threedi-api-client

Nelen & Schuurmans

Aug 25, 2023

CONTENTS:

- A Python library for interfacing with the 3Di API
- Free software: BSD license
- Documentation: https://threedi-api-client.readthedocs.io

CHAPTER

ONE

FEATURES

- Object-oriented API interaction generated with https://openapi-generator.tech/.
- Asynchronous support.
- Advanced file download and upload utility functions.

CHAPTER

TWO

INSTALLATION

We recommend *pip* to install this package:

```
pip install --user threedi-api-client
```

If async support is required, install as follows:

```
pip install --user threedi-api-client[aio]
```

CHAPTER

THREE

CREDITS

The OpenAPI client has been generated with OpenAPI generator (https://openapi-generator.tech/), which is licensed under the Apache License 2.0.

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

3.1 API Reference

3.1.1 Usage

The `threedi_api_client.ThreediApi` is the main entry point to make calls to the 3Di API. It handles the login process for you and can be directly used as client for all API endpoints.

In earlier versions of this library the main entry point was `threedi_api_client.ThreediApiClient`. This method will remain available until `threedi_api_client` version 4.0. Read below how to migrate from `ThreediApiClient` to `ThreediApi`.

class threedi_api_client.**ThreediApi**(*env_file=None*, *config=None*, *version='v3'*, *asynchronous=False*, *retries=3*)

Client for the 3Di API.

The API object exposes numerous methods that interface with the API, all named according to the pattern `{resource}_{action}`, for example `simulations_list`. Consult the docstrings of these methods for further information

ThreediApi requires a THREEDI_API_HOST and user credentials. These can either be stored in a `.env` file, supplied via environment variables, or passed as a config dictionary.

The preferred way of setting user credentials is using a "Personal API Key" in the variable 'THREEDI_API_PERSONAL_API_TOKEN'. Using 'THREEDI_API_USERNAME' and 'THREEDI_API_PASSWORD' to authenticate is also supported, but considered legacy. Consider upgrading to personal API keys if you are using this form of authentication.

1) A sample `.env` file could look like this:

```
THREEDI_API_HOST=https://api.3di.live
THREEDI_API_PERSONAL_API_TOKEN=lHtKAuCV.secret123
```

This is used in your script as follows:

```
from threedi_api_client import ThreediApi

env_file = "<path>/.env"
```

(continues on next page)

(continued from previous page)

```
with ThreediApi(env_file=env_file) as api:
    ...
```

2) The same variables can be set as environment variables from the terminal that you use to run the python interpreter with. On Windows:

```
set THREEDI_API_HOST "https://api.3di.live"
set THREEDI_API_PERSONAL_API_TOKEN "lHtKAuCV.secret123"
```

On Linux or OSX:

```
export THREEDI_API_HOST=https://api.3di.live
export THREEDI_API_PERSONAL_API_TOKEN=lHtKAuCV.secret123
```

3) The `config` keyword argument can be used like:

```
from threedi_api_client import ThreediApi
from getpass import getpass

config = {
    "THREEDI_API_HOST": "https://api.3di.live",
    "THREEDI_API_PERSONAL_API_TOKEN": getpass(),
}

with ThreediApi(config=config) as api:
    ...
```

Parameters

- **env_file** (*str or pathlib.Path*) – path to a configuration file
- **config** (*dict*) – configuration dictionary for this client
- **version** (*str*) – the API version to use (default: 'v3')
- **asynchronous** (*bool*) – whether to return an asynchronous API client for usage with asyncio. Note: this requires installation of `threedi_api_client[aio]`.
- **retries** (*int or object*) – the number of retries; see notes section for more granular control over the retry policy

Returns

This class constructs an Api object that was autogenerated by the OpenAPI generator. The autogenerated code lives under `threedi_api_client.openapi`.

Notes

OAuth2 For using OAuth2, supply an THREEDI_API_ACCESS_TOKEN. If the THREEDI_API_ACCESS_TOKEN has expired, there are 3 methods implemented for automatic token renewal:

- refresh token without client secret: supply a THREEDI_API_REFRESH_TOKEN
- refresh token with client secret: supply a THREEDI_API_REFRESH_TOKEN and THREEDI_API_CLIENT_SECRET
- client credentials flow: supply a THREEDI_API_CLIENT_SECRET

Timeouts

A request without a timeout may block your python script indefinitely. It is always a good idea to prevent this by setting a timeout. Do this using the `_request_timeout` parameter on every api request. It is currently not possible to configure a default timeout.

