mapbox Documentation

Release 0.16

Mapbox

Aug 01, 2022

CONTENTS

A Python client for Mapbox web services

The Mapbox Python SDK is a low-level client API, not a Resource API such as the ones in boto3 or github3.py. Its methods return objects containing HTTP responses from the Mapbox API.

CHAPTER

ONE

SERVICES

- **Analytics V1** examples, website
  - API usage for services by resource.
  - available for premium and enterprise plans.
- **Directions V4** examples, website
  - Profiles for driving, walking, and cycling
  - GeoJSON & Polyline formatting
  - Instructions as text or HTML
- **Geocoding V5** examples, website
  - Forward (place names longitude, latitude)
  - Reverse (longitude, latitude place names)
- **Map Matching V4** examples, website
  - Snap GPS traces to OpenStreetMap data
- **Static Maps V4** examples, website
  - Generate standalone images from existing Mapbox *mapids* (tilesets)
  - Render with GeoJSON overlays
- **Static Styles V1** examples, website
  - Generate standalone images from existing Mapbox *styles*
  - Render with GeoJSON overlays
  - Adjust pitch and bearing, decimal zoom levels
- **Surface V4 DEPRECATED**
- **Uploads V1** examples, website
  - Upload data to be processed and hosted by Mapbox.
- **Datasets V1** examples, website
  - Manage editable collections of GeoJSON features
  - Persistent storage for custom geographic data
- **Tilequery V4** examples, website
  - Retrieve data about specific features from a vector tileset

- **Maps V4** examples, website
  - Retrieve an image tile, vector tile, or UTFGrid in the specified format
  - Retrieve vector features from Mapbox Editor projects as GeoJSON or KML
  - Retrieve TileJSON metadata for a tileset
  - Retrieve a single marker image without any background map

Please note that there may be some lag between the release of new Mapbox web services and releases of this package.

CHAPTER

TWO

INSTALLATION

```
$ pip install mapbox
```

CHAPTER

THREE

TESTING

```
$ pip install -e .[test]
$ python -m pytest
```

To run the examples as integration tests on your own Mapbox account

```
$ MAPBOX_ACCESS_TOKEN="MY_ACCESS_TOKEN" python -m pytest --doctest-glob='*.md' docs/*.md
```

CHAPTER

FOUR

SEE ALSO

- Command line interface: https://github.com/mapbox/mapbox-cli-py
- Javascript SDK: https://github.com/mapbox/mapbox-sdk-js

CHAPTER

FIVE

DOCUMENTATION

Access Tokens

All Mapbox APIs require an access token. Thus all service object constructors take an *access_token* keyword argument. Access can be granted to a geocoding service, for example, like so:

```python >>> from mapbox import Geocoder >>> geocoder = Geocoder(access_token="pk.YOUR_ACCESS_TOKEN")
```

Please note that an actual token string must be used. Tokens may be generated using the web application at [https://www.mapbox.com/account/access-tokens{}](https://www.mapbox.com/account/access-tokens).

Your Mapbox access token can also be set in the environment of your program

`bash export MAPBOX_ACCESS_TOKEN="pk.YOUR_ACCESS_TOKEN" `

and it will be found automatically when creating a new instance. We'll use the *Geocoder* in this example but the same applies for all *mapbox* classes.

```python >>> geocoder = Geocoder() >>> import os >>> geocoder.session.params['access_token'] == os.environ['MAPBOX_ACCESS_TOKEN'] True
```

Best practice for access tokens and geocoding sources is to create a new instance for each new access token or source dataset.

Special considerations

You access token can be associated with different *scopes*. **TODO** How to get an access token. **TODO**

Datasets

The *Datasets* class from the *mapbox.services.datasets* module provides access to the Mapbox Datasets API. You can also import it directly from the *mapbox* module.

```python >>> from mapbox import Datasets
```

See https://www.mapbox.com/api-documentation/maps/#datasets for general documentation of the API.

Your Mapbox access token should be set in your environment; see the [access tokens](access_tokens.md) documentation for more information. To use the Datasets API, you must use a token created with `datasets:*` scopes. See https://www.mapbox.com/account/apps/.

Upload methods

The methods of the *Datasets* class that provide access to the Datasets API generally take dataset id and feature id arguments and return an instance of [*requests.Response*](http://docs.python-requests.org/en/latest/api/#requests.Response).

