OpenET Client

Release 0.1

Nick Santos

Jun 07, 2022

CONTENTS:

Caution: This project is not affiliated with OpenET and has been developed independently.

CHAPTER

ONE

INSTALLATION

```
python -m pip install openet-client
```

If you want support for spatial operations that help with wrapping the geodatabase API, also run

```
python -m pip install openet-client[spatial]
```

This will attempt to install *geopandas* and *fiona*, which are required for spatial processing. These packages may have trouble (especially on Windows) due to external dependencies. We recommend using a conda environment and the conda packages to simplify that install. In that case, simply use conda to install geopandas to install the necessary dependencies instead of running the above command.

You may also download the repository and run `python setup.py install` to use the package, replacing `python` with the full path to your python interpreter, if necessary.

CHAPTER

TWO

GETTING STARTED

The OpenET Client Python package provides convenience functions to make it easier to request and retrieve data from OpenET's API, such as tools to match your spatial data to OpenET's and send requests for raster data and manage the download process.

Core functionality of the package includes:

- Automatic matching of any geopandas-compatible dataset (including Shapefiles) with OpenET geodatabase API fields to make downloading timeseries data easier
- Fully automated retrieval and attachment of ET values back to input datasets - provide the dataset and API parameters and receive the results back as a field on the input data
- Send multiple requests to the raster API and have the client manage waiting for and downloading the data when it's ready, or have your code continue doing other work and manually trigger a download check later on.

Note: Using this client still requires knowing some information about the OpenET API itself, including the API endpoints you wish to use and the parameters you want to send to the API endpoints. We recommend familiarizing yourself with the documentation for the OpenET API itself to use this Python package. While the API documentation expects you to handle everything related to sending requests to the API and understanding the response, this package handles those tasks for you, but you need to know which requests you want to send.

The OpenET API's documentation can be found at https://open-et.github.io/docs/build/html/index.html

CHAPTER

THREE

USING THE CLIENT WITH THE OPENET GEODATABASE API

The geodatabase API is supported, including functions that allow pulling ET data for spatial objects.

```
import os
import geopandas
import openet_client

features = "PATH TO YOUR SPATIAL DATA" # must be a format geopandas supports, which is␣
↪most spatial data
df = geopandas.read_file(features)

client = openet_client.OpenETClient()
client.token = os.environ["OPENET_TOKEN"]
result = client.geodatabase.get_et_for_features(params={
        "aggregation": "mean",
        "feature_collection_name": "CA",
        "model": "ensemble_mean",
        "variable": "et",
        "start_date": 2018,
        "end_date": 2018
    },
    features=df,
    feature_type=openet_client.geodatabase.FEATURE_TYPE_GEOPANDAS,
    output_field="et_2018_mean_ensemble_mean",
    endpoint="timeseries/features/stats/annual"
)
```

More documentation for this portion of the API will be forthcoming, but note that, like the raster API, you provide a set of parameters that will be sent directly to OpenET based on the endpoint. The function *get_et_for_features* takes many additional parameters that indicate what kind of data you're providing as an input, in this case a geopandas data frame. You also can provide different geodatabase feature endpoints.

This function then calculates the centroid of each feature, finds the fields in OpenET that are associated with those centroids, then downloads the ET data for those fields based on the *params* you provide. It attaches the ET to a data frame as a new field with the name specified in *output_field*. Note that for large features, it does not currently retrieve ET for multiple fields and aggregate them to the larger area. For that functionality, use the raster functionality.

This function also caches the field IDs for the features to avoid future lookups that use API quota. Rerunning the same features with different params will run significantly faster and use significantly fewer API requests behind the scenes.

3.1 Geodatabase API Access Class and Methods

class openet_client.**Geodatabase**(*client*)

feature_ids_list(*params=None*)

The base OpenET Method - sends the supplied params to metadata/openet/region_of_interest/feature_ids_list and returns the requests.Response object

Parameters
params –

Returns

get_et_for_features(*params*, *features*, *feature_type*, *output_field=None*, *geometry_field='geometry'*, *endpoint='timeseries/features/stats/annual'*, *wait_time=5000*, *batch_size=40*, *return_type='joined'*, *join_type='outer'*)

Takes one of multiple data formats (user specified, we're not inspecting it - options are geopandas, geojson) and gets its coordinate values, then gets the field IDs in OpenET for the coordinate pair, retrieves the ET data and returns it as a geopandas data frame with the results in the specified output_field

Parameters

- **params** –
- **features** –
- **endpoint** – which features endpoint should it use?
- **return_type** – How should we return the data? Options are "raw" to return just the JSON from OpenET, "list" to return a list of dictionaries with the OpenET data, "pandas" to return a pandas

  data frame of the results, or "joined" to return the

  data joined back to the input data. "joined" is the default.
- **join_type** – When merging results back in, what type of join should we use? Defaults to "outer" so that records are retained even if no results come back for them. This is also useful behavior when we have multiple timeseries records, such as for monthly results, but it can duplicate input records (not always desirable). To change the behavior, change this to any value supported by pandas.merge or change the return_type so no join occurs.

