*)
Convert ISO 8601 time format to datetime format

This function converts a date in ISO format, e.g. `2017-09-14` to a *datetime* instance, e.g. `datetime.datetime(2017,9,14,0,0)`

Parameters **date** (*str*) – date in ISO 8601 format

Returns datetime instance

Return type datetime

sentinelhub.time_utils.**datetime_to_iso**(*date*, *only_date=True*)
Convert datetime format to ISO 8601 time format

This function converts a date in datetime instance, e.g. `datetime.datetime(2017,9,14,0,0)` to ISO format, e.g. `2017-09-14`

Parameters

- **date** (*datetime*) – datetime instance to convert
- **only_date** (*bool*) – whether to return date only or also time information. Default is *True*

Returns date in ISO 8601 format

Return type str

sentinelhub.time_utils.**get_current_date**()
Get current date in ISO 8601 format

Returns current date in ISO 8601 format

Return type str

sentinelhub.time_utils.**is_valid_time**(*time*)
Check if input string represents a valid time/date stamp

Parameters **time** (*str*) – a string containing a time/date stamp

Returns *True* is string is a valid time/date stamp, *False* otherwise

Return type bool

sentinelhub.time_utils.**parse_time**(*time_input*)
Parse input time/date string into ISO 8601 string

Parameters **time_input** (*str or datetime.date or datetime.datetime*) – time/date to parse

Returns parsed string in ISO 8601 format

Return type str

sentinelhub.time_utils.**parse_time_interval**(*time*)
Parse input into an interval of two times, specifying start and end time, in ISO 8601 format, for example:

```
(2017-01-15:T00:00:00, 2017-01-16:T23:59:59)
```

The input time can have the following formats, which will be parsed as:

- *YYYY-MM-DD -> [YYYY-MM-DD:T00:00:00, YYYY-MM-DD:T23:59:59]*
- *YYYY-MM-DDThh:mm:ss -> [YYYY-MM-DDThh:mm:ss, YYYY-MM-DDThh:mm:ss]*
- list or tuple of two dates in form *YYYY-MM-DD -> [YYYY-MM-DDT00:00:00, YYYY-MM-DDT23:59:59]*
- list or tuple of two dates in form *YYYY-MM-DDThh:mm:ss -> [YYYY-MM-DDThh:mm:ss, YYYY-MM-DDThh:mm:ss]*

All input times can also be specified as *datetime* objects. Instances of *datetime.date* will be treated as *YYYY-MM-DD* and instance of *datetime.datetime* will be treated as *YYYY-MM-DDThh:mm:ss*.

Parameters **time** (*str or datetime.datetime or (str, str) or (datetime.datetime, datetime.datetime)*) – An input time

Returns interval of start and end date of the form *YYYY-MM-DDThh:mm:ss*

Return type (str, str)

Raises ValueError

sentinelhub.time_utils.**filter_times**(*timestamps*, *time_difference*)
Filters out dates within time_difference, preserving only the oldest date.

Parameters

- **timestamps** (*list(datetime.datetime)*) – A list of timestamp objects
- **time_difference** (*datetime.timedelta*) – A time difference threshold

Returns An ordered list of timestamps *d_1<=d_2<=...<=d_n* such that *d_(i+1)-d_i > time_difference*

Return type list(datetime.datetime)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

s

sentinelhub.areas, 111
sentinelhub.aws, 113
sentinelhub.aws_safe, 118
sentinelhub.config, 121
sentinelhub.constants, 123
sentinelhub.data_collections, 126
sentinelhub.data_request, 128
sentinelhub.download.aws_client, 140
sentinelhub.download.client, 140
sentinelhub.download.request, 142
sentinelhub.download.sentinelhub_client, 144
sentinelhub.fis, 144
sentinelhub.geo_utils, 144
sentinelhub.geometry, 147
sentinelhub.geopedia, 151
sentinelhub.io_utils, 154
sentinelhub.ogc, 157
sentinelhub.opensearch, 159
sentinelhub.os_utils, 161
sentinelhub.sentinelhub_batch, 162
sentinelhub.sentinelhub_rate_limit, 166
sentinelhub.sentinelhub_request, 167
sentinelhub.sentinelhub_session, 169
sentinelhub.testing_utils, 170
sentinelhub.time_utils, 171