Usage

Create a new dataset using the *Dataset* class, giving it a name and description. The *id* of the created dataset is in JSON data of the response.

```python >>> datasets = Datasets() >>> create_resp = datasets.create( ... name='example', description='An example dataset') >>> new = create_resp.json() >>> new['name'] 'example' >>> new['description'] 'An example dataset' >>> new_id = new['id']

```

You can find it in your account's list of datasets.

```python >>> listing_resp = datasets.list() >>> [ds['id'] for ds in listing_resp.json()] [...]

```

Instead of scanning the list for attributes of the dataset, you can read them directly by dataset id.

```python >>> attrs = datasets.read_dataset(new_id).json() >>> attrs['id'] == new_id True >>> attrs['name'] 'example' >>> attrs['description'] 'An example dataset'

```

If you want to change a dataset's name or description, you can.

```python >>> attrs = datasets.update_dataset( ... new_id, name='updated example', description='An updated example dataset' ... ).json() >>> # attrs = datasets.read_dataset(new_id).json() >>> attrs['id'] == new_id True >>> attrs['name'] 'updated example' >>> attrs['description'] 'An updated example dataset'

```

You can delete the dataset and it will no longer be present in your listing.

```python >>> resp = datasets.delete_dataset(new_id) >>> resp.status_code 204 >>> listing_resp = datasets.list() >>> [ds['id'] for ds in listing_resp.json()] [...]

```

Dataset features

The main point of a dataset is store a collection of GeoJSON features. Let us create a new dataset and then add a GeoJSON feature to it.

```python >>> resp = datasets.create( ... name='features-example', description='An example dataset with features') >>> new_id = resp.json()['id'] >>> feature = { ... 'type': 'Feature', 'id': '1', 'properties': {'name': 'Insula Nulla'}, ... 'geometry': {'type': 'Point', 'coordinates': [0, 0]}} >>> resp = datasets.update_feature(new_id, '1', feature) >>> resp.status_code 200

```

In the feature collection of the dataset you will see this feature.

```python >>> collection = datasets.list_features(new_id).json() >>> len(collection['features']) 1 >>> first = collection['features'][0] >>> first['id'] '1' >>> first['properties']['name'] 'Insula Nulla'

```

Individual feature access

You can also read, update, and delete features individually.

Read

The *read_feature()* method has the semantics of HTTP GET.

```python >>> resp = datasets.read_feature(new_id, '1') >>> resp.status_code 200 >>> feature = resp.json() >>> feature['id'] '1' >>> feature['properties']['name'] 'Insula Nulla'

```

Update

The *update_feature()* method has the semantics of HTTP PUT. If there is no feature in the dataset with the given id, a new feature will be created.

```python >>> update = { ... 'type': 'Feature', 'id': '1', 'properties': {'name': 'Insula Nulla C'}, ... 'geometry': {'type': 'Point', 'coordinates': [0, 0]}} >>> update = datasets.update_feature(new_id, '1', update).json() >>> update['id'] '1' >>> update['properties']['name'] 'Insula Nulla C'

```

Delete

The *delete_feature()* method has the semantics of HTTP DELETE.

```python >>> resp = datasets.delete_feature(new_id, '1') >>> resp.status_code 204

```

Finally, let's clean up the features example dataset.

```python >>> resp = datasets.delete_dataset(new_id) >>> resp.status_code 204 >>> listing_resp = datasets.list() >>> [ds['id'] for ds in listing_resp.json()] [...]

```

Geocoding

The *Geocoder* class from the *mapbox.services.geocoding* module provides access to the Mapbox Geocoding API. You can also import it directly from the *mapbox* module.

```python

```
>>> from mapbox import Geocoder
```

```

See https://www.mapbox.com/api-documentation/search/#geocoding for general documentation of the API.

Your Mapbox access token should be set in your environment; see the [access tokens](access_tokens.md) documentation for more information.

Geocoding sources

If your account enables access to the *mapbox.places-permanent* dataset, you can use it specify it with a keyword argument to the *Geocoder* constructor.

```python

```
>>> perm_geocoder = Geocoder(name='mapbox.places-permanent')
```

```

For the default *mapbox.places* geocoder, you don't need to specify any arguments

```python

```
>>> geocoder = Geocoder()
```
```

```

## Geocoder methods

The methods of the *Geocoder* class that provide access to the Geocoding API return an instance of [*requests.Response*](http://docs.python-requests.org/en/latest/api/#requests.Response). In addition to the *json()* method that returns Python data parsed from the API, the *Geocoder* responses provide a *geojson()* method that converts that data to a GeoJSON like form.

## Limits

The Geocoding API is rate limited. Details of the limits and current state are accessible through response headers.

```python
>>> response = geocoder.forward('Chester, NJ')
>>> response.headers['X-Rate-Limit-Interval']
'60'
>>> response.headers['X-Rate-Limit-Limit']
'600'
>>> response.headers['X-Rate-Limit-Reset']
'1447701074'
```

Response format

The JSON response extends GeoJSON's *FeatureCollection*.

```python
>>> response = geocoder.forward('Chester, NJ')
>>> collection = response.json()
>>> collection['type'] == 'FeatureCollection'
True
>>> sorted(collection.keys())
['attribution', 'features', 'query', 'type']
>>> collection['query']
['chester', 'nj']
```

Zero or more objects that extend GeoJSON's *Feature* are contained in the collection, sorted by relevance to the query.

```python
>>> first = collection['features'][0]
>>> first['type'] == 'Feature'
True
```

Forward geocoding

Places at an address may be found using *Geocoder.forward()*.

```python
>>> response = geocoder.forward("200 queen street")
>>> response.status_code
```

(continues on next page)
```

(continued from previous page)

```
200
>>> response.headers['Content-Type']
'application/vnd.geo+json; charset=utf-8'
>>> first = response.geojson()['features'][0]
>>> first['place_name']
'200 Queen St...'
```

```

## Forward geocoding with proximity

Place results may be biased toward a given longitude and latitude.

```python
>>> response = geocoder.forward(
...     "200 queen street", lon=-66.05, lat=45.27)
>>> response.status_code
200
>>> first = response.geojson()['features'][0]
>>> first['place_name']
'200 Queen St...'
>>> [int(coord) for coord in first['geometry']['coordinates']]
[-66, 45]
```

```

## Forward geocoding with bounding box

Place results may be limited to those falling within a given bounding box.

```python
>>> response = geocoder.forward(
...     "washington", bbox=[-78.338320,38.520792,-77.935454,38.864909], types=('place',))
>>> response.status_code
200
>>> first = response.geojson()['features'][0]
>>> first['place_name']
'Washington, Virginia, United States'
>>> [round(coord, 2) for coord in first['geometry']['coordinates']]
[-78.16, 38.71]
```

``` ## Forward geocoding with limited results

The number of results may be limited.

```python
>>> response = geocoder.forward(
...     "washington", limit=3)
>>> response.status_code
200
>>> len(response.geojson()['features'])
3
```

```

## Reverse geocoding

Places at a longitude, latitude point may be found using *Geocoder.reverse()*.

```python
>>> response = geocoder.reverse(lon=-73.989, lat=40.733)
>>> response.status_code
200
>>> features = sorted(response.geojson()['features'], key=lambda x: x['place_name'])
>>> for f in features:
...     print('{place_name}: {id}'.format(**f))
120 East 13th Street, Manhattan, New York, New York 10003... address...
Greenwich Village... neighborhood...
Manhattan... locality...
New York, New York... postcode...
New York, New York... place...
New York... region...
United States: country...
```

Reverse geocoding with limited results by location type

The number of results may be limited by a single type

```python
>>> response = geocoder.reverse(lon=-73.989, lat=40.733, limit=1, types=['country'])
>>> response.status_code
200
>>> features = response.geojson()['features']
>>> len(features)
1
>>> print('{place_name}: {id}'.format(**features[0]))
United States: country...
```

Filtering by country code

forward() can be restricted to a list of country codes. No results in Canada will be returned if the query is filtered for 'us' results only.

```python
>>> response = geocoder.forward("200 queen street", country=['us'])
>>> response.status_code
200
>>> any(['Canada' in f['place_name'] for f in response.geojson()['features']])
False
```

Filtering by type

Both *forward()* and *reverse()* can be restricted to one or more place types.

```python
```

```
>>> response = geocoder.reverse(
...     lon=-73.989, lat=40.733, types=['poi'])
>>> response.status_code
200
>>> features = response.geojson()['features']
>>> all([f['id'].startswith('poi') for f in features])
True
```

```

# Directions Matrix

The *DirectionsMatrix* class from the *mapbox.services.matrix* module provides access to the Mapbox Matrix API V1. You can also import it directly from the *mapbox* module.

```python >>> from mapbox import DirectionsMatrix

```

See https://www.mapbox.com/api-documentation/navigation/#matrix for general documentation of the API.

Your Mapbox access token should be set in your environment; see the [access tokens](access_tokens.md) documentation for more information.

DirectionsMatrix methods

DirectionsMatrix methods return an instance of [*requests.Response*](http://docs.python-requests.org/en/latest/api/#requests.Response). If the response is successful, the *json()* method returns Python data parsed directly from the API.

Usage

If you need to optimize travel between several waypoints, you can use the Matrix API to create a matrix showing travel times between all waypoints. Each of your input waypoints should be a GeoJSON point feature, a GeoJSON geometry, or a (longitude, latitude) pair.

```python >>> service = DirectionsMatrix()

```

The input waypoints to the *directions* method are [features](input_features.md), typically GeoJSON-like feature dictionaries.

```python >>> portland = { … ‘type’: ‘Feature’, … ‘properties’: {‘name’: ‘Portland, OR’}, … ‘geometry’: { … ‘type’: ‘Point’, … ‘coordinates’: [-122.7282, 45.5801]}} >>> bend = { … ‘type’: ‘Feature’, … ‘properties’: {‘name’: ‘Bend, OR’}, … ‘geometry’: { … ‘type’: ‘Point’, … ‘coordinates’: [-121.3153, 44.0582]}} >>> corvallis = { … ‘type’: ‘Feature’, … ‘properties’: {‘name’: ‘Corvallis, OR’}, … ‘geometry’: { … ‘type’: ‘Point’, … ‘coordinates’: [-123.268, 44.5639]}}

```

The *matrix* method can be called with a list of point features and the travel profile.

```python >>> response = service.matrix([portland, bend, corvallis], profile=’mapbox/driving’) >>> response.status_code 200 >>> response.headers[‘Content-Type’] ‘application/json; charset=utf-8’

```

And the response JSON contains a matrix, a 2-D list with travel times (seconds) between all input waypoints. The diagonal will be zeros.

```python >>> from pprint import pprint >>> pprint(response.json()[‘durations’]) [[0.0, …, …], […, 0.0, …], […, …, 0.0]]

```

See `import mapbox; help(mapbox.DirectionsMatrix)` for more detailed usage.

Directions

The *Directions* class from the *mapbox.services.directions* module provides access to the Mapbox Directions API. You can also import it directly from the *mapbox* module.

```python >>> from mapbox import Directions
```

See https://www.mapbox.com/api-documentation/navigation/#directions for general documentation of the API.

Your Mapbox access token should be set in your environment; see the [access tokens](access_tokens.md) documentation for more information.

Directions methods

The methods of the *Directions* class that provide access to the Directions API return an instance of [*requests.Response*](http://docs.python-requests.org/en/latest/api/#requests.Response). In addition to the *json()* method that returns Python data parsed from the API, the *Directions* responses provide a *geojson()* method that converts that data to a GeoJSON like form.

Usage

To get travel directions between waypoints, you can use the Directions API to route up to 25 points. Each of your input waypoints will be visited in order and should be represented by a GeoJSON point feature.

```python >>> service = Directions()
```

The input waypoints to the *directions* method are [features](input_features.md), typically GeoJSON-like feature dictionaries.

```python >>> origin = { … 'type': 'Feature', … 'properties': {'name': 'Portland, OR'}, … 'geometry': { … 'type': 'Point', … 'coordinates': [-122.7282, 45.5801]}} >>> destination = { … 'type': 'Feature', … 'properties': {'name': 'Bend, OR'}, … 'geometry': { … 'type': 'Point', … 'coordinates': [-121.3153, 44.0582]}}
```

The *directions()* method can be called with a list of features and the desired profile.

```python >>> response = service.directions([origin, destination], … 'mapbox.driving') >>> response.status_code 200 >>> response.headers['Content-Type'] 'application/json; charset=utf-8'
```

It returns a response object with a *geojson()* method for accessing the route(s) as a GeoJSON-like FeatureCollection dictionary.

```python >>> driving_routes = response.geojson() >>> driving_routes['features'][0]['geometry']['type'] 'LineString'
```

See `import mapbox; help(mapbox.Directions)` for more detailed usage.

Input features

Many of the Mapbox APIs take geographic features (waypoints) as input.

The *mapbox* module supports the following inputs

- An iterable of GeoJSON-like `Feature`s
- An iterable of objects which implement the [*__geo_interface__*](https://gist.github.com/sgillies/2217756)
```

Static Maps

The *Static* class from the *mapbox.services.static* module provides access to the Mapbox Static Maps API. You can also import it directly from the *mapbox* module.

```python >>> from mapbox import Static

```

See https://www.mapbox.com/api-documentation/legacy/static-classic for general documentation of the API.