Retry policy

It is common to configure a retry policy to prevent exceptions due to the service being temporarily unavailable. The `ThreediApi` supports this through the `retries` parameter. Due to different backends, the configuration details differ between synchronous and asynchronous usage.

For basic usage, supply an integer (which is the maximum number of retries):

```
>>> api = ThreediApi(..., retries=3)
```

For synchronous usage, you may also supply a `urllib3.util.Retry` object (see urllib3 docs):

```
>>> import urllib3
>>> policy = urllib3.util.Retry(total=3, backoff_factor=1.0)
>>> api = ThreediApi(..., retries=policy)
```

For asynchronous usage, you may also supply a `aiohttp_retry.ExponentialRetry` object. See the aiohttp_retry docs). The `aiohttp_retry` package is shipped with `threedi_api_client`.

```
>>> from threedi_api_client.aio import aiohttp_retry
>>> policy = aiohttp_retry.ExponentialRetry(attempts=3, factor=1.0)
>>> api = ThreediApi(..., retries=policy)
```

Other configuration options are:

```
- the exceptions on which to retry (default: None)
- the statuses on which to retry (default: 413, 429, 503, 504)
- the HTTP methods on which to retry (default: 'DELETE', 'GET', 'HEAD', 'OPTIONS',
↪'PUT', 'TRACE')
```

Migration from ThreediApiClient

Formerly, the `threedi_api_client.ThreediApiClient` was used to interact with the 3Di API. As of `threedi_api_client` version 4, this method is deprecated. Currently, both methods are allowed, but the legacy one will give warnings.

There are three changes:

1. The `ThreediApi` object directly exposes methods to interact with 3Di API resources. There is no need of separately constructing *api* objects for each resource. The root package `openapi_client` will disappear in future versions: do not import it anymore. Direct access to the (new) generated API code is possible through `threedi_api_client.openapi`.
2. The configuration variables are now prefixed with `"THREEDI_API_"` instead of `"API_"`.
3. The `"THREEDI_API_HOST"` must not include the version.
4. Advanced users of the asynchronous client (imported from `threedi_api_client.aio`) should start using `threedi_api_client.ThreediApi` with `asynchronous=True`.

Take for example a script that looks like this:

```
from threedi_api_client import ThreediApiClient
from openapi_client import SimulationsApi

config = {
    "API_HOST": "https://api.3di.live/v3.0"
    "API_USERNAME": "your.username"
    "API_PASSWORD": "your.password"
}

with ThreediApiClient(config=config) as api_client:
    api = SimulationsApi(api_client)
    result = api.simulations_list()
```

Applying the changes listed above, it is refactored to this:

```
from threedi_api_client import ThreediApi

config = {
    "THREEDI_API_HOST": "https://api.3di.live",  # no version!
    "THREEDI_API_PERSONAL_API_TOKEN": "your_personal_api_token_here"
}

with ThreediApi(config=config) as api:
    result = api.simulations_list()
```

3.1.2 Download/upload

This library supplies utility functions for file downloading and uploading. The functions support automatic retries, multipart downloads and streaming uploads. Their usage is described below.

Synchronous

threedi_api_client.files.**download_file**(*url: str*, *target: Path*, *chunk_size: int = 16777216*, *timeout: Optional[Union[float, Timeout]] = 5.0*, *pool: Optional[PoolManager] = None*, *callback_func: Optional[Callable[[int, int], None]] = None*) → Tuple[Path, int]

Download a file to a specified path on disk.

It is assumed that the file server supports multipart downloads (range requests).

Parameters

- **url** – The url to retrieve.
- **target** – The location to copy to. If this is an existing file, it is overwritten. If it is a directory, a filename is generated from the filename in the url.
- **chunk_size** – The number of bytes per request. Default: 16MB.
- **timeout** – The total timeout in seconds.
- **pool** – If not supplied, a default connection pool will be created with a retry policy of 3 retries after 1, 2, 4 seconds.
- **callback_func** – optional function used to receive: bytes_downloaded, total_bytes for example: def callback(bytes_downloaded: int, total_bytes: int) -> None

Returns

Tuple of file path, total number of downloaded bytes.