Index

A

add_file_extension() (*sentinelhub.aws.AwsService static method*), 114
api_id (*sentinelhub.data_collections.DataCollection attribute*), 127
AreaSplitter (*in module sentinelhub.areas*), 111
AwsConstants (*class in sentinelhub.constants*), 126
AwsDownloadClient (*class in sentinelhub.download.aws_client*), 140
AwsProduct (*class in sentinelhub.aws*), 115
AwsProductRequest (*class in sentinelhub.data_request*), 138
AwsRequest (*class in sentinelhub.data_request*), 137
AwsService (*class in sentinelhub.aws*), 113
AwsTile (*class in sentinelhub.aws*), 116
AwsTileRequest (*class in sentinelhub.data_request*), 138

B

bands (*sentinelhub.data_collections.DataCollection attribute*), 128
BaseGeometry (*class in sentinelhub.geometry*), 147
BBox (*class in sentinelhub.geometry*), 148
bbox (*sentinelhub.geometry.BBoxCollection attribute*), 151
bbox (*sentinelhub.geometry.Geometry attribute*), 151
bbox (*sentinelhub.sentinelhub_batch.SentinelHubBatch attribute*), 164
bbox_list (*sentinelhub.geometry.BBoxCollection attribute*), 151
bbox_to_dimensions() (*in module sentinelhub.geo_utils*), 144
bbox_to_resolution() (*in module sentinelhub.geo_utils*), 145
BBoxCollection (*class in sentinelhub.geometry*), 151
BBoxSplitter (*class in sentinelhub.areas*), 111
body() (*sentinelhub.sentinelhub_request.SentinelHubRequest static method*), 168
bounds() (*sentinelhub.sentinelhub_request.SentinelHubRequest static method*), 169
buffer() (*sentinelhub.geometry.BBox method*), 149

C

cancel_job() (*sentinelhub.sentinelhub_batch.SentinelHubBatch method*), 165
contains_orbit_direction() (*sentinelhub.data_collections.DataCollection method*), 128
content (*sentinelhub.sentinelhub_rate_limit.PolicyBucket attribute*), 166
count_cost_per_second() (*sentinelhub.sentinelhub_rate_limit.PolicyBucket method*), 166
create() (*sentinelhub.sentinelhub_batch.SentinelHubBatch class method*), 162
create_parent_folder() (*in module sentinelhub.os_utils*), 161
create_request() (*sentinelhub.data_request.AwsProductRequest method*), 138
create_request() (*sentinelhub.data_request.AwsRequest method*), 138
create_request() (*sentinelhub.data_request.AwsTileRequest method*), 139
create_request() (*sentinelhub.data_request.DataRequest method*), 129
create_request() (*sentinelhub.data_request.FisRequest method*), 135
create_request() (*sentinelhub.data_request.GeopediaImageRequest method*), 137
create_request() (*sentinelhub.data_request.GeopediaRequest method*), 136
create_request() (*sentinelhub.data_request.GeopediaWmsRequest*

method), 136
create_request() (*sentinelhub.data_request.OgcRequest method*), 131
create_request() (*sentinelhub.sentinelhub_request.SentinelHubRequest method*), 167
CRS (*class in sentinelhub.constants*), 123
crs (*sentinelhub.geometry.BaseGeometry attribute*), 147
CRSMeta (*class in sentinelhub.constants*), 123
CustomGridSplitter (*class in sentinelhub.areas*), 113
CustomUrlParam (*class in sentinelhub.constants*), 124

D

DataCollection (*class in sentinelhub.data_collections*), 127
DataCollectionDefinition (*class in sentinelhub.data_collections*), 126
DataRequest (*class in sentinelhub.data_request*), 128
DataSource (*in module sentinelhub.data_collections*), 128
datetime_to_iso() (*in module sentinelhub.time_utils*), 171
define_from() (*sentinelhub.data_collections.DataCollection method*), 127
delete() (*sentinelhub.sentinelhub_batch.SentinelHubBatch method*), 165
derive() (*sentinelhub.data_collections.DataCollectionDefinition method*), 127
download() (*sentinelhub.download.client.DownloadClient method*), 141
download_safe_format() (*in module sentinelhub.data_request*), 139
DownloadClient (*class in sentinelhub.download.client*), 140
DownloadRequest (*class in sentinelhub.download.request*), 142