Your Mapbox access token should be set in your environment; see the [access tokens](access_tokens.md) documentation for more information.

Static methods

The methods of the *Static* class that provide access to the Static Maps API return an instance of [*requests.Response*](http://docs.python-requests.org/en/latest/api/#requests.Response). Its *content()* method returns the raw bytestring that can be saved into an image file with the appropriate extension.

Usage

Static maps are standalone images that can be displayed on web and mobile devices without the aid of a mapping library or API.

```python >>> service = Static()

```

```python >>> response = service.image('mapbox.satellite', ... lon=-61.7, lat=12.1, z=12) >>> response.status_code 200 >>> response.headers['Content-Type'] 'image/png'

```

Static maps can also display GeoJSON overlays and the [simplestyle-spec](https://github.com/mapbox/simplestyle-spec) styles will be respected and rendered.

```python >>> portland = { ... 'type': 'Feature', ... 'properties': {'name': 'Portland, OR'}, ... 'geometry': { ... 'type': 'Point', ... 'coordinates': [-122.7282, 45.5801]}} >>> bend = { ... 'type': 'Feature', ... 'properties': {'name': 'Bend, OR'}, ... 'geometry': { ... 'type': 'Point', ... 'coordinates': [-121.3153, 44.0582]}}

```

If features are provided the map image will be centered on them and will cover their extents.

```python >>> response = service.image('mapbox.satellite', ... features=[portland, bend]) >>> response.status_code 200 >>> response.headers['Content-Type'] 'image/png'

```

Finally, the contents can be written to file.

```python >>> with open('/tmp/map.png', 'wb') as output: ... _ = output.write(response.content)

```

![map.png](map.png)

See `import mapbox; help(mapbox.Static)` for more detailed usage.

Uploads

The *Uploads* class from the *mapbox.services.uploads* module provides access to the Mapbox Uploads API. You can also import it directly from the *mapbox* module.

```python >>> from mapbox import Uploader

```

See https://www.mapbox.com/api-documentation/maps/#uploads for general documentation of the API.

Your Mapbox access token should be set in your environment; see the [access tokens](access_tokens.md) documentation for more information. To use the Uploads API, you must use a token created with `uploads:*` scopes. See https://www.mapbox.com/account/apps/.

Upload methods

The methods of the *Uploads* class that provide access to the Uploads API return an instance of [*requests.Response*](http://docs.python-requests.org/en/latest/api/#requests.Response).

Usage

Upload any supported file to your account using the `Uploader`. The file object must be opened in binary mode (*rb*) and produce bytes when read, not unicode strings.

The name of the destination dataset can be any string of <= 32 chars. Choose one suited to your application or generate one using, e.g., *uuid.uuid4().hex*. In the example below, we use a string defined in a test fixture.

```python >>> service = Uploader() >>> from time import sleep >>> from random import randint >>> mapid = get-fixture('uploads_dest_id') # 'uploads-test' >>> with open('tests/twopoints.geojson', 'rb') as src: ... upload_resp = service.upload(src, mapid) ... >>> if upload_resp.status_code == 422: ... for i in range(5): ... sleep(5) ... with open('tests/twopoints.geojson', 'rb') as src: ... upload_resp = service.upload(src, mapid) ... if up-load_resp.status_code != 422: ... break

```

This 201 Created response indicates that your data file has been received and is being processed. Poll the Upload API to determine if the processing has finished using the upload identifier from the the body of the above response.

```python >>> upload_resp.status_code 201 >>> upload_id = upload_resp.json()['id'] >>> for i in range(5): ... sta-tus_resp = service.status(upload_id).json() ... if status_resp['complete']: ... break ... sleep(5) ... >>> mapid in status_resp['tileset'] True

```

See `import mapbox; help(mapbox.Uploader)` for more detailed usage.

Analytics

The *Analytics* class from the *mapbox.services.analytics* module provides access to the Mapbox Analytics API. You can also import it directly from the *mapbox* module.

Note: This API is available only for premium and enterprise plans.

```python >>> from mapbox import Analytics

```

See https://www.mapbox.com/api-documentation/accounts/#analytics for general documentation of the API.

Your Mapbox access token should be set in your environment; see the [access tokens](access_tokens.md) documentation for more information.