Raises

- **threedi_api_client.openapi.ApiException** – raised on unexpected server responses (HTTP status codes other than 206, 413, 429, 503)
- **urllib3.exceptions.HTTPError** – various low-level HTTP errors that persist after retrying: connection errors, timeouts, decode errors, invalid HTTP headers, payload too large (HTTP 413), too many requests (HTTP 429), service unavailable (HTTP 503)

threedi_api_client.files.**upload_file**(*url: str*, *file_path: Path*, *chunk_size: int = 16777216*, *timeout: Optional[Union[float, Timeout]] = None*, *pool: Optional[PoolManager] = None*, *md5: Optional[bytes] = None*, *callback_func: Optional[Callable[[int, int], None]] = None*, *headers: Optional[Dict] = None*) → int

Upload a file at specified file path to a url.

Parameters

- **url** – The url to upload to.
- **file_path** – The file path to read data from.
- **chunk_size** – The size of the chunk in the streaming upload. Note that this function does not do multipart upload. Default: 16MB.
- **timeout** – The total timeout in seconds. The default is a connect timeout of 5 seconds and a read timeout of 10 minutes.

- **pool** – If not supplied, a default connection pool will be created with a retry policy of 3 retries after 1, 2, 4 seconds.
- **md5** – The MD5 digest (binary) of the file. Supply the MD5 to enable server-side integrity check. Note that when using presigned urls in AWS S3, the md5 hash should be included in the signing procedure.
- **callback_func** – optional function used to receive: bytes_uploaded, total_bytes for example: def callback(bytes_uploaded: int, total_bytes: int) -> None
- **headers** – optional extra headers for the PUT request.

Returns
The total number of uploaded bytes.

Raises

- **IOError** – Raised if the provided file is incompatible or empty.
- **threedi_api_client.openapi.ApiException** – raised on unexpected server responses (HTTP status codes other than 206, 413, 429, 503)
- **urllib3.exceptions.HTTPError** – various low-level HTTP errors that persist after retrying: connection errors, timeouts, decode errors, invalid HTTP headers, payload too large (HTTP 413), too many requests (HTTP 429), service unavailable (HTTP 503)

Asynchronous

async threedi_api_client.aio.files.**download_file**(*url: str*, *target: Path*, *chunk_size: int = 16777216*, *timeout: Optional[Union[float, ClientTimeout]] = None*, *connector: Optional[BaseConnector] = None*, *executor: Optional[ThreadPoolExecutor] = None*, *retries: int = 3*, *backoff_factor: float = 1.0*, *callback_func: Optional[Callable[[int, int], Awaitable[None]]] = None*) → Tuple[Path, int]

Download a file to a specified path on disk.

It is assumed that the file server supports multipart downloads (range requests).

Parameters

- **url** – The url to retrieve.
- **target** – The location to copy to. If this is an existing file, it is overwritten. If it is a directory, a filename is generated from the filename in the url.
- **chunk_size** – The number of bytes per request. Default: 16MB.
- **timeout** – The total timeout of the download of a single chunk in seconds. By default, there is no total timeout, but only socket timeouts of 5s.
- **connector** – An optional aiohttp connector to support connection pooling. If not supplied, a default TCPConnector is instantiated with a pool size (limit) of 4.
- **executor** – The ThreadPoolExecutor to execute local file I/O in. If not supplied, default executor is used.
- **retries** – Total number of retries per request.
- **backoff_factor** – Multiplier for retry delay times (1, 2, 4, . . .)

- **callback_func** – optional async function used to receive: bytes_downloaded, total_bytes for example: async def callback(bytes_downloaded: int, total_bytes: int) -> None

Returns
Tuple of file path, total number of uploaded bytes.