Returns

get_et_for_openet_feature_list(*feature_ids*, *endpoint*, *params*, *wait_time=5000*, *batch_size=40*)

Retrieve ET for a list of OpenET Feature IDs and return the raw JSON data. To handle retrieving for spatial data you already have, use get_et_for_features instead.

Parameters

- **feature_ids** – a list of strings containing OpenET feature IDs
- **endpoint** – The OpenET endpoint to run the request against. No default
- **params** – The parameters as specified in the OpenET documentation (https://open-et.github.io)

- **wait_time** – How long to wait between requests to avoid hitting a rate limit. Defaults to 5 seconds
- **batch_size** – How large of batches should we use by default? Defaults to 40

Returns

A list of dictionaries as returned from JSON by the OpenET API

get_feature_ids(*features*, *field=None*, *wait_time=5000*)

An internal method used to get a list of coordinate pairs and return the feature ID. Values come back as a dictionary where the input item in the list (coordinate pair shown as DD Longitude space DD latitude) is a dictionary key and the value is the OpenET featureID

Parameters

- **features** –
- **field** – when field is defined, features will be a pandas data frame with a field that has the coordinate values to use. In that case, results will be joined back to the data frame as the field openet_feature_id.
- **wait_time** – how long in ms should we wait between subsequent requests?

Returns

CHAPTER

FOUR

USING THE CLIENT WITH THE OPENET RASTER API

4.1 Examples

One common need is issuing a raster export command and then waiting to proceed until the image is available and downloaded to the current machine. To do that:

> **Warning:** The current approach will make all exported rasters public in order to be able to automatically download them - do not proceed to use this code if that isn't acceptable.

```
import openet_client

# arguments are in the form of a dictionary with keys and
# values conforming to https://open-et.github.io/docs/build/html/ras_export.html
# In the future, geometry may accept OGR or GEOS objects and create the string itself
arguments = {
    'start_date': '2016-01-01',
    'end_date': '2016-03-20',
    'geometry': '-120.72612533471566,37.553211935016215,-120.72612533471566,37.
↪474782294423676,-120.59703597924691,37.474782294423676,-120.59703597924691,37.
↪553211935016215',
    'filename_suffix': 'client_test',
    'variable': 'et',
    'model': 'ensemble',
    'units': 'metric'
}

client = openet_client.OpenETClient("your_open_et_token_value_here")

# note that the path matches OpenET's raster export endpoint
client.raster.export(arguments, synchronous=True)  # synchronous says to wait for it to␣
↪download before proceeding
print(client.raster.downloaded_raster_paths)  # get the paths to the downloaded rasters␣
↪(will be a list, even for a single raster)
```

4.1.1 Batching it

You may also want to queue up multiple rasters, then wait to download them all. To do that, run the `raster.export` commands with `synchronous=False` (the default), then issue a call to `wait_for_rasters`

```
import openet_client

client = openet_client.OpenETClient("your_open_et_token_value_here")
arguments1 = {}  # some set of arguments, similar to the first example
arguments2 = {}  # same
client.raster.export(arguments1)
client.raster.export(arguments2)
client.raster.wait_for_rasters()  # this will keep running until all rasters are␣
↪downloaded - it will wait up to a day by default, but that's configurable by providing␣
↪a `max_time` argument in seconds
print(client.raster.downloaded_raster_paths)  # a list with all downloaded rasters
# or
rasters = client.raster.registry.values()  # get all the Raster objects including remote␣
↪URLs and local paths
```

Doing work while you wait + manual control

You might also not want to *wait* around for the rasters to export, but still have control over the process. Here's how to manually control the flow

```
import openet_client

client = openet_client.OpenETClient("your_open_et_token_value_here")
arguments = {}  # some set of arguments, similar to the first example
my_raster = client.raster.export(arguments)

# ... any other code you like here - the OpenET API will do its work and make the raster␣
↪ready - or not, depending on your place in their queue ...

client.raster.check_statuses()  # check the API's all_files endpoint to see which rasters␣
↪are ready
if my_raster.status == openet_client.raster.STATUS_AVAILABLE  # check that the raster we␣
↪want is now ready
    client.raster.download_available_rasters()  # try to download the ones that are␣
↪ready and not yet downloaded (from this session)
```