Analytics methods

The methods of *Analytics* class that provide access to the Analytics API return an instance of [*requests.Response*](http://docs.python-requests.org/en/latest/api/#requests.Response). *Analytics* response also include the *json()* method which returns Python data parsed from API.

Usage

The Mapbox Analytics API is used to get API usage for services by resource. It returns counts per day for given resource and period.

```python >>> analytics = Analytics()
```

```

The input to *analytics* method are resource_type, username, id, period, access_token.

```python >>> response = analytics.analytics('accounts', 'mapbox-sdk-py-user') >>> response.status_code 200
```

Map Matching

The *MapMatcher* class from the *mapbox.services.mapmatching* module provides access to the Mapbox Map Matching API. You can also import it directly from the *mapbox* module.

```python >>> from mapbox import MapMatcher
```

See https://www.mapbox.com/api-documentation/navigation/#map-matching for general documentation of the API.

Your Mapbox access token should be set in your environment; see the [access tokens](access_tokens.md) documentation for more information.

MapMatcher methods

The methods of the *MapMatcher* class return an instance of [*requests.Response*](http://docs.python-requests.org/en/latest/api/#requests.Response).

In addition to the *json()* method that returns Python data parsed from the API, the responses provide a *geojson()* method that converts that data to a GeoJSON like form.

Usage

The Mapbox Map Matching API lets you take recorded GPS traces and snap them to the OpenStreetMap road and path network. This is helpful for aligning noisy traces and displaying them cleanly on a map.

The Map Matching API is limited to 60 requests per minute and results must be displayed on a Mapbox map using one of our SDKs. For high volume or other use cases, contact us.

```python >>> service = MapMatcher()
```

The input data to the Map Matcher must be a single GeoJSON-like Feature with a LineString geometry. The optional *coordTimes* property should be an array of the same length as the coordinates containing timestamps to help make the matching more accurate.

``` >>> line = { ... "type": "Feature", ... "properties": { ... "coordTimes": [ ... "2015-04-21T06:00:00Z", ... "2015-04-21T06:00:05Z", ... "2015-04-21T06:00:10Z", ... "2015-04-21T06:00:15Z", ... "2015-04-21T06:00:20Z"]}, ... "geometry": { ... "type": "LineString", ... "coordinates": [ ... [13.418946862220764, 52.50055852688439], ... [13.419011235237122, 52.50113000479732], ... [13.419756889343262, 52.50171780290061], ... [13.419885635375975, 52.50237416816131], ... [13.420631289482117, 52.50294888790448]]}}
```

Use the *match()* method to match the LineString to a profile.

```python >>> response = service.match(line, profile='mapbox.driving') >>> response.status_code 200 >>> response.headers['Content-Type'] 'application/json; charset=utf-8'
```

The response geojson contains a FeatureCollection with a single feature, with the new LineString corrected to match segments from the selected profile.
```

```python >>> corrected = response.geojson()['features'][0] >>> corrected['geometry']['type'] 'LineString' >>> corrected['geometry'] == line['geometry'] False >>> len(corrected['geometry']) == len(line['geometry']) True
```

See `import mapbox; help(mapbox.MapMatcher)` for more detailed usage.

Static API for Styles

```python >>> from mapbox import StaticStyle
```

See https://www.mapbox.com/api-documentation/maps/#static for general documentation of the API.

Your Mapbox access token should be set in your environment; see the [access tokens](access_tokens.md) documentation for more information.

StaticStyle methods

The methods of the *StaticStyle* class that provide access to the Static Maps API return an instance of [*requests.Response*](http://docs.python-requests.org/en/latest/api/#requests.Response). Its *content()* method returns the raw bytestring that can be saved into an image file with the appropriate extension.

Usage

To render a mapbox style to a static image, create a *StaticStyle* instance

```python >>> service = StaticStyle()
```

```python >>> response = service.image( ... username='mapbox', ... style_id='streets-v9', ... lon=-122.7282, lat=45.5801, zoom=12) >>> response.status_code 200 >>> response.headers['Content-Type'] 'image/png'
```

The contents can be written to file as an *image/png* file

```python >>> with open('/tmp/static.png', 'wb') as output: ... _ = output.write(response.content)
```

The result

Styles with raster data are delivered as *image/jpeg*

```python >>> response = service.image( ... username='mapbox', ... style_id='satellite-v9', ... lon=-122.7282, lat=45.5801, zoom=12) >>> response.status_code 200 >>> response.headers['Content-Type'] 'image/jpeg' >>> with open('/tmp/static.jpg', 'wb') as output: ... _ = output.write(response.content)
```