Raises

- **threedi_api_client.openapi.ApiException** – raised on unexpected server responses (HTTP status codes other than 206, 413, 429, 503)
- **aiohttp.ClientError** – various low-level HTTP errors that persist after retrying: connection errors, timeouts, decode errors, invalid HTTP headers, payload too large (HTTP 413), too many requests (HTTP 429), service unavailable (HTTP 503)

async threedi_api_client.aio.files.**upload_file**(*url: str*, *file_path: Path*, *chunk_size: int = 16777216*, *timeout: Optional[Union[float, ClientTimeout]] = None*, *connector: Optional[BaseConnector] = None*, *md5: Optional[bytes] = None*, *executor: Optional[ThreadPoolExecutor] = None*, *retries: int = 3*, *backoff_factor: float = 1.0*, *callback_func: Optional[Callable[[int, int], Awaitable[None]]] = None*) → int

Upload a file at specified file path to a url.

Parameters

- **url** – The url to upload to.
- **file_path** – The file path to read data from.
- **chunk_size** – The size of the chunk in the streaming upload. Note that this function does not do multipart upload. Default: 16MB.
- **timeout** – The total timeout of the upload in seconds. By default, there is no total timeout, but only socket connect timeout of 5 seconds and a socket read timeout of 10 minutes.
- **connector** – An optional aiohttp connector to support connection pooling.
- **md5** – The MD5 digest (binary) of the file. Supply the MD5 to enable server-side integrity check. Note that when using presigned urls in AWS S3, the md5 hash should be included in the signing procedure.
- **executor** – The ThreadPoolExecutor to execute local file I/O and MD5 hashing in. If not supplied, default executor is used.
- **retries** – Total number of retries per request.
- **backoff_factor** – Multiplier for retry delay times (1, 2, 4, . . .)
- **callback_func** – optional async function used to receive: bytes_uploaded, total_bytes for example: async def callback(bytes_uploaded: int, total_bytes: int) -> None

Returns
The total number of uploaded bytes.

Raises

- **IOError** – Raised if the provided file is incompatible or empty.
- **threedi_api_client.openapi.ApiException** – raised on unexpected server responses (HTTP status codes other than 206, 413, 429, 503)

- **aiohttp.ClientError** – various low-level HTTP errors that persist after retrying: connection errors, timeouts, decode errors, invalid HTTP headers, payload too large (HTTP 413), too many requests (HTTP 429), service unavailable (HTTP 503)

3.1.3 Examples

First, get an instance of the `ThreediApi`:

```
from threedi_api_client import ThreediApi
env_file = "<path>/.env"
api = ThreediApi(env_file)
```

Now you can easily make use of the api models generated by the openapi client. Let us create a simulation. We will use a `Simulation` model instance to pass data to the API. Some fields are optional but we do need to specify:

- the unique organisation ID we want to run the simulation for
- the model schema to use for the simulation by referring to the id of the threedimodel resource
- datetime (in ISO 8601 (UTC) format) for the simulation start
- either a end datetime (also in ISO 8601 (UTC) format) or the duration parameter in seconds

If you do not know the unique ID for your organisation you can make use of the API to request it.

```
api.organisations_list(name__istartswith="nelen")
{'count': 2,
'next': None,
'previous': None,
'results': [{'name': 'Nelen & Schuurmans',
      'unique_id': 'b8f91de705774fe8a4e7cb2d9413bf5c',
      'url': 'https://api.3di.live/v3.0/organisations/61f5a464c35044c19bc7d4b42d7f58cb/'}
↪,
     {'name': 'Nelen & Schuurmans alleen werknemers',
      'unique_id': 'e82c74c4fb5846b3ae990c0cc69130c6',
      'url': 'https://api.3di.live/v3.0/organisations/cde64bc165644be9af023fc4fa18d098/'}
↪]}
```

Now we can create the `Simulation` model instance.

```
from threedi_api_client.openapi import Simulation

# start date will be a datetime object
from datetime import datetime

my_extreme_event_simulation = Simulation(
        name="my extreme event",    # (optional)
        threedimodel=1,             # The model schema to use for the simulation by␣
↪referring to the id of the threedimodel resource
        organisation='b8f91de705774fe8a4e7cb2d9413bf5c',
        start_datetime=datetime.utcnow(),  # accepts datetime instance
        duration=7200               # in secs ==> 2 hours
)
```

The `simulations_create` method allows you to create a new Simulation resource.

```
api.simulations_create(my_extreme_event_simulation)
{'created': 'now',
'duration': 7200,
'duration_humanized': '2 hours, 0 minutes, 0 seconds',
'end_datetime': '2019-11-04T16:19:46Z',
'id': 631,
'name': 'my extreme event',
'organisation': 'b8f91de705774fe8a4e7cb2d9413bf5c',
'organisation_name': 'Nelen & Schuurmans',
'slug': 'my-extreme-event-378f55a5-06df-4021-8fb6-65bbb70519dc',
'start_datetime': '2019-11-04T14:19:46Z',
'threedimodel': 'https://api.3di.live/v3.0/threedimodels/1/',
'threedimodel_id': '1',
'url': 'https://api.3di.live/v3.0/simulations/631/',
'user': 'lars.claussen',
'uuid': '378f55a5-06df-4021-8fb6-65bbb70519dc'}
```

Simulations allow for adding an arbitrary number of events to them like

- rain events
- sources and sinks
- initial conditions
- laterals
- saved states
- structure controls

All of them have their own openapi client model. To add a constant rain event to the simulation you would do the following.

```
from threedi_api_client.openapi import ConstantRain
const_rain = ConstantRain(
    simulation=631,   # the ID we got from our create call above
    offset=60,        # let the rain start after one minute
    duration=5000,    # let the rain last for 5000 secs
    value=0.0006,     # not too extreme after all...;-)
    units="m/s"       # the only unit supported for now
)
api.simulations_events_rain_constant_create(631, const_rain)
{'duration': 5000,
'offset': 60,
'simulation': 'https://api.3di.live/v3.0/simulations/631/',
'units': 'm/s',
'url': 'https://api.3di.live/v3.0/simulations/631/events/rain/constant/17/',
'value': 0.0006}
```

If you want to see which events are defined on a given simulation

```
api.simulations_events(631)
{'boundaries': None,
'breach': [],
'filerasterrain': [],
```

(continues on next page)

(continued from previous page)

```
'filerastersourcessinks': [],
'filetimeseriesrain': [],
'filetimeseriessourcessinks': [],
'initial_groundwaterlevel': None,
'initial_onedwaterlevel': None,
'initial_onedwaterlevelpredefined': None,
'initial_savedstate': None,
'initial_twodwaterlevel': None,
'laterals': [],
'lizardrasterrain': [],
'lizardrastersourcessinks': [],
'lizardtimeseriesrain': [],
'lizardtimeseriessourcessinks': [],
'savedstates': [],
'timedstructurecontrol': [],
'timeseriesrain': [{'constant': True,
                    'duration': 5000,
                    'interpolate': False,
                    'offset': 60,
                    'simulation': 'https://api.3di.live/v3.0/simulations/631/',
                    'units': 'm/s',
                    'url': 'https://api.3di.live/v3.0/simulations/631/events/rain/timeseries/
↪17/',
                    'values': [[0.0, 0.0006], [5000.0, 0.0]]}],
'timeseriessourcessinks': []}
```

Advanced usage

See below for an example of uploading a rain raster.

```
from pathlib import Path
from threedi_api_client.files import upload_file

simulation_pk = 1
filename = 'bergermeer_rasters_from_geotiffs.nc'
local_file_path = Path('./data/bergermeer_rasters_from_geotiffs.nc')

# Create rain raster upload resource in API
# returns a 'file_upload' instance containing a
# put_url property which is the URL to the object
# storage object to be uploaded with an HTTP PUT requests.
file_upload = api.simulations_events_rain_rasters_upload(
    filename, simulation_pk)

# Upload the file
upload_file(file_upload.put_url, local_file_path)
```

Async client

This project also provides an asynchronous api client. To use the async-client make sure you install the optional dependencies using `pip install threedi-api-client[aio]` and then import from the `aio` submodule. The async-client works the same as the synchronous client, except all api calls are coroutines.

For example, to asynchronously request files from the api:

```
import asyncio

from threedi_api_client.api import ThreediApi
from threedi_api_client.openapi.api.v3_api import V3Api

config = {
    "THREEDI_API_HOST": "https://api.3di.live",
    "THREEDI_API_PERSONAL_API_TOKEN": "your_personal_api_token_here"
}


async def main():
    async with ThreediApi(config=config) as api_client:
        api_client: V3Api
        print(await api_client.files_list())


if __name__ == '__main__':
    asyncio.run(main())
```

PYTHON MODULE INDEX

t

INDEX