E

epsg (*sentinelhub.constants.CRS attribute*), 123
EsaSafeType (*class in sentinelhub.constants*), 126
evalscript (*sentinelhub.sentinelhub_batch.SentinelHubBatch attribute*), 164
extension (*sentinelhub.constants.MimeType attribute*), 125

F

filter_times() (*in module sentinelhub.time_utils*), 172
FisRequest (*class in sentinelhub.data_request*), 134
FisService (*class in sentinelhub.fis*), 144

G

geojson (*sentinelhub.geometry.BaseGeometry attribute*), 148
Geometry (*class in sentinelhub.geometry*), 150
geometry (*sentinelhub.geometry.BaseGeometry attribute*), 147
geometry (*sentinelhub.geometry.BBox attribute*), 149
geometry (*sentinelhub.geometry.BBoxCollection attribute*), 151
geometry (*sentinelhub.geometry.Geometry attribute*), 150
geometry (*sentinelhub.sentinelhub_batch.SentinelHubBatch attribute*), 164
GeopediaFeatureIterator (*class in sentinelhub.geopedia*), 153
GeopediaImageRequest (*class in sentinelhub.data_request*), 136
GeopediaImageService (*class in sentinelhub.geopedia*), 153
GeopediaRequest (*class in sentinelhub.data_request*), 135
GeopediaService (*class in sentinelhub.geopedia*), 151
GeopediaSession (*class in sentinelhub.geopedia*), 152
GeopediaWmsRequest (*class in sentinelhub.data_request*), 136
GeopediaWmsService (*class in sentinelhub.geopedia*), 153
get_area_dates() (*in module sentinelhub.opensearch*), 160
get_area_info() (*in module sentinelhub.opensearch*), 160
get_aux_data_name() (*sentinelhub.aws_safe.SafeTile method*), 120
get_aws_index() (*sentinelhub.aws.AwsTile method*), 117
get_aws_service() (*sentinelhub.data_request.AwsRequest method*), 138
get_base_url() (*sentinelhub.aws.AwsService method*), 113
get_base_url() (*sentinelhub.ogc.OgcImageService method*), 158
get_baseline() (*sentinelhub.aws.AwsService method*), 114
get_config_dict() (*sentinelhub.config.SHConfig method*), 122
get_config_location() (*sentinelhub.config.SHConfig method*), 122
get_content_list() (*in module sentinelhub.os_utils*), 161
get_current_date() (*in module sentinelhub.time_utils*), 171
get_data() (*sentinelhub.data_request.DataRequest*

method), 129
get_data_collection() (*sentinelhub.aws.AwsProduct method*), 115
get_datastrip_list() (*sentinelhub.aws_safe.SafeProduct method*), 119
get_datastrip_metadata_name() (*sentinelhub.aws_safe.SafeProduct method*), 119
get_datastrip_name() (*sentinelhub.aws_safe.SafeProduct method*), 119
get_datastrip_time() (*sentinelhub.aws_safe.SafeTile method*), 120
get_datatake_time() (*sentinelhub.aws_safe.SafeTile method*), 120
get_date() (*sentinelhub.aws.AwsProduct method*), 115
get_dates() (*sentinelhub.data_request.FisRequest method*), 135
get_dates() (*sentinelhub.data_request.OgcRequest method*), 131
get_dates() (*sentinelhub.geopedia.GeopediaWmsService method*), 153
get_dates() (*sentinelhub.ogc.OgcImageService method*), 158
get_dates() (*sentinelhub.ogc.WebFeatureService method*), 159
get_dates_in_range() (*in module sentinelhub.time_utils*), 171
get_download_list() (*sentinelhub.data_request.DataRequest method*), 129
get_expected_max_value() (*sentinelhub.constants.MimeType method*), 126
get_field_iterator() (*sentinelhub.geopedia.GeopediaFeatureIterator method*), 154
get_file_list() (*in module sentinelhub.os_utils*), 161
get_filename_list() (*sentinelhub.data_request.DataRequest method*), 129
get_filepath() (*sentinelhub.aws.AwsProduct method*), 116
get_filepath() (*sentinelhub.aws.AwsTile method*), 118
get_folder_list() (*in module sentinelhub.os_utils*), 161
get_geojson() (*sentinelhub.geometry.BaseGeometry method*), 148
get_geometries() (*sentinelhub.ogc.WebFeatureService method*), 159
get_geometry_iterator() (*sentinelhub.geopedia.GeopediaFeatureIterator method*), 154
get_gml_url() (*sentinelhub.aws.AwsTile method*), 118
get_gpd_iterator() (*sentinelhub.geopedia.GeopediaImageService method*), 153
get_hashed_name() (*sentinelhub.download.request.DownloadRequest method*), 143
get_image_dimension() (*in module sentinelhub.geo_utils*), 145
get_image_dimensions() (*sentinelhub.ogc.OgcImageService static method*), 158
get_img_name() (*sentinelhub.aws_safe.SafeTile method*), 120
get_items() (*sentinelhub.data_request.GeopediaImageRequest method*), 137
get_json() (*in module sentinelhub.download.client*), 142
get_json() (*sentinelhub.download.client.DownloadClient method*), 141
get_latest_request() (*sentinelhub.sentinelhub_batch.SentinelHubBatch static method*), 165
get_main_folder() (*sentinelhub.aws_safe.SafeProduct method*), 119
get_main_folder() (*sentinelhub.aws_safe.SafeTile method*), 120
get_params() (*sentinelhub.config.SHConfig method*), 122
get_partition() (*sentinelhub.geometry.BBox method*), 149
get_polygon() (*sentinelhub.geometry.BBox method*), 149
get_preview_name() (*sentinelhub.aws_safe.SafeTile method*), 121
get_preview_url() (*sentinelhub.aws.AwsTile method*), 118
get_product_id() (*sentinelhub.aws.AwsTile method*), 118
get_product_metadata_name() (*sentinelhub.aws_safe.SafeProduct method*), 119
get_product_url() (*sentinelhub.aws.AwsProduct method*), 116
get_qi_name() (*sentinelhub.aws_safe.SafeTile method*), 121
get_qi_url() (*sentinelhub.aws.AwsTile method*), 118
get_relative_paths() (*sentinelhub.download.request.DownloadRequest method*), 143
get_report_name() (*sentinelhub.aws_safe.SafeProduct method*), 119

get_report_time() (sentinelhub.aws_safe.SafeProduct method), 119
get_request() (sentinelhub.fis.FisService method), 144
get_request() (sentinelhub.geopedia.GeopediaImageService method), 153
get_request() (sentinelhub.geopedia.GeopediaWmsService method), 153
get_request() (sentinelhub.ogc.OgcImageService method), 157
get_request_params() (sentinelhub.download.request.DownloadRequest method), 143
get_requests() (sentinelhub.aws.AwsProduct method), 115
get_requests() (sentinelhub.aws.AwsService method), 113
get_requests() (sentinelhub.aws.AwsTile method), 117
get_requests() (sentinelhub.aws_safe.SafeProduct method), 119
get_requests() (sentinelhub.aws_safe.SafeTile method), 120
get_safe_format() (in module sentinelhub.data_request), 139
get_safe_struct() (sentinelhub.aws_safe.SafeProduct method), 119
get_safe_struct() (sentinelhub.aws_safe.SafeTile method), 120
get_safe_type() (sentinelhub.aws.AwsService method), 113
get_sample_type() (sentinelhub.constants.MimeType method), 125
get_sensing_time() (sentinelhub.aws_safe.SafeTile method), 120
get_sh_oauth_url() (sentinelhub.config.SHConfig method), 122
get_sh_ogc_url() (sentinelhub.config.SHConfig method), 123
get_sh_processing_api_url() (sentinelhub.config.SHConfig method), 122
get_sh_rate_limit_url() (sentinelhub.config.SHConfig method), 123
get_size() (sentinelhub.geopedia.GeopediaFeatureIterator method), 154
get_storage_paths() (sentinelhub.download.request.DownloadRequest method), 143
get_string() (sentinelhub.constants.MimeType method), 125
get_tile() (sentinelhub.sentinelhub_batch.SentinelHubBatch method), 165
get_tile_dict() (sentinelhub.areas.TileSplitter method), 112
get_tile_id() (sentinelhub.aws_safe.SafeTile method), 120
get_tile_info() (in module sentinelhub.opensearch), 160
get_tile_info() (sentinelhub.aws.AwsTile method), 117
get_tile_info_id() (in module sentinelhub.opensearch), 159
get_tile_metadata_name() (sentinelhub.aws_safe.SafeTile method), 120
get_tile_url() (sentinelhub.aws.AwsProduct method), 116
get_tile_url() (sentinelhub.aws.AwsTile method), 117
get_tiles() (sentinelhub.data_request.FisRequest method), 135
get_tiles() (sentinelhub.data_request.OgcRequest method), 131
get_tiles() (sentinelhub.ogc.WebFeatureService method), 159
get_tiling_grid() (sentinelhub.sentinelhub_batch.SentinelHubBatch static method), 164
get_transform_function (sentinelhub.constants.CRS attribute), 124
get_transform_vector() (sentinelhub.geometry.BBox method), 150
get_url() (sentinelhub.aws.AwsProduct method), 115
get_url() (sentinelhub.aws.AwsTile method), 117
get_url() (sentinelhub.ogc.OgcImageService method), 157
get_url_list() (sentinelhub.data_request.DataRequest method), 129
get_utm_bbox() (in module sentinelhub.geo_utils), 145
get_utm_crs() (in module sentinelhub.geo_utils), 147
get_version() (sentinelhub.constants.PackageProps static method), 123
get_wait_time() (sentinelhub.sentinelhub_rate_limit.PolicyBucket method), 166
get_wfs_iterator() (sentinelhub.geopedia.GeopediaWmsService method), 153
get_wfs_iterator() (sentinelhub.ogc.OgcImageService method), 158
get_world_bbox() (sentinelhub.areas.OsmSplitter method), 112

get_xml() (*in module sentinelhub.download.client*), 142
get_xml() (*sentinelhub.download.client.DownloadClient method*), 141

H

handle_deprecated_data_source() (*in module sentinelhub.data_collections*), 128
has_eocloud_url() (*sentinelhub.config.SHConfig method*), 122
has_reports() (*sentinelhub.aws.AwsService method*), 114
HistogramType (*class in sentinelhub.constants*), 125

I

info (*sentinelhub.sentinelhub_batch.SentinelHubBatch attribute*), 164
input_data() (*sentinelhub.sentinelhub_request.SentinelHubRequest static method*), 167
InputDataDict (*class in sentinelhub.sentinelhub_request*), 169
is_api_format() (*sentinelhub.constants.MimeType method*), 125
is_early_compact_l2a() (*sentinelhub.aws.AwsService method*), 115
is_fixed() (*sentinelhub.sentinelhub_rate_limit.PolicyBucket method*), 166
is_image_format() (*sentinelhub.constants.MimeType method*), 125
is_request_bucket() (*sentinelhub.sentinelhub_rate_limit.PolicyBucket method*), 166
is_s3_request() (*sentinelhub.download.aws_client.AwsDownloadClient static method*), 140
is_sentinel1 (*sentinelhub.data_collections.DataCollection attribute*), 128
is_tiff_format() (*sentinelhub.constants.MimeType method*), 125
is_utm() (*sentinelhub.constants.CRS method*), 124
is_valid_request() (*sentinelhub.data_request.DataRequest method*), 129
is_valid_time() (*in module sentinelhub.time_utils*), 172
iso_to_datetime() (*in module sentinelhub.time_utils*), 171
iter_requests() (*sentinelhub.sentinelhub_batch.SentinelHubBatch static method*), 164
iter_tiles() (*sentinelhub.sentinelhub_batch.SentinelHubBatch method*), 165
iter_tiling_grids() (*sentinelhub.sentinelhub_batch.SentinelHubBatch static method*), 164

L

lower_left (*sentinelhub.geometry.BBox attribute*), 148

M

make_folder() (*in module sentinelhub.os_utils*), 162
middle (*sentinelhub.geometry.BBox attribute*), 149
MimeType (*class in sentinelhub.constants*), 125

N

next_date() (*in module sentinelhub.time_utils*), 171

O

ogc_string() (*sentinelhub.constants.CRS method*), 124
OgcImageService (*class in sentinelhub.ogc*), 157
OgcRequest (*class in sentinelhub.data_request*), 130
OgcService (*class in sentinelhub.ogc*), 157
opengis_string (*sentinelhub.constants.CRS attribute*), 124
OrbitDirection (*class in sentinelhub.data_collections*), 126
OsmSplitter (*class in sentinelhub.areas*), 111
output() (*sentinelhub.sentinelhub_batch.SentinelHubBatch static method*), 163
output() (*sentinelhub.sentinelhub_request.SentinelHubRequest static method*), 168
output_response() (*sentinelhub.sentinelhub_request.SentinelHubRequest static method*), 168

