Sentinelsat Documentation

Release 0.9.1

Marcel Wille, Kersten Clauss

Jun 01, 2017

Contents

Sentinelsat makes finding and downloading Copernicus Sentinel satellite images from the Sentinels Scientific Datahub easy.

It offers an easy to use command line interface.

```
sentinel search --sentinel2 --cloud 30 user password search_polygon.geojson
```

and a powerful Python API.

```
from sentinelsat.sentinel import SentinelAPI, get_coordinates

api = SentinelAPI('user', 'password')
products = api.query(get_coordinates('search_polygon.geojson'), \
                     producttype = 'SLC', \
                     orbitdirection='ASCENDING')
api.download_all(products)
```

CHAPTER 1

Contents

Installation

Sentinelsat depends on homura, which depends on PycURL. When the dependencies are fulfilled install with `pip install sentinelsat`.

Unix

Ubuntu

```
sudo apt-get install build-essential libcurl4-openssl-dev python-dev python-pip
```

Fedora

```
sudo yum groupinstall "Development Tools"
sudo yum install libcurl libcurl-devel python-devel python-pip
```

Windows

The easiest way to install pycurl is with pycurl wheels provided by Christoph Gohlke

```
pip install pycurl.whl
```

or with‘Conda <http://conda.pydata.org/docs/>‘_

```
conda install pycurl
```

OSX

TODO: How to install on OSX.

Tests

```
git clone https://github.com/ibamacsr/sentinelsat.git
cd sentinelsat
pip install -e .[test]
export SENTINEL_USER=<your scihub username>
export SENTINEL_PASSWORD=<your scihub password>
py.test -v
```

Python Versions and Dependencies

Sentinelsat works with all Python versions >2.7. The convenience functions `to_dataframe()` and `to_geodataframe()` require `Pandas` and/or `GeoPandas` to be present. Due to a `matplotlib` `GeoPandas` can not be installed in Python 3.3 - every other aspect of sentinelsat will work in Python 3.3.

Troubleshooting

The download from Scihub will fail if the server certificate cannot be verified because no default CA bundle is defined, as on Windows, or when the CA bundle is outdated. In most cases the easiest solution is to install or update `certifi`:

`pip install -U certifi` You can also override the the path setting to the PEM file of the CA bundle using the `pass_through_opts` keyword argument when calling `api.download()` or `api.download_all()`:

```
from pycurl import CAINFO
api.download_all(products, pass_through_opts={CAINFO: 'path/to/my/cacert.pem'})
```

Command Line Interface

Sentinelsat's CLI is divided into two commands:

- `sentinel search` to query and download a number of images over an area
- `sentinel download` to download individual images by their unique identifier

Quickstart

A basic search query consists of a search polygon as well as the username and password to access the Scihub.

```
sentinel search [OPTIONS] <user> <password> <geojson>
```

Search areas are provided as GeoJSON polygons, which can be created with QGIS or geojson.io. If you do not specify a start and end date only products published in the last 24 hours will be queried.

Start and end dates refer to the acquisition date given by the beginPosition of the products, i.e. the start of the acquisition time.

Sentinel-1

Search and download all Sentinel-1 scenes of type SLC, in descending orbit for the year 2015.

```
sentinel search -s 20150101 -e 20151231 -d \
-q "producttype=SLC, orbitdirection=Descending" \
-u "https://scihub.copernicus.eu/dhus" <user> <password> <poly.geojson>
```

Download a single Sentinel-1 GRDH scene covering Santa Claus Village in Finland on Christmas Eve 2015.

```
sentinel download --md5 -u "https://scihub.copernicus.eu/dhus/" <user> <password>␣
↪a9048d1d-fea6-4df8-bedd-7bcb212be12e
```

Sentinel-2

Search and download Sentinel-2 scenes for January 2016 with a maximum cloud cover of 40%.

```
sentinel search -s 20160101 -e 20160131 --sentinel2 --cloud 40 <user> <password>
↪<poly.geojson>
```

Download all Sentinel-2 scenes published in the last 24 hours.

```
sentinel search --sentinel2 <user> <password> <poly.geojson>
```

sentinel search

```
sentinel search [OPTIONS] <user> <password> <geojson>
```

Options:

-s	--start	TEXT	Start date of the query in the format YYYYMMDD.
-e	--end	TEXT	End date of the query in the format YYYYMMDD.
-d	--download		Download all results of the query.
-f	--footprints		Create geojson file search_footprints.geojson with footprints of the query result.
-p	--path	PATH	Set the path where the files will be saved.
-q	--query	TEXT	Extra search keywords you want to use in the query. Separate keywords with comma. Example: 'producttype=GRD,polarisationmode=HH'.
-u	--url	TEXT	Define another API URL. Default URL is 'https://scihub.copernicus.eu/apihub/'.
	--md5		Verify the MD5 checksum and write corrupt product ids and filenames to corrupt_scenes.txt.
	--sentinel1		Limit search to Sentinel-1 products.
	--sentinel2		Limit search to Sentinel-2 products.
-c	--cloud	INT	Maximum cloud cover in percent. (Automatically sets –sentinel2)
	--help		Show help message and exit.
	--version		Show version number and exit.

Query parameters:

ESA maintains a list of valid search keywords to query the SciHub.

sentinel download

```
sentinel download [OPTIONS] <user> <password> <productid>
```

Options:

-p	--path	PATH	Set the path where the files will be saved.
-u	--url	TEXT	Define another API URL. Default URL is 'https://scihub.copernicus.eu/apihub/'.
	--md5		Verify the MD5 checksum and write corrupt product ids and filenames to corrupt_scenes.txt.
	--version		Show version number and exit.

Python API

Quickstart

```
# connect to the API
from sentinelsat.sentinel import SentinelAPI, get_coordinates
api = SentinelAPI('user', 'password', 'https://scihub.copernicus.eu/dhus')

# download single scene by known product id
api.download(<product_id>)

# search by polygon, time, and SciHub query keywords
products = api.query(get_coordinates('map.geojson'), \
                     '20151219', date(2015, 12, 29), \
                     platformname = 'Sentinel-2', \
                     cloudcoverpercentage = '[0 TO 30]')

# download all results from the search
api.download_all(products)

# GeoJSON FeatureCollection containing footprints and metadata of the scenes
api.to_geojson(products)
```

Valid search query keywords can be found at the ESA SciHub documentation.

Logging

Sentinelsat logs to `sentinelsat` and the API to `sentinelsat.SentinelAPI`.

There is no predefined logging handler, so in order to have your script print the log messages, either use `logging.baseConfig`

```
import logging

logging.basicConfig(format='%(message)s', level='INFO')
```

or add a custom handler for `sentinelsat` (as implemented in `cli.py`)

```
import logging

logger = logging.getLogger('sentinelsat')
logger.setLevel('INFO')
```

```
h = logging.StreamHandler()
h.setLevel('INFO')
fmt = logging.Formatter('%(message)s')
h.setFormatter(fmt)
logger.addHandler(h)
```

Sorting & Filtering

In addition to the search query keywords sentinelsat allows filtering and sorting of search results before download. To simplify these operations sentinelsat offers the convenience functions `to_dict()`, `to_geojson()`, `to_dataframe()` and `to_geodataframe()` which return the search results as a Python dictionary, a Pandas DataFrame or a GeoPandas GeoDataFrame, respectively. `to_dataframe()` and `to_geodataframe()` require `Pandas` and `GeoPandas` to be installed, respectively.

In this example we query Sentinel-2 scenes over a location and convert the query results to a Pandas DataFrame. The DataFrame is then sorted by cloud cover and ingestiondate. We limit the query to first 5 results within our timespan and download them, starting with the least cloudy scene. Filtering can be done with all data types, as long as you pass the `id` to the download function.

```
# connect to the API
from sentinelsat.sentinel import SentinelAPI, get_coordinates
api = SentinelAPI('user', 'password', 'https://scihub.copernicus.eu/dhus')

# search by polygon, time, and SciHub query keywords
products = api.query(get_coordinates('map.geojson'), \
                     '20151219', date(2015, 12, 29), \
                     platformname = 'Sentinel-2')

# convert to Pandas DataFrame
products_df = api.to_dataframe(products)

# sort and limit to first 5 sorted products
products_df_sorted = products_df.sort_values(['cloudcoverpercentage', 'ingestiondate
↪'], ascending=[True, True])
products_df_sorted = products_df_sorted.head(5)

# download sorted and reduced products in order
for product_id in products_df_sorted["id"]:
  api.download(product_id)
```

API

exception `sentinelsat.sentinel.`**`InvalidChecksumError`**
MD5 checksum of local file does not match the one from the server.

class `sentinelsat.sentinel.`**`SentinelAPI`**(*user*, *password*, *api_url='https://scihub.copernicus.eu/apihub/'*)
Class to connect to Sentinel Data Hub, search and download imagery.

Parameters **user** : string

username for DataHub

password : string

password for DataHub

api_url : string, optional

URL of the DataHub defaults to 'https://scihub.copernicus.eu/apihub'

Attributes

session	(requests.Session object) Session to connect to DataHub
api_url	(str) URL to the DataHub
page_size	(int) number of results per query page current value: 100 (maximum allowed on ApiHub)

Methods

download(id[, directory_path, checksum, ...])	Download a product using homura.
download_all(products[, directory_path, ...])	Download all products returned in query().
format_query(area[, initial_date, end_date])	Create the SciHub API query string
get_product_odata(id)	Access SciHub OData API to get info about a Product.
get_products_size(products)	Return the total filesize in GB of all products in the query
load_query(query[, start_row])	Do a full-text query on the SciHub API using the OpenSearch format specified in
query(area[, initial_date, end_date])	Query the SciHub API with the coordinates of an area, a date interval and any other search keywords accepted by the SciHub API.
to_dataframe(products)	Return the products from a query response as a Pandas DataFrame with the values in their appropriate Python types.
to_dict(products[, parse_values])	Return the products from a query response as a dictionary with the values in their appropriate Python types.
to_geodataframe(products)	Return the products from a query response as a GeoPandas GeoDataFrame with the values in their appropriate Python types.
to_geojson(products)	Return the products from a query response as a GeoJSON with the values in their appropriate Python types.

download(*id*, *directory_path='.'*, *checksum=False*, *check_existing=False*, ***kwargs*)

Download a product using homura.

Uses the filename on the server for the downloaded file, e.g. "S1A_EW_GRDH_1SDH_20141003T003840_20141003T003920_002658_002F54_4DD1.zip".

Incomplete downloads are continued and complete files are skipped.

Further keyword arguments are passed to the homura.download() function.

Parameters **id** : string

UUID of the product, e.g. 'a8dd0cfd-613e-45ce-868c-d79177b916ed'

directory_path : string, optional

Where the file will be downloaded

checksum : bool, optional

If True, verify the downloaded file's integrity by checking its MD5 checksum. Throws InvalidChecksumError if the checksum does not match. Defaults to False.

check_existing : bool, optional

If True and a fully downloaded file with the same name exists on the disk, verify its integrity using its MD5 checksum. Re-download in case of non-matching checksums. Defaults to False.

Returns **path** : string

Disk path of the downloaded file,

product_info : dict

Dictionary containing the product's info from get_product_info().

Raises **InvalidChecksumError**

If the MD5 checksum does not match the checksum on the server.

download_all(*products*, *directory_path='.'*, *max_attempts=10*, *checksum=False*, *check_existing=False*, ***kwargs*)

Download all products returned in query().

File names on the server are used for the downloaded files, e.g. "S1A_EW_GRDH_1SDH_20141003T003840_20141003T003920_002658_002F54_4DD1.zip".

In case of interruptions or other exceptions, downloading will restart from where it left off. Downloading is attempted at most max_attempts times to avoid getting stuck with unrecoverable errors.

Parameters **products** : list

List of products returned with self.query()

directory_path : string

Directory where the downloaded files will be downloaded

max_attempts : int, optional

Number of allowed retries before giving up downloading a product. Defaults to 10.

Returns dict[string, dict|None]

A dictionary with an entry for each product mapping the downloaded file path to its product info (returned by get_product_info()). Product info is set to None if downloading the product failed.

Other Parameters **See download().**

static **format_query**(*area*, *initial_date=None*, *end_date=datetime.datetime(2017, 6, 1, 14, 46, 46, 212979)*, ***keywords*)

Create the SciHub API query string

get_product_odata(*id*)

Access SciHub OData API to get info about a Product. Returns a dict containing the id, title, size, md5sum, date, footprint and download url of the Product. The date field receives the Start ContentDate of the API.

static **get_products_size**(*products*)

Return the total filesize in GB of all products in the query

load_query(*query*, *start_row=0*)

Do a full-text query on the SciHub API using the OpenSearch format specified in https://scihub.copernicus.eu/twiki/do/view/SciHubUserGuide/3FullTextSearch

query(*area*, *initial_date=None*, *end_date=datetime.datetime(2017, 6, 1, 14, 46, 46, 212932)*, ***keywords*)
Query the SciHub API with the coordinates of an area, a date interval and any other search keywords accepted by the SciHub API.

static to_dataframe(*products*)
Return the products from a query response as a Pandas DataFrame with the values in their appropriate Python types.

static to_dict(*products*, *parse_values=True*)
Return the products from a query response as a dictionary with the values in their appropriate Python types.

static to_geodataframe(*products*)
Return the products from a query response as a GeoPandas GeoDataFrame with the values in their appropriate Python types.

static to_geojson(*products*)
Return the products from a query response as a GeoJSON with the values in their appropriate Python types.

exception sentinelsat.sentinel.**SentinelAPIError**(*http_status=None*, *code=None*, *msg=None*, *response_body=None*)
Invalid responses from SciHub.

sentinelsat.sentinel.**get_coordinates**(*geojson_file*, *feature_number=0*)
Return the coordinates of a polygon of a GeoJSON file.

Parameters **geojson_file** : str

location of GeoJSON file_path

feature_number : int

Feature to extract polygon from (in case of MultiPolygon FeatureCollection), defaults to first Feature

Returns polygon coordinates

string of comma separated coordinate tuples (lon, lat) to be used by SentinelAPI

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index