Because the styles are rendered by the GL engine (TODO link) we can specify pitch, bearing, and decimal zoom levels.

```python >>> response = service.image( ... username='mapbox', ... style_id='streets-v9', ... lon=-122.7282, lat=45.5801, zoom=12, ... pitch=45.0, bearing=277.5) >>> with open('/tmp/static_pitch.png', 'wb') as output: ... _ = output.write(response.content)
```

The result

Static maps can also display GeoJSON overlays and the [simplestyle-spec](https://github.com/mapbox/simplestyle-spec) styles will be respected and rendered.

```python >>> portland = { ... 'type': 'Feature', ... 'properties': {'name': 'Portland, OR'}, ... 'geometry': { ... 'type': 'Point', ... 'coordinates': [-122.7282, 45.5801]}} >>> bend = { ... 'type': 'Feature', ... 'properties': {'name': 'Bend, OR'}, ... 'geometry': { ... 'type': 'Point', ... 'coordinates': [-121.3153, 44.0582]}}
```

If features are provided the map image will be centered on them and will cover their extents.

```python >>> response = service.image( ... username='mapbox', ... style_id='streets-v9', ... features=[portland, bend]) >>> with open('/tmp/static_features.png', 'wb') as output: ... _ = output.write(response.content)
```

![todo image](static_features.png)

See `import mapbox; help(mapbox.StaticStyle)` for more detailed usage.

Tilequery

The *Tilequery* class provides access to the Mapbox Tilequery API. You can import it from either the *mapbox* module or the *mapbox.services.tilequery* module.

__mapbox__:

```python >>> from mapbox import Tilequery
```

__mapbox.services.tilequery__:

```python >>> from mapbox.services.tilequery import Tilequery
```

See https://www.mapbox.com/api-documentation/maps/#tilequery for general documentation of the API.

Use of the Tilequery API requires an access token, which you should set in your environment. For more information, see the [access tokens](access_tokens.md) documentation.

Tilequery Method

The public method of the *Tilequery* class provides access to the Tilequery API and returns an instance of [*requests.Response*](http://docs.python-requests.org/en/latest/api/#requests.Response).

Usage: Retrieving Features

Instantiate *Tilequery*.

```python >>> tilequery = Tilequery()
```

Call the *tilequery* method, passing in values for *map_id*, *lon*, and *lat*. Pass in values for optional arguments as necessary - *radius*, *limit*, *dedupe*, *geometry*, and *layers*.

```python >>> response = tilequery.tilequery("mapbox.mapbox-streets-v8", lon=0.0, lat=1.1)
```

Evaluate whether the request succeeded, and retrieve the features from the response object.

```python >>> if response.status_code == 200: ... features = response.json()
```

Maps

The *Maps* class provides access to the Mapbox Maps API. You can import it from either the *mapbox* module or the *mapbox.services.maps* module.

__mapbox__:

```python >>> from mapbox import Maps
```

__mapbox.services.maps__:

```python >>> from mapbox.services.maps import Maps
```

See https://www.mapbox.com/api-documentation/maps/#maps for general documentation of the API.

Use of the Maps API requires an access token, which you should set in your environment. For more information, see the [access tokens](access_tokens.md) documentation.

Maps Methods

The public methods of the *Maps* class provide access to the Maps API and return an instance of [*requests.Response*](http://docs.python-requests.org/en/latest/api/#requests.Response).

Usage: Retrieving Tiles

Instantiate *Maps*.

```python >>> maps = Maps()
```

Call the *tile* method, passing in values for *map_id*, *x* (column), *y* (row), and *z* (zoom level). (You may pass in individual values for x, y, and z or a Mapbox *mercantile tile*.) Pass in values for optional arguments as necessary - *retina* (double scale), *file_format*, *style_id*, and *timestamp*.

__x, y, and **z**__:

```python >>> response = maps.tile("mapbox.streets", 0, 0, 0)
```

__mercantile **tile**__:

```python >>> response = maps.tile("mapbox.streets", *mercantile.tile(0, 0, 0)) # doctest: +SKIP
```

Evaluate whether the request succeeded, and retrieve the tile from the response object.

```python >>> if response.status_code == 200: # doctest: +SKIP ... with open("./0.png", "wb") as output: ... output.write(response.content)
```

Usage: Retrieving Features from Mapbox Editor Projects

Instantiate *Maps*.

```python >>> maps = Maps()
```

Call the *features* method, passing in a value for *map_id*. Pass in a value for the optional argument, *feature_format*, as necessary.

```python >>> response = maps.features("mapbox.streets")
```

```

Evaluate whether the request succeeded, and retrieve the vector features from the response object. The approach will depend upon the format of the vector features.

__GeoJSON__:

```python >>> if response.status_code == 200: # doctest: +SKIP … features = response.json()
```

__KML__:

```python >>> if response.status_code == 200: # doctest: +SKIP … with open("./features.kml", "w") as output: … output.write(response.text)
```

Usage: Retrieving TileJSON Metadata

Instantiate *Maps*.