P

PackageProps (*class in sentinelhub.constants*), 123
parse_datetime() (*sentinelhub.aws.AwsTile static method*), 117
parse_tile_list() (*sentinelhub.aws.AwsProduct static method*), 115
parse_tile_name() (*sentinelhub.aws.AwsTile static method*), 117
parse_time() (*in module sentinelhub.time_utils*), 172
parse_time_interval() (*in module sentinelhub.time_utils*), 172
pixel_to_utm() (*in module sentinelhub.geo_utils*), 146
PolicyBucket (*class in sentinelhub.sentinelhub_rate_limit*), 166

PolicyType (*class in sentinelhub.sentinelhub_rate_limit*), 167
prev_date() (*in module sentinelhub.time_utils*), 171
projection (*sentinelhub.constants.CRS attribute*), 124
provide_session() (*sentinelhub.geopedia.GeopediaSession method*), 153
pyproj_crs (*sentinelhub.constants.CRS attribute*), 124

R

raise_if_invalid() (*sentinelhub.download.request.DownloadRequest method*), 143
read_csv() (*in module sentinelhub.io_utils*), 155
read_data() (*in module sentinelhub.io_utils*), 154
read_image() (*in module sentinelhub.io_utils*), 155
read_jp2_image() (*in module sentinelhub.io_utils*), 155
read_json() (*in module sentinelhub.io_utils*), 155
read_numpy() (*in module sentinelhub.io_utils*), 155
read_tar() (*in module sentinelhub.io_utils*), 154
read_text() (*in module sentinelhub.io_utils*), 155
read_tiff_image() (*in module sentinelhub.io_utils*), 154
read_xml() (*in module sentinelhub.io_utils*), 155
reduce_by_maxcc() (*in module sentinelhub.opensearch*), 160
register_next() (*sentinelhub.sentinelhub_rate_limit.SentinelHubRateLimit method*), 166
rename() (*in module sentinelhub.os_utils*), 162
reprocess_tile() (*sentinelhub.sentinelhub_batch.SentinelHubBatch method*), 165
RequestType (*class in sentinelhub.constants*), 126
reset() (*sentinelhub.config.SHConfig method*), 122
restart() (*sentinelhub.geopedia.GeopediaSession method*), 152
restart_job() (*sentinelhub.sentinelhub_batch.SentinelHubBatch method*), 165
reverse() (*sentinelhub.geometry.BBox method*), 149
reverse() (*sentinelhub.geometry.BBoxCollection method*), 151
reverse() (*sentinelhub.geometry.Geometry method*), 150

S

SafeProduct (*class in sentinelhub.aws_safe*), 118
SafeTile (*class in sentinelhub.aws_safe*), 120
save() (*sentinelhub.config.SHConfig method*), 122
save_data() (*sentinelhub.data_request.DataRequest method*), 130
search_iter() (*in module sentinelhub.opensearch*), 161
sentinelhub.areas (*module*), 111
sentinelhub.aws (*module*), 113
sentinelhub.aws_safe (*module*), 118
sentinelhub.config (*module*), 121
sentinelhub.constants (*module*), 123
sentinelhub.data_collections (*module*), 126
sentinelhub.data_request (*module*), 128
sentinelhub.download.aws_client (*module*), 140
sentinelhub.download.client (*module*), 140
sentinelhub.download.request (*module*), 142
sentinelhub.download.sentinelhub_client (*module*), 144
sentinelhub.fis (*module*), 144
sentinelhub.geo_utils (*module*), 144
sentinelhub.geometry (*module*), 147
sentinelhub.geopedia (*module*), 151
sentinelhub.io_utils (*module*), 154
sentinelhub.ogc (*module*), 157
sentinelhub.opensearch (*module*), 159
sentinelhub.os_utils (*module*), 161
sentinelhub.sentinelhub_batch (*module*), 162
sentinelhub.sentinelhub_rate_limit (*module*), 166
sentinelhub.sentinelhub_request (*module*), 167
sentinelhub.sentinelhub_session (*module*), 169
sentinelhub.testing_utils (*module*), 170
sentinelhub.time_utils (*module*), 171
SentinelHubBatch (*class in sentinelhub.sentinelhub_batch*), 162
SentinelHubDownloadClient (*class in sentinelhub.download.sentinelhub_client*), 144
SentinelHubRateLimit (*class in sentinelhub.sentinelhub_rate_limit*), 166
SentinelHubRequest (*class in sentinelhub.sentinelhub_request*), 167
SentinelHubSession (*class in sentinelhub.sentinelhub_session*), 169
ServiceType (*class in sentinelhub.constants*), 123
ServiceUrl (*class in sentinelhub.constants*), 123
session_headers (*sentinelhub.geopedia.GeopediaSession attribute*), 152
session_headers (*sentinelhub.sentinelhub_session.SentinelHubSession attribute*), 169
session_id (*sentinelhub.geopedia.GeopediaSession*

attribute), 152
session_info (*sentinelhub.geopedia.GeopediaSession attribute*), 152
setUpClass() (*sentinelhub.testing_utils.TestSentinelHub class method*), 170
SHConfig (*class in sentinelhub.config*), 121
SHConstants (*class in sentinelhub.constants*), 126
size() (*in module sentinelhub.os_utils*), 162
sort_download_list() (*sentinelhub.aws.AwsService method*), 114
start_analysis() (*sentinelhub.sentinelhub_batch.SentinelHubBatch method*), 165
start_job() (*sentinelhub.sentinelhub_batch.SentinelHubBatch method*), 165
structure_recursion() (*sentinelhub.aws.AwsService method*), 114
sys_is_windows() (*in module sentinelhub.os_utils*), 162

T

tearDownClass() (*sentinelhub.testing_utils.TestSentinelHub class method*), 170
test_numpy_data() (*sentinelhub.testing_utils.TestSentinelHub method*), 170
TestCaseContainer (*class in sentinelhub.testing_utils*), 170
TestSentinelHub (*class in sentinelhub.testing_utils*), 170
tile_id_to_tile() (*sentinelhub.aws.AwsTile static method*), 118
tile_is_valid() (*sentinelhub.aws.AwsTile method*), 117
TileMissingException, 159
TileSplitter (*class in sentinelhub.areas*), 112
tiling_grid() (*sentinelhub.sentinelhub_batch.SentinelHubBatch static method*), 163
to_utm_bbox() (*in module sentinelhub.geo_utils*), 145
to_wgs84() (*in module sentinelhub.geo_utils*), 146
token (*sentinelhub.sentinelhub_session.SentinelHubSession attribute*), 169
transform() (*sentinelhub.geometry.BBox method*), 149
transform() (*sentinelhub.geometry.BBoxCollection method*), 151
transform() (*sentinelhub.geometry.Geometry method*), 150
transform_point() (*in module sentinelhub.geo_utils*), 147

U

update() (*sentinelhub.sentinelhub_rate_limit.SentinelHubRateLimit method*), 166
update_info() (*sentinelhub.sentinelhub_batch.SentinelHubBatch method*), 164
upper_right (*sentinelhub.geometry.BBox attribute*), 149
url_to_tile() (*sentinelhub.aws.AwsService static method*), 114
user_id (*sentinelhub.geopedia.GeopediaSession attribute*), 152
user_info (*sentinelhub.geopedia.GeopediaSession attribute*), 152
utm_to_pixel() (*in module sentinelhub.geo_utils*), 146

W

WcsRequest (*class in sentinelhub.data_request*), 133
WebFeatureService (*class in sentinelhub.ogc*), 158
wfs_id (*sentinelhub.data_collections.DataCollection attribute*), 128
wgs84_to_pixel() (*in module sentinelhub.geo_utils*), 146
wgs84_to_utm() (*in module sentinelhub.geo_utils*), 145
wkt (*sentinelhub.geometry.BaseGeometry attribute*), 148
WmsRequest (*class in sentinelhub.data_request*), 132
write_bytes() (*in module sentinelhub.io_utils*), 157
write_csv() (*in module sentinelhub.io_utils*), 156
write_data() (*in module sentinelhub.io_utils*), 155
write_image() (*in module sentinelhub.io_utils*), 156
write_jp2_image() (*in module sentinelhub.io_utils*), 156
write_json() (*in module sentinelhub.io_utils*), 156
write_numpy() (*in module sentinelhub.io_utils*), 157
write_text() (*in module sentinelhub.io_utils*), 156
write_tiff_image() (*in module sentinelhub.io_utils*), 156
write_xml() (*in module sentinelhub.io_utils*), 157