4.2 Raster API Class and Methods

class openet_client.raster.**Raster**(*request_result*)

Internal object for managing raster exports - tracks current status, the remote URL and the local file path once it exists.

download_file(*retry_interval=20*, *max_wait=600*)

Attempts to download a raster, assuming it's ready for download. Will make multiple attempts over a few minutes because sometimes it takes a while for the permissions to propagate, so the

first few responses may give a 403 error. We then have a timeout (max_wait) where if we exceed that value, we exit anyway without downloading.

Downloads the file to a tempfile path - the user may move the file after that if they wish.

Parameters

- **retry_interval** – time in seconds between repeated attempts
- **max_wait** – How long, in seconds should we wait for the correct permissions before stopping attempts to download.

Returns

class openet_client.raster.**RasterManager**(*client*)

The manager that becomes the .raster attribute on the OpenETClient object. Handles submitting raster export requests and polling for completed exports.

As constructed, could slow down if it handles many thousands of raster exports, or if the all_files OpenET endpoint displays lots of files as options.

Generally speaking, you won't create this object yourself, but you can set client.raster.wait_interval to the length of time, in seconds, that the manager should wait between polling the all_files endpoint for new exports when waiting for new rasters.

property available_rasters

Which rasters have we marked as ready to download, but haven't yet been retrieved?

Returns

check_statuses(*rasters=None*)

Updates the status information on each raster only - does not attempt to download them.

Parameters
rasters – the list of rasters to update the status of - if not provided, defaults to all rasters that are queued and not downloaded

Returns
None

download_available_rasters()

Attempts to download all available rasters individually

Returns

export(*params=None*, *synchronous=False*, *public=True*, *transform=False*)

Handles the raster/export endpoint for OpenET. Optionally waits for the raster to be exported and downloaded before proceeding. See documentation examples for usage details.

Parameters

- **params** – A dictionary of arguments with keys matching the raster/export endpoints parameters and values matching the requirements for the values of those keys. If the "geometry" key of this value is a GEOS or an OGR object with a .coords attribute, then the correct

geometry string to send to the API will be composed for you. Otherwise, it is your responsibility to match the values with the API's requirements.

- **synchronous** – Whether or not to wait for the raster to export and be downloaded before exiting this function and proceeding
- **public** – Whether or not to make the raster public - at this point, keeping this as True is required for all the features of this package to work, but if you just want to use the package to submit a bunch of raster jobs, but not to *download* those rasters, then you may set this to False.
- **transform** – If a GEOS or OGR object with `transform` and `coords` attributes is passed in as part of params["geography"], such as an object from GeoDjango, then you can optionally specify that this function handle a simple transformation to WGS84 (EPSG 4326) for you. For more complicated transformations or datum transformations, it may be best to handle the transformation before running this function. Setting it to False doesn't control usage of a GEOS/OGR object is used, only if its coordinates are transformed first.

Returns

Raster object - when synchronous, the local_file attribute will have the path to the downloaded raster on disk - otherwise it will have the status of the raster

`property queued_rasters`

Which rasters are we still waiting for?

Returns

`wait_for_rasters`(*uuid=None*, *max_time=86400*)

When we want to just wait until the rasters are ready, we call this method, which polls the all_files endpoint at set intervals and checks which rasters are done.

Then updates the statuses on individual rasters and calls the download functions for the individual rasters and time some are ready to download, but won't exit until all rasters are available and have had at least one download attempt.

Parameters

- **uuid** –
- **max_time** – Maximum time in seconds to wait for all rasters to complete - defaults to 86400 (a day)

Returns

CHAPTER

FIVE

SENDING YOUR OWN REQUESTS TO THE API

5.1 Client Class and Methods

class openet_client.**OpenETClient**(*token=None*)

send_request(*endpoint*, *method='get'*, *disable_encoding=False*, ***kwargs*)

Handles sending most requests to the API - they provide the endpoint and the args. Since the API is in the process of switching from GET to POST requests, we have logic that switches between those depending on the request method

Parameters

- **endpoint** – The text path to the OpenET endpoint - e.g. raster/export - skip the base URL.
- **method** – "get" or "post" (case sensitive) - should match what the API supports for the endpoint
- **kwargs** – The arguments to send (via get or post) to the API

Returns

requests.Response object of the results.

CHAPTER

SIX

THE DATA DOWNLOAD CACHE

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX