xyzspaces

HERE Europe B.V.

Apr 22, 2021

GETTING STARTED

Manage your XYZ Hub or HERE Data Hub spaces from Python.

XYZ is an Open Source, real-time, cloud database system providing access to large geospatial data at scale. An XYZ "Hub" manages "spaces" that contain "features" (geodata "records") with tags and properties, with spaces and features having unique IDs. A RESTful API exists to provide low-level access to interact with a XYZ Hub.

This Python package allows to interact with your XYZ spaces and features on a given Hub using a higher level programmatic interface that wraps the RESTful API. Using this package you can:

- Create, read, list, update, share, delete spaces (also: get space info and stats).
- Add, read, update, iterate, search, cluster (hex/quad bins), delete features.
- Search features by ID, tag, property, bbox, tile, radius, geometry.

Based on the XYZ Hub the HERE Data Hub is a commercial service (with a free plan), that offers some additional features (in a pro plan), like clustering, virtual spaces, activity logs, schema validation, rule based tagging and likely more to come.

CHAPTER

ONE

PREREQUISITES

Before you install the `xyzspaces` package make sure you meet the following prerequisites:

- A Python installation, 3.6+ recommended, with the `pip` or `conda` command available to install dependencies.
- A HERE developer account, free and available under HERE Developer Portal.
- An XYZ API access token from your XYZ Hub server or the XYZ portal (see also its Getting Started section) in an environment variable named XYZ_TOKEN which you can set like this (with a valid value, of course):

```
export XYZ_TOKEN=MY-XYZ-TOKEN
```

CHAPTER

TWO

INSTALLATION

xyzspaces depends for its spatial functionality on a large geospatial, open source stack of libraries (Geopandas, turfpy). See the *Dependencies* section below for more details. The C depedencies of Geopandas such as (GEOS, GDAL, PROJ) can sometimes be a challenge to install. Therefore, we advise you to closely follow the recommendations below to avoid installation problems.

2.1 Installing with Anaconda / conda

To install xyzspaces and all its dependencies, we recommend to use the conda package manager. This can be obtained by installing the Anaconda Distribution (a free Python distribution for data science), or through miniconda (minimal distribution only containing Python and the conda package manager). See also the installation docs for more information on how to install Anaconda or miniconda locally.

The advantage of using the conda package manager is that it provides pre-built binaries for all the required dependencies of xyzspaces for all platforms (Windows, Mac, Linux).

To install the latest version of xyzspaces from conda-forge, you can then do:

```
conda install -c conda-forge xyzspaces
```

2.1.1 Creating a new environment

Creating a new environment is not strictly necessary, but given that installing other geospatial packages from different channels may cause dependency conflicts (as mentioned in the note above), it can be good practice to install the geospatial stack in a clean environment starting fresh.

The following commands create a new environment with the name `xyz_env`, configures it to install packages always from conda-forge, and installs xyzspaces in it:

```
conda create -n xyz_env
conda activate xyz_env
conda config --env --add channels conda-forge
conda config --env --set channel_priority strict
conda install python=3 xyzspaces
```

2.2 Installing with pip

xyzspaces can also be installed with pip:

```
pip install xyzspaces
```

Warning: When using pip to install xyzspaces, you need to make sure that all dependencies of Geopandas are installed correctly.

- fiona provides binary wheels with the dependencies included for Mac and Linux, but not for Windows.
- pyproj and shapely provide binary wheels with dependencies included for Mac, Linux, and Windows.
- rtree does not provide wheels.
- Windows wheels for *shapely*, *fiona*, *pyproj* and *rtree* can be found at Christopher Gohlke's website.

So depending on your platform, you might need to compile and install their C dependencies manually. We refer to the individual packages for more details on installing those. Using conda (see above) avoids the need to compile the dependencies yourself.

2.3 Installing from source

You may install the latest development version by cloning the *GitHub* repository and using pip to install from the local directory:

```
git clone https://github.com/heremaps/xyz-spaces-python.git
cd xyz-spaces-python
pip install .
```

It is also possible to install the latest development version directly from the GitHub repository with:

```
pip install -e git+https://github.com/heremaps/xyz-spaces-python#egg=xyzspaces
```

For installing xyzspaces from source, the same *note* on the need to have all dependencies correctly installed applies.

See the *section on conda* above for more details on getting running with Anaconda.

2.4 Dependencies

Required dependencies:

- requirements

Dev dependencies:

- dev requirements

CHAPTER

THREE

EXAMPLES

The Jupyter notebooks show various functionalities of xyzspaces. You can directly play with examples by clicking on the binder button:

To run the example notebooks locally See docs/notebooks/README.md.

The GIF below shows an interaction with an example notebook, demonstrating how to use a spatial search on a big public dataset, loaded from the HERE Data Hub.

3.1 Interactive examples

These are some preliminary code snippets that can be executed online. Click on the "Start" button first before you execute the following cells!

The following is a cell. Click the "run" button or press Shift-Enter inside the cell to execute it. Launching the computation backend may take a few seconds, and you may need to re-start it.

```
import xyzspaces
xyzspaces.__version__
```

This is another cell. Replace the "MY_XYZ_TOKEN" with your real XYZ token and click "run" again to search for the GeoJSON feature of the White House in Washington, DC, USA!

```
import os
import geojson
from xyzspaces.datasets import get_microsoft_buildings_space

os.environ["XYZ_TOKEN"] = "MY_XYZ_TOKEN"
space = get_microsoft_buildings_space()
feat = next(space.search(tags=["postalcode@20500"]))
print(geojson.dumps(feat, indent=4))
```

More to come... please stay tuned!

CHAPTER

FOUR

XYZSPACES PACKAGE

The XYZ Spaces for Python - manage your XYZ Hub server or HERE Data Hub.

XYZ Spaces for Python allows you to manage XYZ spaces, projects and tokens with the Hub API, Project API, and Token API, respectively. The Hub API provides the most features to let you read and write GeoJSON data (features) from and to an XYZ space, and perform some higher-level operations like return features inside or clipped by some bounding box or map tile. The Project API and Token API let you manage your XYZ projects and tokens.

See also: - XYZ Hub server: https://github.com/heremaps/xyz-hub - HERE Data Hub: https://developer.here.com/products/data-hub

class xyzspaces.**XYZ**(*credentials=None*, *server='https://xyz.api.here.com'*)

Bases: object

A single interface to interact with your XYZ Hub server or HERE Data Hub.

__init__(*credentials=None*, *server='https://xyz.api.here.com'*)

Instantiate an XYZ object, optionally with access credentials and custom base URL.

Parameters

- **credentials** (*Optional[str]*) – A string to serve as authentication (a bearer token). Will be looked up in environment variable XYZ_TOKEN if not provided.
- **server** (*str*) – A string as base URL for Data Hub APIs. Required only if Data Hub APIs are self-hosted. For self-hosted Data Hub instances credentials is not required.

4.1 Subpackages

4.1.1 xyzspaces.datasets package

This package provides access to some public datasets used.

xyzspaces.datasets.**get_countries_data**()

Pull countries example GeoJSON from the net or a locally cached file.

If this is not locally cached, yet, it will be after the first call, unless the file cannot be saved, in which case it will be re-downloaded again with every call.

Source (under http://unlicense.org): https://raw.githubusercontent.com/johan/world.geo.json/master/countries.geo.json

The data contains 180 countries, and does not cover all existing countries, ca. 200. For example the Vatican is missing.

Returns A JSON object.

xyzspaces.datasets.**get_chicago_parks_data**()
Create GeoJSON from file chicago_parks.geo.json stored locally.

xyzspaces.datasets.**get_microsoft_buildings_space**()
Create a space object for the MS "US Buildings Footprints" dataset.

The original source for this dataset can be found on https://github.com/Microsoft/USBuildingFootprints.

Returns A space object.

4.2 Submodules

4.2.1 xyzspaces.__version__ module

Project version information.

4.2.2 xyzspaces.apis module

This module does access the APIs of an XYZ Hub server or HERE Data Hub.

It provides classes like *HubApi*, *ProjectApi*, and *TokenApi* to interact with the respective XYZ APIs in a programmatic way.

class xyzspaces.apis.**Api**(*server*, *credentials=None*)
Bases: object

A low-level HTTP RESTful API client.

This uses requests to make HTTP requests with typical parameters and will return the entire response when calling instances directly, or when using the aliased HTTP methods like *Api.get()*, *Api.put()* etc. provided for convenience.

All these methods like *Api.get()*, *Api.put()* etc. will raise ApiError if the status code of the HTTP response is not in the interval [200, 300).

This class is a base class for concrete HERE XYZ APIs, but can also be used outside of that particular context for any RESTful API, maybe with slight changes regarding authentication.

__init__(*server*, *credentials=None*)
Instantiate an *Api* object.

Parameters

- **server** (*Optional[str]*) – The URL of the server implementing the API.
- **credentials** (*Optional[Any]*) – Any object to serve as authentication (not used in this class).

__call__(*method*, *path=''*, *params=None*, *headers=None*, *cookies=None*, *json=None*, *data=None*, *proxies=None*)
Make an API call with parameters passed to requests.

Parameters

- **method** (*str*) – The HTTP method name, e.g. "GET", "PUT", etc.
- **path** (*Optional[str]*) – The HTTP path to be appended to the server attribute.
- **params** (*Optional[Dict]*) – A dict holding the HTTP query parameters.
- **headers** (*Optional[Dict]*) – A dict holding the HTTP request headers.

- **cookies** (*Optional[Dict]*) – A dict holding the HTTP request cookies.
- **json** (*Optional[Dict]*) – A JSON object (usually a dict) to be passed as request body with content-type `application/json`.
- **data** (*Optional[Dict]*) – A str to be passed as request body with content-type `application/x-www-form-urlencoded`.
- **proxies** (*Optional[Dict]*) – A dict holding the HTTP proxies to be used.

Returns The HTTP response returned by the `requests` package.

Raises ***ApiError*** – If the status code of the HTTP response is not in the interval [200, 300).

Return type requests.models.Response

get(***kwargs*)
Send a HTTP GET request.

Parameters **kwargs** – Keyword arguments passed when sending the HTTP request.

Returns The HTTP response.

Return type requests.models.Response

put(***kwargs*)
Send a HTTP PUT request.

Parameters **kwargs** – Keyword arguments passed when sending the HTTP request.

Returns The HTTP response.

Return type requests.models.Response

patch(***kwargs*)
Send a HTTP PATCH request.

Parameters **kwargs** – Keyword arguments passed when sending the HTTP request.

Returns The HTTP response.

Return type requests.models.Response

post(***kwargs*)
Send a HTTP POST request.

Parameters **kwargs** – Keyword arguments passed when sending the HTTP request.

Returns The HTTP response.

Return type requests.models.Response

delete(***kwargs*)
Send a HTTP DELETE request.

Parameters **kwargs** – Keyword arguments passed when sending the HTTP request.

Returns The HTTP response.

Return type requests.models.Response

class xyzspaces.apis.**ProjectApi**(*server='https://xyz.api.here.com'*, *credentials=None*)
Bases: *xyzspaces.apis.Api*

XYZ RESTful Project API abstraction.

Instances of this class allow to manage XYZ Hub projects.

API calls must be authenticated via a bearer token which needs to be provided as a `credentials` parameter when initialising an instance.

__init__ (*server='https://xyz.api.here.com'*, *credentials=None*)
Instantiate a *ProjectApi* object.

Parameters

- **server** (*Optional[str]*) – The URL of the server implementing the API.
- **credentials** (*Optional[str]*) – A string to serve as authentication (a bearer token). Will be looked up in environment variable `XYZ_TOKEN` if not provided.

get_projects (*paginate=None*, *handle=None*, *limit=None*)
List the projects either for the token owner or list the published projects.

Parameters

- **paginate** (*Optional[bool]*) – A Boolean telling if the results should be paginated (default: ?).
- **handle** (*Optional[int]*) – A string to be used when running the next request in a sequence of pagninated responses. Works only when `paginate` is `True`.
- **limit** (*Optional[int]*) – The max. number of projects to return. Works only when `paginate` is `True`.

Returns A JSON object containing information about the projects.

Return type dict

get_project (*project_id*)
Get the project by ID.

Parameters **project_id** (*str*) – A string representing the project ID.

Returns A JSON object with information about the requested project.

Return type dict

post_project (*data*)
Create a project.

Parameters **data** – A JSON object describing the project to be created.

Returns A JSON object with all information about the created project.

Return type dict

put_project (*project_id*, *data*)
Update a project by ID.

Update the project with the provided project ID and create it if it does not exist. This will replace the whole project definition.

Parameters

- **project_id** (*str*) – A string representing the desired project ID.
- **data** – A JSON object describing the project details to be updated.

Returns A JSON object with information about the updated project.

Return type dict

patch_project (*project_id*, *data*)
Update parts of a project by ID.

Parameters

- **project_id** (*str*) – A string representing the desired project ID.
- **data** – A JSON object describing the project parts to be updated.

Returns A JSON object with information about the updated project.

Return type dict

delete_project(*project_id*)
Delete a project by ID.

Parameters **project_id** (*str*) – A string representing the desired project ID.

Returns An empty string if the operation was successful.

Return type str

class xyzspaces.apis.**TokenApi**(*server='https://xyz.api.here.com'*, *credentials=None*)
Bases: *xyzspaces.apis.Api*

XYZ RESTful Token API abstraction.

Instances of this API class allow to manage HERE XYZ tokens.

API calls must be authenticated via authenticated access cookies derived from the username and password of an existing HERE developer account with access to HERE XYZ. These need to be provided as a `credentials` parameter when initialising an instance.

Example:

```
>>> from xyzspaces.apis import TokenApi
>>> api = TokenApi(credentials=dict(username="foo", password="bar"))
>>> api.get_tokens()
[...]
```

__init__(*server='https://xyz.api.here.com'*, *credentials=None*)
Instantiate a *TokenApi* object.

Parameters

- **server** (*Optional[str]*) – The URL of the server implementing the API.
- **credentials** (*Optional[Dict[str, str]]*) – A dict to hold the authentication details (the fields `username` and `password` with valid values for the user's existing HERE developer account). Will be looked up in environment variables `HERE_USER` and `HERE_PASSWORD` if not provided.

get_token(*token_id*, ***kwargs*)
Get info for given token ID.

Parameters

- **token_id** (*str*) – A string representing the requested token.
- **kwargs** – A dict with additional parameters passed to the HTTP request made when provided.

Returns A JSON object with the information about the requested token.

Return type dict

get_tokens(***kwargs*)
Get list of tokens for the authenticated user.

Parameters **kwargs** – A dict with additional parameters passed to the HTTP request made when provided.

Returns A list with information about all available tokens.

Return type list

post_token(*json={}*, ***kwargs*)
Create a new permanent or temporary token.

Parameters

- **json** (*Dict*) – A dict with information about the token to be created.
- **kwargs** – A dict with additional parameters passed to the HTTP request made when provided.

Returns A JSON object with all information about the created token.

Return type dict

delete_token(*token_id*, ***kwargs*)
Delete the token with the provided ID.

Parameters

- **token_id** (*str*) – A string representing a valid token.
- **kwargs** – A dict with additional parameters passed to the HTTP request made when provided.

Returns An empty string if the operation was successful.

Return type str

class xyzspaces.apis.**HubApi**(*server='https://xyz.api.here.com'*, *credentials=None*)
Bases: *xyzspaces.apis.Api*

XYZ RESTful Hub API abstraction.

Instances of this API class allow to manage HERE XYZ spaces.

API calls must be authenticated via a bearer token which needs to be provided as a `credentials` parameter when initialising an instance.

A few convenience methods for calling HTTP methods directly are inherited from the *Api* base class, and can also be used, although this class provides dedicated methods for the Hub API, like *HubApi.get_spaces()* starting with HTTP method names and an underscore, followed by a name resembling the respective API endpoint, in this case `GET /hub/spaces`.

Example:

```
>>> from xyzspaces.apis import HubApi
>>> api = HubApi(credentials="MY_XYZ_TOKEN")
>>> api.get_spaces()
[...]
```

This is based on the HERE XYZ Hub API specification defined here: https://xyz.api.here.com/hub/static/swagger/#/

__init__(*server='https://xyz.api.here.com'*, *credentials=None*)
Instantiate an *HubApi* object.

Parameters

- **server** (*Optional[str]*) – The URL of the server implementing the API.

- **credentials** (*Optional[str]*) – A string to serve as authentication (a bearer token). Will be looked up in environment variable `XYZ_TOKEN` if not provided.

get_hub(*params=None*)
Get basic information about the XYZ Hub.

Parameters **params** (*Optional[dict]*) – A dict holding the HTTP query parameters.

Returns A JSON object with hub information.

Return type dict

get_spaces(*params=None*)
Get Spaces information.

Parameters **params** (*Optional[dict]*) – A dict holding the HTTP query parameters.

Returns A JSON object with list of spaces.

Return type dict

get_space(*space_id*, *params=None*)
Get a space by ID.

Parameters

- **space_id** (*str*) – The desired space ID.
- **params** (*Optional[dict]*) – A dict for query params.

Returns A JSON object with information about the space_id.

Return type dict

post_space(*data*)
Create a space.

Parameters **data** (*dict*) – A dict describing the space to be created.

Returns A JSON object with all information about the created space.

Return type dict

patch_space(*space_id*, *data*)
Update a space.

Parameters

- **space_id** (*str*) – A string representing the desired space ID.
- **data** (*dict*) – A JSON object describing the space attributes to be updated.

Returns A JSON object with information about the updated space.

Return type dict

delete_space(*space_id*)
Delete a space.

Parameters **space_id** (*str*) – A string representing desired space ID.

Returns An empty string if the operation was successful.

Return type str

get_space_features(*space_id*, *feature_ids*, *force_2d=None*)
Get features by ID.

Parameters

- **space_id** (*str*) – The desired space ID.
- **feature_ids** (*List[str]*) – A list of feature_ids.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Returns A feature collection with all features inside the specified space.

Return type dict

Example: Get single feature from space

```
>>> feats = api.get_space_features(
...         space_id=space_id, feature_ids=["GER", "BRA"])
>>> print(feats)
```

get_space_feature(*space_id*, *feature_id*, *force_2d=None*)

Get a feature by ID.

Parameters

- **space_id** (*str*) – The desired space ID.
- **feature_id** (*str*) – The desired feature ID.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Returns A feature with the specified feature ID inside the space with the specified ID.

Return type dict

Example: Read the feature from the space.

```
>>>     feature = api.get_space_feature(
...     space_id=space_id, feature_id=feature_id)
>>>     print(json.dumps(feature, indent=4, sort_keys=True))
```

get_space_statistics(*space_id*)

Get statistics.

Parameters **space_id** (*str*) – The desired space ID.

Returns A JSON object with some statistics about the specified space.

Return type dict

Example:

```
>>>     stats = api.get_space_statistics(space_id=space_id)
>>>     print(json.dumps(stats, indent=4, sort_keys=True))
```

get_space_bbox(*space_id*, *bbox*, *tags=None*, *clip=None*, *limit=None*, *params=None*, *selection=None*, *skip_cache=None*, *clustering=None*, *clusteringParams=None*, *force_2d=None*)

Get features inside some given bounding box.

Parameters

- **space_id** (*str*) – A string with the ID of the desired XYZ space.
- **bbox** (*List[Union[int, float]]*) – A list of four numbers representing the West, South, East and North margins, respectively, of the bounding box.

- **tags** (*Optional[List[str]]*) – A list of strings holding tag values.
- **clip** (*Optional[bool]*) – A Boolean indicating if the result should be clipped (default: False).
- **limit** (*Optional[int]*) – A max. number of features to return in the result.
- **params** (*Optional[dict]*) – ...
- **selection** (*Optional[List[str]]*) – ...
- **skip_cache** (*Optional[bool]*) – ...
- **clustering** (*Optional[str]*) – ...
- **clusteringParams** (*Optional[dict]*) – ...
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Returns A dict representing a feature collection.

Return type dict

Example:

```
>>>     bb = [0, 0, 20, 20]
>>>     bbox = api.get_space_bbox(space_id=space_id, bbox=bb)
>>>     print(len(bbox["features"]))
>>>     print(bbox["type"])
```

get_space_tile(*space_id*, *tile_type*, *tile_id*, *tags=None*, *clip=None*, *params=None*, *selection=None*, *skip_cache=None*, *clustering=None*, *clusteringParams=None*, *margin=None*, *limit=None*, *force_2d=None*, *mode=None*, *viz_sampling=None*)

Get features in tile.

Parameters

- **space_id** (*str*) – A string with the ID of the desired XYZ space.
- **tile_type** (*str*) – A string with the name of a tile type, one of "quadkeys", "web", "tms" or "here". See below.
- **tile_id** (*str*) – A string holding a valid tile ID according to the specified `tile_type`.
- **tags** (*Optional[List[str]]*) – A list of strings holding tag values.
- **clip** (*Optional[bool]*) – A Boolean indicating if the result should be clipped (default: False).
- **margin** (*Optional[int]*) – ...
- **limit** (*Optional[int]*) – A max. number of features to return in the result.
- **params** (*Optional[dict]*) – ...
- **selection** (*Optional[List[str]]*) – ...
- **skip_cache** (*Optional[bool]*) – ...
- **clustering** (*Optional[str]*) – ...
- **clusteringParams** (*Optional[dict]*) – ...
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

- **mode** (*Optional[str]*) – A string to indicate how to optimize the resultset and geometries for display. Allowed values are `raw` and `viz`.
- **viz_sampling** (*Optional[str]*) – A string to indicate the sampling strength in case of `mode=viz`. Allowed values are: `low`, `med`, `high`, and `off`, default: `med`.

Returns A dict representing a feature collection.

Return type dict

Available tile types are:

- quadkeys, Virtual Earth Tile System,
- web, Tiled Web Map.
- tms, OSGEO Tile Map Service,
- here, ?

get_space_search(*space_id*, *tags=None*, *limit=None*, *params=None*, *selection=None*, *skip_cache=None*, *force_2d=None*)

Search for features.

Parameters

- **space_id** (*str*) – A string with the ID of the desired XYZ space.
- **tags** (*Optional[List[str]]*) – A list of strings holding tag values.
- **limit** (*Optional[int]*) – A max. number of features to return in the result.
- **params** (*Optional[dict]*) – ...
- **selection** (*Optional[List[str]]*) – ...
- **skip_cache** (*Optional[bool]*) – ...
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Returns A dict representing a feature collection.

Return type dict

Example:

```
>>>     feats = api.get_space_search(space_id=space_id)
>>>     print(feats["type"] )
>>>     print(len(feats["features"]) )
```

get_space_iterate(*space_id*, *limit*, *force_2d=None*)

Iterate features in the space (yielding them one by one).

Parameters

- **space_id** (*str*) – A string representing desired space ID.
- **limit** (*int*) – A max. number of features to return in the result.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Yields A feature in space.

Return type Generator

get_space_all(*space_id*, *limit*, *max_len=1000*)
Get all features as one single GeoJSON feature collection.

This is a convenience method, not directly available in the XYZ API. It hides the API paging mechanism and returns all data in one chunk. So be careful if you don't know how much data you will get.

Parameters

- **space_id** (*str*) – A string representing desired space ID.
- **limit** (*int*) – A max. number of features to return in the result.
- **max_len** – A max. number of features to return in the result.

Returns A dict representing a feature collection.

Return type dict

Example:

```
>>>     fc = api.get_space_all(space_id=space_id, limit=100)
>>>     print(len(fc["features"]) )
>>>     print(fc["type"])
```

get_space_count(*space_id*)
Get feature count.

Parameters **space_id** (*str*) – A string with the ID of the desired XYZ space.

Returns A dict containing the number of features inside the specified space.

Return type dict

put_space_features(*space_id*, *data*, *add_tags=None*, *remove_tags=None*)
Create or replace multiple features.

Parameters

- **space_id** (*str*) – A string with the ID of the desired XYZ space.
- **data** (*dict*) – A JSON object describing one or more features to add.
- **add_tags** (*Optional[List[str]]*) – A list of strings describing tags to be added to the features.
- **remove_tags** (*Optional[List[str]]*) – A list of strings describing tags to be removed from the features.

Returns A dict representing a feature collection.

Return type dict

Example:

```
>>> from xyzspaces.datasets import get_countries_data
>>>     gj_countries = get_countries_data()
>>>     features = api.put_space_features(
...     space_id=space_id,
...     data=gj_countries,
...     add_tags=["foo", "bar"],
...     remove_tags=["bar"],
... )
>>> print(features)
```

post_space_features(*space_id*, *data*, *add_tags=None*, *remove_tags=None*)
Modify multiple features in the space.

Parameters

- **space_id** (*str*) – A string with the ID of the desired XYZ space.
- **data** (*dict*) – A JSON object describing one or more features to modify.
- **add_tags** (*Optional[List[str]]*) – A list of strings describing tags to be added to the features.
- **remove_tags** (*Optional[List[str]]*) – A list of strings describing tags to be removed from the features.

Returns A dict representing a feature collection.

Return type dict

Example:

```
>>> data = dict(type="FeatureCollection", features=[deu, ita])
>>> space_features = api.post_space_features(
...     space_id=space_id, data=data)
>>> print(space_features)
```

delete_space_features(*space_id*, *id=None*, *tags=None*)
Delete multiple features from the space.

Parameters

- **space_id** (*str*) – A string with the ID of the desired XYZ space.
- **id** (*Optional[List[str]]*) – A list of feature IDs to delete.
- **tags** (*Optional[List[str]]*) – A list of strings describing tags the features to be deleted must have.

Returns A response from API call.

Return type str

Example:

```
>>>     deu = api.get_space_feature(space_id=space_id, feature_id="DEU")
>>>     ita = api.get_space_feature(space_id=space_id, feature_id="ITA")
>>>     deleted_features = api.delete_space_features(
...     space_id=space_id, id=["DEU", "ITA"])  # noqa: E501
```

put_space_feature(*space_id*, *data*, *feature_id=None*, *add_tags=None*, *remove_tags=None*)
Create or replace a single feature.

Parameters

- **space_id** (*str*) – A string with the ID of the desired XYZ space.
- **data** (*dict*) – A JSON object describing the feature to be added.
- **feature_id** (*Optional[str]*) – A string with the ID of the feature to be created.
- **add_tags** (*Optional[List[str]]*) – A list of strings describing tags to be added to the feature.
- **remove_tags** (*Optional[List[str]]*) – A list of strings describing tags to be removed from the feature.

Returns A dict representing a feature.

Return type dict

Example:

```
>>>     api.put_space_feature(
...     space_id=space_id, feature_id=feature_id, data=fra)
```

patch_space_feature(*space_id*, *feature_id*, *data*, *add_tags=None*, *remove_tags=None*)

Patch a single feature in the space.

Parameters

- **space_id** (*str*) – A string with the ID of the desired XYZ space.
- **feature_id** (*str*) – A string with the ID of the feature to be modified.
- **data** (*dict*) – A JSON object describing the feature to be changed.
- **add_tags** (*Optional[List[str]]*) – A list of strings describing tags to be added to the feature.
- **remove_tags** (*Optional[List[str]]*) – A list of strings describing tags to be removed from the feature.

Returns A dict representing a feature.

Return type dict

delete_space_feature(*space_id*, *feature_id*)

Delete a single feature from the space.

Parameters

- **space_id** (*str*) – A string with the ID of the desired XYZ space.
- **feature_id** (*str*) – A string with the ID of the feature to be deleted.

Returns An empty string if the operation was successful.

Return type str

get_space_spatial(*space_id*, *lat=None*, *lon=None*, *ref_space_id=None*, *ref_feature_id=None*, *radius=None*, *tags=None*, *limit=None*, *params=None*, *selection=None*, *skip_cache=None*, *force_2d=None*)

Get features with radius search.

Parameters

- **space_id** (*str*) – A string with the ID of the desired XYZ space.
- **lat** (*Optional[float]*) – A float in WGS'84 decimal degree (-90 to +90) of the center Point.
- **lon** (*Optional[float]*) – A float in WGS'84 decimal degree (-180 to +180) of the center Point.
- **ref_space_id** (*Optional[str]*) – A string as alternative for defining center coordinates, it is possible to reference a geometry in a space, hence it is needed to provide the `ref_space_id` where the referenced feature is stored. Always to use in combination with `ref_feature_id`.
- **ref_feature_id** (*Optional[str]*) – A string as unique identifier of a feature in the referenced space. The geometry of that feature gets used for the spatial query. Always to use in combination with `ref_space_id`.

- **radius** (*Optional[int]*) – An int in meter which defines the diameter of the search request.
- **tags** (*Optional[List[str]]*) – A list of strings holding tag values.
- **limit** (*Optional[int]*) – A max. number of features to return in the result.
- **params** (*Optional[dict]*) – A dict holding the HTTP query parameters.
- **selection** (*Optional[List[str]]*) – A list of strings holding properties values.
- **skip_cache** (*Optional[bool]*) – A Boolean if set to `True` the response is not returned from cache.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Returns A dict representing a feature collection.

Raises **ValueError** – If incorrect params are passed, either `lat` and `lon` or `ref_space_id` and `ref_feature_id` must have a value.

Return type dict

post_space_spatial(*space_id*, *data*, *radius=None*, *tags=None*, *limit=None*, *params=None*, *selection=None*, *skip_cache=None*, *force_2d=None*)

Post features which intersect the provided geometry.

Parameters

- **space_id** (*str*) – A string with the ID of the desired XYZ space.
- **data** (*dict*) – A JSON object which is getting used for the spatial search.
- **radius** (*Optional[int]*) – An int which defines the diameter(meters) of the search request.
- **tags** (*Optional[List[str]]*) – A list of strings holding tag values.
- **limit** (*Optional[int]*) – A max. number of features to return in the result.
- **params** (*Optional[dict]*) – A dict holding the HTTP query parameters.
- **selection** (*Optional[List[str]]*) – A list of strings holding properties values.
- **skip_cache** (*Optional[bool]*) – A Boolean if set to `True` the response is not returned from cache.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Returns A dict representing a feature collection.

Return type dict

4.2.3 xyzspaces.auth module

This module provides HERE authentication via cookies for the Token API.

This *get_auth_cookies()* function simulates the login process on http://developer.here.com to obtain an access cookie for authenticating with certain RESTful APIs like the XYZ Token API.

This implementation is inspired by the Open Source HERE XYZ CLI: https://github.com/heremaps/here-cli/blob/master/src/sso.ts.

xyzspaces.auth.**filter_cookies**(*cookies*, *prefix*)

Filter requests cookies with some given name prefix into a new dict.

Parameters

- **cookies** (*requests.cookies.RequestsCookieJar*) – A requests cookies object.
- **prefix** (*str*) – A prefix string to search in cookies.

Returns A dict.

Return type dict

Example:

Input:

```
<RequestsCookieJar[
    <Cookie locale=en-US for .here.com/>,
    <Cookie here_account=foobar for account.here.com/>,
    <Cookie here_account.sig=barfoo for account.here.com/>]
>
```

Output:

```
{'here_account': 'foobar', 'here_account.sig': 'barfoo'}
```

xyzspaces.auth.**get_auth_cookies**(*username*, *password*)

Get authentication cookies from name and password of a HERE account.

Parameters

- **username** (*str*) – Username for HERE account.
- **password** (*str*) – Password for HERE account.

Returns A dict.

Raises ***AuthenticationError*** – If status_code for HTTP response returned by requests is not equal to 200.

Return type dict

4.2.4 xyzspaces.constants module

This module defines project level constants.

4.2.5 xyzspaces.curl module

This module contains functionality related to `curl` commands.

It provides methods for creating/executing `curl` commands which mimic the `requests` signatures.

This module has been mainly designed to be used in the `Api` module for logging purposes.

The `curl` module could be also used as stand alone:

Example:

```
>>> import xyzspaces.curl as curl
>>> command = curl.get(url='https://xkcd.com/552/info.0.json')
['curl', '--request', 'GET', 'https://xkcd.com/552/info.0.json']
>>> curl.execute(command)
b'{"month": "3", "num": 552, "link": "", "year": "2009", "news": "", "safe_title":
↪"Correlation", "transcript": "[[A man is talking to a woman]]\\nMan: I used to␣
↪think correlation implied causation.\\nMan: Then I took a statistics class.  Now I␣
↪don\'t.\\nWoman: Sounds like the class helped.\\nMan: Well, maybe.\\n{{Title text:␣
↪Correlation doesn\'t imply causation, but it does waggle its eyebrows suggestively␣
↪and gesture furtively while mouthing \'look over there\'.}}", "alt": "Correlation␣
↪doesn\'t imply causation, but it does waggle its eyebrows suggestively and gesture␣
↪furtively while mouthing \'look over there\'.", "img": "https://imgs.xkcd.com/
↪comics/correlation.png", "title": "Correlation", "day": "6"}'  # noqa: E501
[...]
```

`xyzspaces.curl.`**`get`**(*url*, *params=None*, ***kwargs*)

Run `curl GET` mimicking the `requests.get()` signature.

This completes a `curl GET` command.

Parameters

- **`url`** – The URL of the server including the path.
- **`params`** – The query string params.
- **`kwargs`** – Further keyword arguments that should match those expected by `requests`.

Returns The `List[str]` representation of the `curl GET` command.

Return type List[str]

Example:

```
>>> import xyzspaces.curl as curl
>>> curl.get(url="https://xkcd.com/552/info.0.json")
['curl', '--request', 'GET', 'ttps://xkcd.com/552/info.0.json']
```

`xyzspaces.curl.`**`put`**(*url*, *data=None*, ***kwargs*)

Run `curl PUT` command mimicking the `requests.put()` signature.

This completes a `curl PUT` command.

Parameters

- **`url`** – The URL of the server including the path for the new object.

- **data** – (optional) Dictionary, list of tuples, bytes, or file-like object.
- **kwargs** – Further keyword arguments that should match those expected by `requests`.

Returns The `List[str]` representation of the `curl PUT` command.

Return type List[str]

`xyzspaces.curl.`**`patch`**(*url*, *data=None*, ***kwargs*)

Run `curl PATCH` command mimicking the `requests.patch()` signature.

This completes a `curl PATCH` command.

Parameters

- **url** – The URL of the server including the path for the new object.
- **data** – (optional) Dictionary, list of tuples, bytes, or file-like object.
- **kwargs** – Further keyword arguments that should match those expected by `requests`.

Returns The `List[str]` representation of the `curl PATCH` command.

Return type List[str]

`xyzspaces.curl.`**`post`**(*url*, *data=None*, *json=None*, ***kwargs*)

Run `curl POST` command mimicking the `requests.post()` signature.

This completes a `curl POST` command.

Parameters

- **url** – The URL of the server including the path for the new object.
- **data** – (optional) Dictionary, list of tuples, bytes, or file-like object.
- **json** – (optional) json data.
- **kwargs** – Further keyword arguments that should match those expected by `requests`.

Returns The `List[str]` representation of the `curl POST` command.

Return type List[str]

`xyzspaces.curl.`**`delete`**(*url*, ***kwargs*)

Run `curl DELETE` command mimicking the `requests.delete()` signature.

This completes a `curl DELETE` command.

Parameters

- **url** – The URL of the server including the path.
- **kwargs** – Further keyword arguments that should match those expected by `requests`.

Returns The `List[str]` representation of the `curl DELETE` command.

Return type List[str]

`xyzspaces.curl.`**`command`**(*url*, *method*, *params=None*, ***kwargs*)

Return a `curl` command equivalent from the params for `requests`.

This builds and returns a list of strings representing a `curl` command that can be directly passed to functions like `subprocess.check_output()`. When joined with blanks into a single string it can also be used for logging or pasting to a terminal.

To be used like `requests.get()`, passing the same params for headers, cookies, etc. So the function signature is similar to `requests.get()`, with additional `method` parameter.

Parameters

- **url** (*str*) – The URL of the server including the path.
- **method** (*str*) – The HTTP method name, e.g. “GET”, “PUT”, etc.
- **params** (*Optional[dict]*) – The query string params.
- **kwargs** (*Optional[Mapping]*) – Further keyword arguments that should match those expected by `requests`.

Returns The generated `curl` command (a list of strings).

Return type List[str]

Example:

```
>>> import xyzspaces.curl as curl
>>> curl.command(method="get", url="https://xkcd.com/552/info.0.json")
['curl', '--request', 'GET', 'https://xkcd.com/552/info.0.json']
```

`xyzspaces.curl.`**execute**(*command*)

Execute a `command` and create the `requests.models.Response` object.

The Python’s `subprocess` module will be used for the executing a `command`.

In the `requests.models.Response` object will be initialized following attributes: _content: The response data from the `stdout` status_code: Simplified conversion from the `subprocess.CompletedProcess.returncode` to `HTTP` ones 200 ~ 0, 500 ~ >0.

Parameters **command** (*List[str]*) – The `curl` to be executed in the `List[str]` type.

Returns The generated `requests.models.Response` object.

Return type requests.models.Response

Example:

```
>>> import xyzspaces.curl as curl
>>> command = curl.get(url='https://xkcd.com/552/info.0.json')
['curl', '--request', 'GET', 'ttps://xkcd.com/552/info.0.json']
>>> curl.execute(command)
b'{"month": "3", "num": 552, "link": "", "year": "2009", "news": "", "safe_title
↪": "Correlation", "transcript": "[[A man is talking to a woman]]\\nMan: I used␣
↪to think correlation implied causation.\\nMan: Then I took a statistics class. ␣
↪Now I don\'t.\\nWoman: Sounds like the class helped.\\nMan: Well, maybe.\\n{
↪{Title text: Correlation doesn\'t imply causation, but it does waggle its␣
↪eyebrows suggestively and gesture furtively while mouthing \'look over there\'.}
↪}", "alt": "Correlation doesn\'t imply causation, but it does waggle its␣
↪eyebrows suggestively and gesture furtively while mouthing \'look over there\'.
↪", "img": "https://imgs.xkcd.com/comics/correlation.png", "title": "Correlation
↪", "day": "6"}'  # noqa: E501
[...]
```

4.2.6 xyzspaces.exceptions module

This module defines API exceptions.

exception xyzspaces.exceptions.**AuthenticationError**
Bases: Exception

Exception raised when authentication fails.

exception xyzspaces.exceptions.**ApiError**
Bases: Exception

Exception raised for API HTTP response status codes not in [200...300).

The exception value will be the response object returned by `requests` which provides access to all its attributes, eg. `status_code`, `reason` and `text`, etc.

Example:

```
>>> try:
>>>     api = HubApi(credentials="MY-XYZ-TOKEN")
>>>     api.get("/hub/nope").json()
>>> except ApiError as e:
>>>     resp = e.value.args[0]
>>>     if resp.status_code == 404 and resp.reason == "Not Found":
>>>         ...
```

__str__()
Return a string from the HTTP response causing the exception.

The string simply lists the repsonse's status code, reason and text content, separated with commas.

exception xyzspaces.exceptions.**TooManyRequestsException**
Bases: Exception

Exception raised for API HTTP response status code 429.

This is a dedicated exception to be used with the *backoff* package, because it requires a specific exception class.

The exception value will be the response object returned by `requests` which provides access to all its attributes, eg. `status_code`, `reason` and `text`, etc.

__str__()
Return a string from the HTTP response causing the exception.

The string simply lists the repsonse's status code, reason and text content, separated with commas.

4.2.7 xyzspaces.logconf module

This module configures logging.

xyzspaces.logconf.**setup_logging**(*default_path='config/logconfig.json'*, *default_level=40*, *env_key='XYZ_LOG_CONFIG'*)
Set up logging configuration.

Parameters

- **default_path** (*str*) – A string representing the path of the config file in JSON format.
- **default_level** (*int*) – An int representing logging level.
- **env_key** (*str*) – A string representing environment variable to enable logging to file.

class xyzspaces.logconf.**NullHandler**(*level=0*)
Bases: logging.Handler

NullHandler class is a 'no-op' handler for use by library developers.

emit(*record*)
Skip the emit record. This is used to give preference to user.

4.2.8 xyzspaces.spaces module

An more Pythonic abstraction for XYZ Spaces.

This contains only one class for an XYZ "space" which in turn provides access to "features". There is no separate class abstraction for single features, but they are taken to be valid geojson.GeoJSON objects. Various other bits of the XYZ Hub API are simply returned as-is, usually wrapped in dictionaries, like the "statistics" of some given XYZ space.

class xyzspaces.spaces.**Space**(*api=None*, *server='https://xyz.api.here.com'*)
Bases: object

An abstraction for XYZ Spaces.

A space object is created with an existing, authenticated XYZ Hub API instance.

Example:

```
>>> space = Space.new(...)
>>> space.delete()
>>> for feat in space.iter_feature():
....   print(feat["id"])
```

classmethod from_id(*space_id*, *server='https://xyz.api.here.com'*)
Instantiate a space object for an existing space ID.

Parameters

- **space_id** (*str*) – A string to represent the id of the space.
- **server** (*str*) – A string as base URL for Data Hub APIs. Required only if Data Hub APIs are self-hosted.

Returns An object of *Space*.

Return type *xyzspaces.spaces.Space*

classmethod new(*title*, *description=None*, *space_id=None*, *schema=None*, *enable_uuid=None*, *listeners=None*, *shared=None*, *server='https://xyz.api.here.com'*)
Create new space object with given title and description.

Optionally, the desired space ID can be provided as well, and will be used if still available.

Parameters

- **title** (*str*) – A string representing the title of the space.
- **description** (*Optional[str]*) – A string representing a description of the space.
- **space_id** (*Optional[str]*) – A string representing space_id.
- **schema** (*Optional[str]*) – JSON object or URL to be added as a schema for space.
- **enable_uuid** (*Optional[bool]*) – A boolean if set True it will create additional activity log space to log the activities in current space.

- **listeners** (*Optional[Dict[str, Union[int, str]]]*) – A dict for activity log listener params.
- **shared** (*Optional[bool]*) – A boolean, if set to `True`, space will be shared with other users having XYZ account, they will be able to read from the space using their own token. By default space will not be a shared space.
- **server** (*str*) – A string as base URL for Data Hub APIs. Required only if Data Hub APIs are self-hosted.

Returns A object of *Space*.

Return type *xyzspaces.spaces.Space*

classmethod virtual(*title*, *description=None*, *server='https://xyz.api.here.com'*, ***kwargs*)

Create a new virtual-space.

Virtual-space references one or multiple spaces. A virtual-space is described by definition which references other existing spaces (the upstream spaces). Queries being done to a virtual-space will return the features of its upstream spaces combined. There are different predefined operations of how to combine the features of the upstream spaces. In order to use virtual spaces feature user needs to have HERE Data Hub paid plan. Plans can be found here: HERE Data Hub.

Parameters

- **title** (*str*) – A string representing the title of the virtual-space.
- **description** (*Optional[str]*) – A string representing a description of the virtual-space.
- **server** (*str*) – A string as base URL for Data Hub APIs. Required only if Data Hub APIs are self-hosted.
- **kwargs** (*Dict[str, Dict]*) – A dict for the operation to perform on upstream spaces.

Returns An object of *Space*.

Return type *xyzspaces.spaces.Space*

__init__(*api=None*, *server='https://xyz.api.here.com'*)

Instantiate a space object, optionally with authenticated api instance and custom base URL as `server`, `server` is required only for self-hosted Data Hub instances.

Parameters

- **api** (*Optional[xyzspaces.apis.HubApi]*) –
- **server** (*str*) –

__repr__()

Return string representation of this instance.

property info

Space config information.

list(*owner='me'*, *include_rights=False*)

Return list of spaces for given owner with access rights if desired.

This does not modify the space object itself.

Parameters

- **owner** (*str*) – A string representing the owner.

- **include_rights** (*bool*) – A boolean. If set to True, the access rights for each space are included in the response.

Returns A JSON object with list of spaces.

Return type Dict

read(*id*)
Read existing space object for given space ID.

Parameters **id** (*str*) –

Return type *xyzspaces.spaces.Space*

update(*title=None*, *description=None*, *tagging_rules=None*, *schema=None*, *shared=None*)
Update space attributes.

This method updates title, description, schema, shared status or tagging rules of space, at least one of these params should have a non-default value to update the space.

Does update the space in the XYZ storage and this object mirroring it. Also apply the tags based on rules mentioned in *tagging_rules* dict.

Parameters

- **title** (*Optional[str]*) – A string representing the title of the space.
- **description** (*Optional[str]*) – A string representing a description of the space.
- **tagging_rules** (*Optional[Dict[str, str]]*) – A dict where the key is the tag to be applied to all features matching the JSON-path expression being the value.
- **schema** (*Optional[str]*) – JSON object or URL to be added as schema for space.
- **shared** (*Optional[bool]*) – A boolean, if set to `True`, space will be shared with other users having XYZ account, they will be able to read from the space using their own token. If set to `False` space will be unshared.

Returns A response from API.

Return type Dict

Example:

```
>>> from xyzspaces import XYZ
>>> xyz = XYZ(credentials="XYZ_TOKEN")
>>> space = xyz.spaces.new(title="new space", description="new space")
>>> tagging_rules = {"large": "$.features[?(@.properties.area>=500)]"}
>>> space.update(title="updated title",
...              description="updated description",
...              tagging_rules=tagging_rules)
```

delete()
Delete this space object.

get_statistics()
Get statistics for this space object.

Returns A JSON object with some statistics about the specified space.

Return type dict

search(*tags=None*, *limit=None*, *params=None*, *selection=None*, *skip_cache=None*, *geo_dataframe=None*, *force_2d=None*)
Search features for this space object.

Parameters

- **tags** (*Optional[List[str]]*) – A list of strings holding tag values.
- **limit** (*Optional[int]*) – A max. number of features to return in the result.
- **params** (*Optional[dict]*) – A dict to represent additional filter on features to be searched. Examples:
  - `params={"p.name": "foo"}` returns all features with a value of property `name` equal to `foo`.
  - `params={"p.name!": "foo"}` returns all features with a value of property `name` not qual to `foo`.
  - `params={"p.count=gte": "10"}` returns all features with a value of property `count` greater than or equal to `10`.
  - `params={"p.count=lte": "10"}` returns all features with a value of property `count` less than or equal to `10`.
  - `params={"p.count=gt": "10"}` returns all features with a value of property `count` greater than `10`.
  - `params={"p.count=lt": "10"}` returns all features with a value of property `count` less than `10`.
  - `params={"p.name=cs": "bar"}` returns all features with a value of property `name` which contains `bar`.
- **selection** (*Optional[List[str]]*) – A list, only these properties will be returned in features result list.
- **skip_cache** (*Optional[bool]*) – If set to `True` the response is not returned from cache. Default is `False`.
- **geo_dataframe** (*Optional[bool]*) – A boolean if set to `True` searched features will be yield as single Geopandas Dataframe.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Yields A Feature object by default. If param `geo_dataframe` is True then yields single Geopandas Dataframe.

Return type Generator[geojson.feature.Feature, None, None]

iter_feature(*limit=100*, *force_2d=None*)

Iterate over features in this space object.

Parameters

- **limit** (*int*) – A max. number of features to return in the result.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Yields A Feature object.

Return type Generator[geojson.feature.Feature, None, None]

get_feature(*feature_id*, *force_2d=None*)

Retrieve one GeoJSON feature with given ID from this space.

Parameters

- **feature_id** (*str*) – Feature id which is to fetched.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Returns A GeoJSON representing a feature with the specified feature ID inside the space.

Return type geojson.base.GeoJSON

add_feature(*data*, *feature_id=None*, *add_tags=None*, *remove_tags=None*)
Add one GeoJSON feature with given ID in this space.

Parameters

- **data** (*Union[geojson.base.GeoJSON, Dict]*) – A JSON object describing the feature to be added.
- **feature_id** (*Optional[str]*) – A string with the ID of the feature to be created.
- **add_tags** (*Optional[List[str]]*) – A list of strings describing tags to be added to the feature.
- **remove_tags** (*Optional[List[str]]*) – A list of strings describing tags to be removed from the feature.

Returns A GeoJSON representing a feature.

Return type geojson.base.GeoJSON

update_feature(*feature_id*, *data*, *add_tags=None*, *remove_tags=None*)
Update one GeoJSON feature with given ID in this space.

Parameters

- **feature_id** (*str*) – A string with the ID of the feature to be modified.
- **data** (*dict*) – A JSON object describing the feature to be changed.
- **add_tags** (*Optional[List[str]]*) – A list of strings describing tags to be added to the feature.
- **remove_tags** (*Optional[List[str]]*) – A list of strings describing tags to be removed from the feature.

Returns A GeoJSON representing a feature.

Return type geojson.base.GeoJSON

delete_feature(*feature_id*)
Delete one GeoJSON feature with given ID in this space.

Parameters **feature_id** (*str*) – A string with the ID of the feature to be deleted.

Returns An empty string if the operation was successful.

get_features(*feature_ids*, *geo_dataframe=None*, *force_2d=None*)
Retrieve one GeoJSON feature with given ID from this space.

Parameters

- **feature_ids** (*List[str]*) – A list of feature_ids.
- **geo_dataframe** (*Optional[bool]*) – A boolean if set to `True` features will be returned as single Geopandas Dataframe.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Returns A feature collection with all features inside the specified space. If param `geo_dataframe` is set to `True` then return features in single Geopandas Dataframe.

Return type Union[geojson.base.GeoJSON, geopandas.geodataframe.GeoDataFrame]

add_features(*features*, *add_tags=None*, *remove_tags=None*, *features_size=2000*, *chunk_size=1*, *id_properties=None*, *mutate=True*)

Add GeoJSON features to this space.

As API has a limitation on the size of features, features are divided into chunks, and multiple processes will process those chunks. Each chunk has a number of features based on the value of `features_size`. Each process handles chunks based on the value of `chunk_size`.

Parameters

- **features** (*Union[geojson.base.GeoJSON, Dict]*) – A JSON object describing one or more features to add.
- **add_tags** (*Optional[List[str]]*) – A list of strings describing tags to be added to the features.
- **remove_tags** (*Optional[List[str]]*) – A list of strings describing tags to be removed from the features.
- **features_size** (*int*) – An int representing a number of features to upload at a time.
- **chunk_size** (*int*) – Number of chunks each process to handle. The default value is 1, for a large number of features please use *chunk_size* greater than 1 to get better results in terms of performance.
- **id_properties** (*Optional[List[str]]*) – List of properties name from which id to be generated if id does not exists for a feature.
- **mutate** (*Optional[bool]*) – If True will update the existing features object passed, this will prevent making copy of the features object which may help to improving performance.

Returns A GeoJSON representing a feature collection.

Return type geojson.base.GeoJSON

_upload_features(*features*, *ids_map*, *add_tags=None*, *remove_tags=None*, *id_properties=None*)

Parameters

- **add_tags** (*Optional[List[str]]*) –
- **remove_tags** (*Optional[List[str]]*) –
- **id_properties** (*Optional[List[str]]*) –

_process_features(*features*, *id_properties*, *ids_map*)

_gen_id_from_properties(*feature*, *id_properties*)

update_features(*features*, *add_tags=None*, *remove_tags=None*)

Update GeoJSON features in this space.

Parameters

- **features** (*geojson.base.GeoJSON*) – A JSON object describing one or more features to modify.
- **add_tags** (*Optional[List[str]]*) – A list of strings describing tags to be added to the features.

- **remove_tags** (*Optional[List[str]]*) – A list of strings describing tags to be removed from the features.

Returns A GeoJSON representing a feature collection.

Return type geojson.base.GeoJSON

delete_features(*feature_ids*, *tags=None*)
Delete GeoJSON features in this space.

Parameters

- **feature_ids** (*List[str]*) – A list of feature IDs to delete.
- **tags** (*Optional[List[str]]*) – A list of strings describing tags the features to be deleted must have.

Returns A response from API.

features_in_bbox(*bbox*, *tags=None*, *clip=None*, *limit=None*, *params=None*, *selection=None*, *skip_cache=None*, *clustering=None*, *clustering_params=None*, *geo_dataframe=None*, *force_2d=None*)
Get features inside some given bounding box.

Parameters

- **bbox** (*List[Union[int, float]]*) – A list of four numbers representing the West, South, East and North margins, respectively, of the bounding box.
- **tags** (*Optional[List[str]]*) – A list of strings holding tag values.
- **clip** (*Optional[bool]*) – A Boolean indicating if the result should be clipped (default: False).
- **limit** (*Optional[int]*) – A max. number of features to return in the result.
- **params** (*Optional[dict]*) – A dict to represent additional filter on features to be searched. Examples:
  - `params={"p.name": "foo"}` returns all features with a value of property `name` equal to `foo`.
  - `params={"p.name!": "foo"}` returns all features with a value of property `name` not qual to `foo`.
  - `params={"p.count=gte": "10"}` returns all features with a value of property `count` greater than or equal to `10`.
  - `params={"p.count=lte": "10"}` returns all features with a value of property `count` less than or equal to `10`.
  - `params={"p.count=gt": "10"}` returns all features with a value of property `count` greater than `10`.
  - `params={"p.count=lt": "10"}` returns all features with a value of property `count` less than `10`.
  - `params={"p.name=cs": "bar"}` returns all features with a value of property `name` which contains `bar`.
- **selection** (*Optional[List[str]]*) – A list, only these properties will be returned in features result list.
- **skip_cache** (*Optional[bool]*) – If set to `True` the response is not returned from cache. Default is `False`.

- **clustering** (*Optional[str]*) – ...
- **clustering_params** (*Optional[dict]*) – ...
- **geo_dataframe** (*Optional[bool]*) – A boolean if set to `True` searched features will be yield as single Geopandas Dataframe.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Yields A Feature object by default. If param `geo_dataframe` is True then yields single Geopandas Dataframe.

Return type Generator[geojson.feature.Feature, None, None]

features_in_tile(*tile_type*, *tile_id*, *tags=None*, *clip=None*, *params=None*, *selection=None*, *skip_cache=None*, *clustering=None*, *clustering_params=None*, *margin=None*, *limit=None*, *geo_dataframe=None*, *force_2d=None*, *mode=None*, *viz_sampling=None*)

Get features in tile.

Parameters

- **tile_type** (*str*) – A string with the name of a tile type, one of "quadkeys", "web", "tms" or "here". See below.
- **tile_id** (*str*) – A string holding a valid tile ID according to the specified `tile_type`.
- **tags** (*Optional[List[str]]*) – A list of strings holding tag values.
- **clip** (*Optional[bool]*) – A Boolean indicating if the result should be clipped (default: False).
- **params** (*Optional[dict]*) – A dict to represent additional filter on features to be searched. Examples:
  - `params={"p.name": "foo"}` returns all features with a value of property `name` equal to `foo`.
  - `params={"p.name!": "foo"}` returns all features with a value of property `name` not qual to `foo`.
  - `params={"p.count=gte": "10"}` returns all features with a value of property `count` greater than or equal to `10`.
  - `params={"p.count=lte": "10"}` returns all features with a value of property `count` less than or equal to `10`.
  - `params={"p.count=gt": "10"}` returns all features with a value of property `count` greater than `10`.
  - `params={"p.count=lt": "10"}` returns all features with a value of property `count` less than `10`.
  - `params={"p.name=cs": "bar"}` returns all features with a value of property `name` which contains `bar`.
- **selection** (*Optional[List[str]]*) – A list, only these properties will be returned in features result list.
- **skip_cache** (*Optional[bool]*) – If set to `True` the response is not returned from cache. Default is `False`.
- **clustering** (*Optional[str]*) – ...
- **clustering_params** (*Optional[dict]*) – ...

- **margin** (*Optional[int]*) – ...
- **limit** (*Optional[int]*) – A max. number of features to return in the result.
- **geo_dataframe** (*Optional[bool]*) – A boolean if set to `True` searched features will be yield as single Geopandas Dataframe.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.
- **mode** (*Optional[str]*) – A string to indicate how to optimize the resultset and geometries for display. Allowed values are `raw` and `viz`.
- **viz_sampling** (*Optional[str]*) – A string to indicate the sampling strength in case of `mode=viz`. Allowed values are: `low`, `med`, `high`, and `off`, default: `med`.

Yields A Feature object by default. If param `geo_dataframe` is True then yields single Geopandas Dataframe.

Raises **ValueError** – If *tile_type* is invalid, valid tile_types are *quadkeys*, *web*, *tms* and *here*.

Return type Generator[geojson.feature.Feature, None, None]

spatial_search(*lat=None*, *lon=None*, *ref_space_id=None*, *ref_feature_id=None*, *radius=None*, *tags=None*, *limit=None*, *params=None*, *selection=None*, *skip_cache=None*, *geo_dataframe=None*, *force_2d=None*)

Get features with radius search.

Parameters

- **lat** (*Optional[float]*) – A float in WGS'84 decimal degree (-90 to +90) of the center Point.
- **lon** (*Optional[float]*) – A float in WGS'84 decimal degree (-180 to +180) of the center Point.
- **ref_space_id** (*Optional[str]*) – A string as alternative for defining center coordinates, it is possible to reference a geometry in a space, hence it is needed to provide the `ref_space_id` where the referenced feature is stored. Always to use in combination with `ref_feature_id`.
- **ref_feature_id** (*Optional[str]*) – A string as unique identifier of a feature in the referenced space. The geometry of that feature gets used for the spatial query. Always to use in combination with `ref_space_id`.
- **radius** (*Optional[int]*) – An int in meter which defines the diameter of the search request.
- **tags** (*Optional[List[str]]*) – A list of strings holding tag values.
- **limit** (*Optional[int]*) – A max. number of features to return in the result.
- **params** (*Optional[dict]*) – A dict to represent additional filter on features to be searched. Examples:
  - `params={"p.name": "foo"}` returns all features with a value of property `name` equal to `foo`.
  - `params={"p.name!": "foo"}` returns all features with a value of property `name` not qual to `foo`.
  - `params={"p.count=gte": "10"}` returns all features with a value of property `count` greater than or equal to `10`.

- params={"p.count=lte": "10"} returns all features with a value of property count less than or equal to 10.
- params={"p.count=gt": "10"} returns all features with a value of property count greater than 10.
- params={"p.count=lt": "10"} returns all features with a value of property count less than 10.
- params={"p.name=cs": "bar"} returns all features with a value of property name which contains bar.

- **selection** (*Optional[List[str]]*) – A list of strings holding properties values.
- **skip_cache** (*Optional[bool]*) – A Boolean if set to True the response is not returned from cache.
- **geo_dataframe** (*Optional[bool]*) – A boolean if set to True searched features will be yield as single Geopandas Dataframe.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Yields A Feature object by default. If param geo_dataframe is True then yields single Geopandas Dataframe.

Return type Generator[geojson.feature.Feature, None, None]

spatial_search_geometry(*data*, *radius=None*, *tags=None*, *limit=None*, *params=None*, *selection=None*, *skip_cache=None*, *divide=False*, *cell_width=None*, *units='m'*, *chunk_size=1*, *geo_dataframe=None*, *force_2d=None*)

Search features which intersect the provided geometry.

Parameters

- **data** (*dict*) – A JSON object which is getting used for the spatial search.
- **radius** (*Optional[int]*) – An int which defines the diameter(meters) of the search request.
- **tags** (*Optional[List[str]]*) – A list of strings holding tag values.
- **limit** (*Optional[int]*) – A max. number of features to return in the result.
- **params** (*Optional[dict]*) – A dict to represent additional filter on features to be searched. Examples:
  - params={"p.name": "foo"} returns all features with a value of property name equal to foo.
  - params={"p.name!": "foo"} returns all features with a value of property name not qual to foo.
  - params={"p.count=gte": "10"} returns all features with a value of property count greater than or equal to 10.
  - params={"p.count=lte": "10"} returns all features with a value of property count less than or equal to 10.
  - params={"p.count=gt": "10"} returns all features with a value of property count greater than 10.
  - params={"p.count=lt": "10"} returns all features with a value of property count less than 10.

  - `params={"p.name=cs": "bar"}` returns all features with a value of property `name` which contains `bar`.

- **selection** (*Optional[List[str]]*) – A list of strings holding properties values.
- **skip_cache** (*Optional[bool]*) – A Boolean if set to `True` the response is not returned from cache.
- **divide** (*Optional[bool]*) – To divide geometry if the resultant features count is large.
- **cell_width** (*Optional[float]*) – Width of each cell in which geometry is to be divided in units specified, default values is meters.
- **units** (*Optional[str]*) – Unit for cell_width please refer, https://github.com/omanges/turfpy/blob/master/measurements.md#units-type
- **chunk_size** (*int*) – Number of chunks each process to handle. The default value is 1, for a large number of features please use *chunk_size* greater than 1 to get better results in terms of performance.
- **geo_dataframe** (*Optional[bool]*) – A boolean if set to `True` searched features will be yield as single Geopandas Dataframe.
- **force_2d** (*Optional[bool]*) – If set to True the features in the response will have only X and Y components, by default all x,y,z coordinates will be returned.

Yields A Feature object by default. If param `geo_dataframe` is True then yields as single Geopandas Dataframe.

Return type Generator[geojson.feature.Feature, None, None]

_spatial_search_geometry(*data*, *feature_list*, *radius=None*, *tags=None*, *limit=None*, *params=None*, *selection=None*, *skip_cache=None*, *force_2d=None*)

Parameters

- **data** (*dict*) –
- **feature_list** (*List[dict]*) –
- **radius** (*Optional[int]*) –
- **tags** (*Optional[List[str]]*) –
- **limit** (*Optional[int]*) –
- **params** (*Optional[dict]*) –
- **selection** (*Optional[List[str]]*) –
- **skip_cache** (*Optional[bool]*) –
- **force_2d** (*Optional[bool]*) –

add_features_geojson(*path*, *encoding='utf-8'*, *features_size=2000*, *chunk_size=1*)

Add features in space from a GeoJSON file.

As API has a limitation on the size of features, features are divided into chunks, and multiple processes will process those chunks. Each chunk has a number of features based on the value of `features_size`. Each process handles chunks based on the value of `chunk_size`.

Parameters

- **path** (*str*) – Path to the GeoJSON file.

- **encoding** (*str*) – A string to represent the type of encoding.
- **features_size** (*int*) – An int representing a number of features to upload at a time.
- **chunk_size** (*int*) – Number of chunks for each process to handle. The default value is 1, for a large number of features please use *chunk_size* greater than 1.

add_features_csv(*path*, *lon_col*, *lat_col*, *id_col=''*, *alt_col=''*, *delimiter=','*, *add_tags=None*, *remove_tags=None*, *features_size=2000*, *chunk_size=1*, *id_properties=None*)

Add features in space from a csv file.

As API has a limitation on the size of features, features are divided into chunks, and multiple processes will process those chunks. Each chunk has a number of features based on the value of `features_size`. Each process handles chunks based on the value of `chunk_size`.

Parameters

- **path** (*str*) – Path to csv file.
- **lon_col** (*str*) – Name of the column for longitude coordinates.
- **lat_col** (*str*) – Name of the column for latitude coordinates.
- **id_col** (*Optional[str]*) – Name of the column for feature id's.
- **alt_col** (*Optional[str]*) – Name of the column for altitude, if not provided altitudes with default value 0.0 will be added.
- **delimiter** (*Optional[str]*) – delimiter which should be used to process the csv file.
- **add_tags** (*Optional[List[str]]*) – A list of strings describing tags to be added to the features.
- **remove_tags** (*Optional[List[str]]*) – A list of strings describing tags to be removed from the features.
- **features_size** (*int*) – An int representing a number of features to upload at a time.
- **chunk_size** (*int*) – Number of chunks each process to handle. The default value is 1, for a large number of features please use *chunk_size* greater than 1 to get better results in terms of performance.
- **id_properties** (*Optional[List[str]]*) – List of properties name from which id to be generated if id does not exists for a feature.

Raises **Exception** – If values of params *lat_col*, *lon_col*, *id_col* do not match with column names in csv file.

cluster(*clustering*, *clustering_params=None*)

Apply clustering algorithm for the space data.

Parameters

- **clustering** (*str*) – Name of the clustering algorithm. Available values : hexbin, quadbin
- **clustering_params** (*Optional[dict]*) – Parameters for the clustering algorithm. Please refer : https://www.here.xyz/api/devguide/usingclustering/

Returns GeoJSON.

Raises **Exception** – If bounding box is not present in space statistics.

Return type dict

Example:

```
>>> from xyzspaces import XYZ
>>> xyz = XYZ(credentials="XYZ_TOKEN")
>>> space = xyz.spaces.from_id(space_id="existing-space-id")
>>> space.cluster(clustering="hexbin")
```

isshared()
Return the shared status of the space.

Returns A boolean to indicate shared status of the space.

Return type bool

add_features_shapefile(*path*, *features_size=2000*, *chunk_size=1*, *encoding='utf-8'*)
Upload shapefile to the space.

Parameters

- **path** (*str*) – A string representing full path of the shapefile. For zipped shapefile prepend `zip://` before path of shapefile.
- **features_size** (*int*) – An int representing a number of features to upload at a time.
- **chunk_size** (*int*) – Number of chunks for each process to handle. The default value is 1, for a large number of features please use *chunk_size* greater than 1.
- **encoding** (*str*) – A string to represent the type of encoding.

Example: >>> from xyzspaces import XYZ >>> xyz = XYZ(credentials="XYZ_TOKEN") >>> space = xyz.spaces.from_id(space_id="existing-space-id") >>> space.add_features_shapefile(path="shapefile.shp")

add_features_wkt(*path*)
To upload data from wkt file to a space

Parameters **path** (*str*) – Path to wkt file

add_features_gpx(*path*, *features_size=2000*, *chunk_size=1*)
Upload data from gpx file to the space.

Parameters

- **path** (*str*) – A string representing full path of the gpx file.
- **features_size** (*int*) – An int representing a number of features to upload at a time.
- **chunk_size** (*int*) – Number of chunks for each process to handle. The default value is 1, for a large number of features please use *chunk_size* greater than 1.

add_features_kml(*path*, *features_size=2000*, *chunk_size=1*)
To upload data from kml file to a space

Parameters

- **path** (*str*) – Path to kml file
- **features_size** (*int*) – An int representing a number of features to upload at a time.
- **chunk_size** (*int*) – Number of chunks for each process to handle. The default value is 1, for a large number of features please use *chunk_size* greater than 1.

add_features_geobuf(*path*, *features_size=2000*, *chunk_size=1*)
To upload data from geobuff file to a space.

Parameters

- **path** (*str*) – Path to geobuff file.

- **features_size** (*int*) – An int representing a number of features to upload at a time.
- **chunk_size** (*int*) – Number of chunks for each process to handle. The default value is 1, for a large number of features please use *chunk_size* greater than 1.

add_features_geopandas(*data*, *features_size=2000*, *chunk_size=1*)
Add features from GeoPandas dataframe to a space.

Parameters

- **data** (*geopandas.geodataframe.GeoDataFrame*) – GeoPandas dataframe to be uploaded
- **features_size** (*int*) – The number of features to upload at a time.
- **chunk_size** (*int*) – Number of chunks for each process to handle. The default value is 1, for a large number of features please use *chunk_size* greater than 1.

clone(*space_id=None*, *chunks=1000*)
Copy current space data into a newly created or into an existing space.

Parameters

- **space_id** (*Optional[str]*) – space id into which to copy data, if not provided will create a new space and copy the data.
- **chunks** (*int*) – A max. number of features to read in a single iteration while iterating over the source space.

Returns The cloned Space Object

browse()
Inspect and analyze space using Data Hub Space Invader. More information about Data Hub Space Invader can be found here.

4.2.9 xyzspaces.tools module

This is a preliminary collection of little tools that use XYZ.

xyzspaces.tools.**subset_geojson**(*token*, *gj*, *bbox=None*, *tile_type=None*, *tile_id=None*, *clip=False*, *lat=None*, *lon=None*, *radius=None*)
Return a subset of the GeoJSON object inside some bbox or map tile or radius.

This will create a temporary space, add the provided GeoJSON object, perform the bbox or tile subsetting and return the resulting GeoJSON object after deleting the temporary space again.

Parameters

- **token** (*str*) – A string containing the XYZ API token.
- **gj** (*dict*) – The GeoJSON data object.
- **bbox** (*Optional[List[float]]*) – The bounding box described as a list of four numbers (its West, South, East, and North margins).
- **tile_type** (*Optional[str]*) – The tile type, for now one of: ...
- **tile_id** (*Optional[str]*) – The tile ID, a string composed of digits to identify the tile according to the specified tile type.
- **clip** (*Optional[bool]*) – A Boolean to indicate if the features should be clipped at the tile or bbox.
- **lat** (*Optional[float]*) – A float to represent latitude.

- **lon** (*Optional[float]*) – A float to represent longitude.
- **radius** (*Optional[int]*) – An int in meter which defines the diameter of the search request. Should be provided with `lat` and `lon` for spatial search.

Returns A GeoJSON object covering the desired tile or bbox subset of the original GeoJSON object.

Raises **ValueError** – If the wrong combination of `bbox`, `tile_type` and `tile_id`, `lat` and `lon` was provided.

Return type dict

4.2.10 xyzspaces.utils module

This is a collection of utilities for using XYZ Hub.

Actually, they are almost unspecific to any XYZ Hub functionality, apart from *feature_to_bbox()*, but convenient to use.

xyzspaces.utils.**join_string_lists**(***kwargs*)

Convert named lists of strings to one dict with comma-separated strings.

Parameters **kwargs** – Lists of strings

Returns Converted dict.

Return type dict

Example:

```
>>> join_string_lists(foo=["a", "b", "c"], bar=["a", "b"], foobar=None)
{"foo": "a,b,c", "bar": "a,b"}
```

xyzspaces.utils.**feature_to_bbox**(*feature*)

Extract bounding box from GeoJSON feature rectangle.

Parameters **feature** (*dict*) – A dict representing a GeoJSON feature.

Returns A list of four floats representing West, South, East and North margins of the resulting bounding box.

Return type List[float]

xyzspaces.utils.**get_xyz_token**()

Read and return the value of the environment variable `XYZ_TOKEN`.

Returns The string value of the environment variable or an empty string if no such variable could be found.

Return type str

xyzspaces.utils.**grouper**(*size*, *iterable*, *fillvalue=None*)

Create groups of *size* each from given iterable.

Parameters

- **size** – An int representing size of each group.
- **iterable** – An iterable.
- **fillvalue** – Value to put for the last group.

Returns A generator.

xyzspaces.utils.**wkt_to_geojson**(*wkt_data*)
Converts wkt to geojson

Parameters **wkt_data** (*str*) – wkt data to be converted

Returns Geojson

Return type dict

xyzspaces.utils.**grid**(*bbox*, *cell_width*, *cell_height*, *units*)
This function generates the grids for the given bounding box

Parameters

- **bbox** – bounding box coordinates
- **cell_width** – Cell width in specified in units
- **cell_height** – Cell height in specified in units
- **units** – Units for given sizes

Returns FeatureCollection of grid boxes genarated for the giving bounding box

xyzspaces.utils.**divide_bbox**(*feature*, *cell_width=None*, *units='m'*)
Divides the given feature into grid boxes as per given cell width

Parameters

- **feature** (*dict*) – Feature to be divide in grid
- **cell_width** (*Optional[float]*) – Width of each grid boxs
- **units** (*Optional[str]*) – Units for the width of grid boxs

Returns List of features in which the input feature is divided

xyzspaces.utils.**flatten_geometry**(*data*)
Flatten the geometries in the given GeoPandas dataframe. Flatten geometry is formed by extracting individual geometries from GeometryCollection, MultiPoint, MultiLineString, MultiPolygon.

Parameters **data** (*geopandas.geodataframe.GeoDataFrame*) – GeoPandas dataframe to be flatten

Returns Flat GeoPandas dataframe

Return type geopandas.geodataframe.GeoDataFrame

CHAPTER

FIVE

LOGGING CONFIGURATION

By default logging is disabled. To enable logging, use below code snippets in your python code to setup logging at DEBUG level:

```
import logging
from xyzspaces import setup_logging

setup_logging(default_level=logging.DEBUG)
```

Default logging configuration is defined in a file. This ensures that log messages will be written to the file `xyz.log` in your current working directory. Here is an example log file (xyz.log):

```
2020-02-21 17:55:46,132 - apis.py:130 - ERROR - Curl command: curl --request GET␣
↪https://xyz.api.here.com/hub/spaces/dummy-111 --header "Authorization: Bearer <XYZ_
↪TOKEN>"
2020-02-21 17:55:46,133 - apis.py:131 - ERROR - Response status code: 404
2020-02-21 17:55:46,133 - apis.py:132 - ERROR - Response headers: {'Content-Type':
↪'application/json', 'Content-Length': '150', 'Connection': 'keep-alive', 'Date':
↪'Fri, 21 Feb 2020 12:25:46 GMT', 'x-amzn-RequestId': '397c8039-79f1-4956-bbe4-
↪46ca78c7ec2d', 'content-encoding': 'gzip', 'Stream-Id': '397c8039-79f1-4956-bbe4-
↪46ca78c7ec2d', 'x-amzn-Remapped-Content-Length': '150', 'x-amzn-Remapped-Connection
↪': 'keep-alive', 'x-amz-apigw-id': 'IPzblGVFjoEF5pg=', 'x-amzn-Remapped-Date': 'Fri,
↪ 21 Feb 2020 12:25:46 GMT', 'X-Cache': 'Error from cloudfront', 'Via': '1.1␣
↪e25383e25378de918d3b187b3239eb5b.cloudfront.net (CloudFront)', 'X-Amz-Cf-Pop':
↪'BOM51-C2', 'X-Amz-Cf-Id': 'nZAJUB_FBiHdojziSoG3SBcMdf8rNyHuOMSlJljyxDNlx1I0O3t9YQ==
↪'}
2020-02-21 17:55:46,134 - apis.py:133 - ERROR - Response text: {"type":"ErrorResponse
↪","error":"Exception","errorMessage":"The requested resource does not exist.",
↪"streamId":"397c8039-79f1-4956-bbe4-46ca78c7ec2d"}
```

To customize the logging configuration, set the variable XYZ_LOG_CONFIG to hold the full path of the logging configuration options file logging_config.json:

```
export XYZ_LOG_CONFIG=~/logging_config.json
```

CHAPTER

SIX

TESTS

xyzspaces uses pytest for testing.

You can run the test suite locally:

```
pip install -r requirements_dev.txt
pytest -v --cov=xyzspaces tests
```

The test suite provides test coverage of around 98% (but less if the tests cannot find your credentials).

CHAPTER

SEVEN

CHANGELOG

7.1 xyzspaces 0.5.0 (2021-02-01)

- **Features**
  - Added functionality to `clone` Space. (#93)
  - Handle HTTP 429 responses with `backoff` package. (#95)
  - support the new `force2D` parameter for all the APIs used to read features. (#96)
  - Add support for new mode and vizSampling params in HubApi.get_space_tile. (#101)
- **Documentation**
  - Start collecting/documenting architecture decision records (ADRs). (#90)
  - Add support for executable code in Sphinx documentation. (#99)
- **Misc**
  - Migrated CI from Travis to GitHub Actions. (#111)

7.2 xyzspaces 0.4.0 (2020-09-18)

- **Features**
  - Added feature to upload data from `kml` file to the space.(#49)
  - Added `limit` param to method `iter_feature` to control numer of features to iterate in single call.(#52)
  - Fixed encoding and projections issue for `shapefile` upload.(#54)
  - Enabled property search while searching features in bounding box.(#56)
  - Added feature to upload data from `geobuff` file to the space.(#57)
  - Remove duplicate features for `spatial_search_geometry` with division(#58)
  - Enabled property search while searching features in tile.(#61)
  - Remove duplicate features while add to space using `add_features` and also added a `mutation` parameter to mutate input features or not.(#64)
  - `description` is optional when creating the space.(#68)
  - Added feature to upload data from `Geopandas Dataframe` file to the space.(#71)

  - Enabled reading space data as Geopandas Dataframe.(#72)
  - Improved performance of CSV upload.(#77)
  - Improvement in the performance of `add_features_geojson`.(#79)
  - Changes to convert shape file with projection of different type to EPSG:4326.(#83)
- **Documentation**
  - New notebook illustrating spatial search on MS US building footprints dataset.(#62)

7.3 xyzspaces 0.3.2 (2020-08-19)

- **Features**
  - Added feature to upload data from `shapefile` to the space.(#40)
  - Added feature to upload data from `WKT` file to the space.(#41)
  - Added feature to upload data from `gpx` file to the space.(#42)
  - Optimized the `spatial search` to search features from large geometries.(#44)
- **Misc**
  - Added Binder support to the repository.(#28)
  - Added **clientId** in query params of the Hub API requests.(#36)
  - Updated `__version__` attribute now it can be used as `from xyzspaces import __version__`.(#38)

7.4 xyzspaces 0.3.1 (2020-07-24)

- **Misc**
  - Minor changes to README.(0.3.1)

7.5 xyzspaces 0.3.0 (2020-07-24)

- **Misc**
  - First public release.(0.3.0)

CHAPTER

EIGHT

BLOGS AND TALKS

Various blog posts and conference presentations about `xyzspaces`:

8.1 Blogs

- A simple way to build your own 3D map with Coloured LIDAR point clouds using xyzspaces.

8.2 Talks

- Online Python Machine Learning Conference & GeoPython 2020

CHAPTER

NINE

CONTRIBUTING TO XYZSPACES

9.1 Overview

Contributions to xyzspaces are very welcome. They are likely to be accepted more quickly if they follow these guidelines.

Below are guidelines, when submitting a pull request:

- All existing tests should pass. Please make sure that the test suite passes, both locally and on Travis CI. Status on Travis will be visible on a pull request.
- New functionality should include tests. Please write reasonable tests for your code and make sure that they pass on your pull request.
- Classes, methods, functions, etc. should have docstrings. The first line of a docstring should be a standalone summary. Parameters and return values should be documented explicitly.
- Follow PEP 8 when possible. We use Black, Flake8, isort, typing and darglint to ensure a consistent code format throughout the project. For more details see *below*.
- Imports should be grouped with standard library imports first, 3rd-party libraries next, and xyzspaces imports third. Within each grouping, imports should be alphabetized. Always use absolute imports when possible, and explicit relative imports for local imports when necessary in tests.
- xyzspaces supports Python 3.6+.

9.1.1 Seven Steps for Contributing

There are seven basic steps to contributing to *xyzspaces*:

1) Fork the *xyzspaces* git repository
2) Create a development environment
3) Install *xyzspaces* dependencies
4) Make changes to code and add tests
5) Run linting
6) Update the documentation
7) Submit a Pull Request

Each of these 7 steps is detailed below.

9.2 1) Forking the *xyz-spaces-python* repository using Git

To the new user, working with Git is one of the more daunting aspects of contributing to *xyzspaces**. It can very quickly become overwhelming, but sticking to the guidelines below will help keep the process straightforward and mostly trouble free. As always, if you are having difficulties please feel free to ask for help.

The code is hosted on GitHub. To contribute you will need to sign up for a free GitHub account.

Some great resources for learning Git:

- Software Carpentry's Git Tutorial
- Atlassian
- the GitHub help pages.
- Matthew Brett's Pydagogue.

9.2.1 Getting started with Git

GitHub has instructions for installing git, setting up your SSH key, and configuring git. All these steps need to be completed before you can work seamlessly between your local repository and GitHub.

9.2.2 Forking

You will need your own fork to work on the code. Go to the xyz-spaces-python project page and hit the `Fork` button. You will want to clone your fork to your machine:

```
git clone git@github.com:your-user-name/xyz-spaces-python.git xyz-spaces-python-
↪yourname
cd xyz-spaces-python-yourname
git remote add upstream git://github.com/heremaps/xyz-spaces-python.git
```

This creates the directory *xyz-spaces-python-yourname* and connects your repository to the upstream (main project) *xyzspaces* repository.

The testing suite will run automatically on Travis-CI once your pull request is submitted. However, if you wish to run the test suite on a branch prior to submitting the pull request, then Travis-CI needs to be hooked up to your GitHub repository. Instructions for doing so are here.

9.2.3 Creating a branch

You want your master branch to reflect only production-ready code, so create a feature branch for making your changes. For example:

```
git branch shiny-new-feature
git checkout shiny-new-feature
```

The above can be simplified to:

```
git checkout -b shiny-new-feature
```

This changes your working directory to the shiny-new-feature branch. Keep any changes in this branch specific to one bug or feature so it is clear what the branch brings to *xyzspaces*. You can have many shiny-new-features and switch in between them using the git checkout command.

To update this branch, you need to retrieve the changes from the master branch:

```
git fetch upstream
git rebase upstream/master
```

This will replay your commits on top of the latest xyzspaces git master. If this leads to merge conflicts, you must resolve these before submitting your pull request. If you have uncommitted changes, you will need to `stash` them prior to updating. This will effectively store your changes and they can be reapplied after updating.

9.3 2) Creating a development environment

A development environment is a virtual space where you can keep an independent installation of *xyzspaces*. This makes it easy to keep both a stable version of python in one place you use for work, and a development version (which you may break while playing with code) in another.

An easy way to create a *xyzspaces* development environment is as follows:

- Install either Anaconda or miniconda
- Make sure that you have *cloned the repository*
- `cd` to the *xyzspaces** source directory

Tell conda to create a new environment, named `xyz_dev`, or any other name you would like for this environment, by running:

```
conda create -n xyz_dev python
```

This will create the new environment, and not touch any of your existing environments, nor any existing python installation.

To work in this environment, you need to `activate` it. The instructions below should work for both Windows, Mac and Linux:

```
conda activate xyz_dev
```

Once your environment is activated, you will see a confirmation message to indicate you are in the new development environment.

To view your environments:

```
conda info -e
```

To return to you home root environment:

```
conda deactivate
```

See the full conda docs here.

At this point you can easily do a *development* install, as detailed in the next sections.

9.4 3) Installing Dependencies

To run *xyzspaces* in an development environment, you must first install *xyzspaces*'s dependencies. We suggest doing so using the following commands (executed after your development environment has been activated):

```
pip install -r requirements.txt
pip install -r requirements_dev.txt
```

This should install all necessary dependencies.

9.5 4) Making changes and writing tests

xyzspaces is serious about testing and strongly encourages contributors to embrace test-driven development (TDD). This development process "relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test." So, before actually writing any code, you should write your tests. Often the test can be taken from the original GitHub issue. However, it is always worth considering additional use cases and writing corresponding tests.

Adding tests is one of the most common requests after code is pushed to *xyzspaces*. Therefore, it is worth getting in the habit of writing tests ahead of time so this is never an issue.

xyzspaces uses the pytest framework.

9.5.1 Writing tests

All tests should go into the `tests` directory. This folder contains many current examples of tests, and we suggest looking to these for inspiration.

9.5.2 Running the test suite

The tests can then be run directly inside your Git clone (without having to install *xyzspaces*) by typing:

```
pytest -v --cov=xyzspaces tests
```

9.6 5) Run linting

For linting please refer *contributing style*

9.7 6) Updating the Documentation

xyzspaces documentation resides in the *docs* folder. Changes to the docs are make by modifying the appropriate file in the *source* folder within *docs*. *xyzspaces* docs use reStructuredText syntax, which is explained here and the docstrings follow the Sphinx Docstring standard.

Once you have made your changes, you may try if they render correctly by building the docs using sphinx. To do so, you can type from project's root folder:

```
sh scripts/build_docs.sh
```

The resulting html pages will be located in *docs/source/_build/html*.

9.8 7) Submitting a Pull Request

Once you've made changes and pushed them to your forked repository, you then submit a pull request to have them integrated into the *xyzspaces* code base.

You can find a pull request (or PR) tutorial in the GitHub's Help Docs.

9.9 Style Guide & Linting

xyzspaces follows the PEP8 standard and uses Black, Flake8, isort, typing and darglint to ensure a consistent code format throughout the project.

Continuous Integration (Travis CI) will run those tools and report any stylistic errors in your code. Therefore, it is helpful before submitting code to run the check yourself. To autoformat the code run:

```
make black
```

To check linting errors run:

```
make lint
```

To check typing errors run:

```
make typing
```

9.10 Signing each Commit

As part of filing a pull request we ask you to sign off the Developer Certificate of Origin (DCO) in each commit. Any Pull Request with commits that are not signed off will be reject by the DCO check. A DCO is lightweight way for a contributor to confirm that you wrote or otherwise have the right to submit code or documentation to a project. Simply add Signed-off-by as shown in the example below to indicate that you agree with the DCO. An example signed commit message:

```
README.md: Fix minor spelling mistake
Signed-off-by: John Doe <john.doe@example.com>
```

Git has the -s flag that can sign a commit for you, see example below:

```
$ git commit -s -m 'README.md: Fix minor spelling mistake'
```

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

X

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

P