```python >>> maps = Maps()
```

Call the *metadata* method, passing in a value for *map_id*. Pass in a value for the optional argument, *secure*, as necessary.

```python >>> response = maps.metadata("mapbox.streets")
```

Evaluate whether the request succeeded, and retrieve the TileJSON metadata from the response object.

```python >>> if response.status_code == 200: … metadata = response.json() … >>> metadata['id'] 'mapbox.streets'
```

Usage: Retrieving a Standalone Marker

Instantiate *Maps*.

```python >>> maps = Maps()
```

Call the *marker* method, passing in a value for *marker_name*. Pass in values for optional arguments as necessary - *label*, *color*, and *retina*.

```python >>> response = maps.marker(marker_name="pin-s")
```

Evaluate whether the request succeeded, and retrieve the marker from the response object.

```python >>> if response.status_code == 200: # doctest: +SKIP … with open("pin-s.png", "wb") as output: … output.write(response.content)
```
```

5.1 mapbox package

5.1.1 Subpackages

mapbox.services package

Submodules

mapbox.services.analytics module

class mapbox.services.analytics.**Analytics**(*access_token=None*, *host=None*, *cache=None*)

Bases: *Service*

Access to Analytics API V1

Attributes

api_name
[str] The API's name.

api_version
[str] The API's version number.

valid_resource_types
[list] The possible values for the resource being requested.

Methods

analytics(resource_type, username[, id, ...])	Returns the request counts per day for a given resource and period.
handle_http_error(response[, ...])	Converts service errors to Python exceptions

analytics(*resource_type*, *username*, *id=None*, *start=None*, *end=None*)

Returns the request counts per day for a given resource and period.

Parameters

resource_type
[str] The resource being requested.

Possible values are "tokens", "styles", "tilesets", and "accounts".

username
[str] The username for the account that owns the resource.

id
[str, optional] The id for the resource.

If resource_type is "tokens", then id is the complete token. If resource_type is "styles", then id is the style id. If resource_type is "tilesets", then id is a map id. If resource_type is "accounts", then id is not required.

start, end
[str, optional] ISO-formatted start and end dates.

If provided, the start date must be earlier than the end date, and the maximum length of time between the start and end dates is one year.

If not provided, the length of time between the start and end dates defaults to 90 days.

Returns

requests.Response

`api_name = 'analytics'`

`api_version = 'v1'`

`valid_resource_types = ['tokens', 'styles', 'accounts', 'tilesets']`

mapbox.services.base module

Base Service class

`class mapbox.services.base.Service(access_token=None, host=None, cache=None)`

Bases: `object`

Service base class

Attributes

default_host
[str] Default service hostname: api.mapbox.com.

api_name
[str] Mapbox API name.

api_version
[str] API version string such as "v1" or "v5".

baseuri
The service's base URI

username
The username in the service's access token

Methods

handle_http_errors(response, custom_messages=None, raise_for_status=False)	Converts service errors to Python exceptions.

`api_name = 'hors service'`

`api_version = 'v0'`

`property baseuri`

The service's base URI

Returns

str

`default_host = 'api.mapbox.com'`

handle_http_error(*response*, *custom_messages=None*, *raise_for_status=False*)

Converts service errors to Python exceptions

Parameters

response
[requests.Response] A service response.

custom_messages
[dict, optional] A mapping of custom exception messages to HTTP status codes.

raise_for_status
[bool, optional] If True, the requests library provides Python exceptions.

Returns

None

property username

The username in the service's access token

Returns

str

mapbox.services.base.Session(*access_token=None*, *env=None*)

Create an HTTP session.

Parameters

access_token
[str] Mapbox access token string (optional).

env
[dict, optional] A dict that subsitutes for os.environ.

Returns

requests.Session

mapbox.services.datasets module

class mapbox.services.datasets.Datasets(*access_token=None*, *host=None*, *cache=None*)

Bases: *Service*

Access to the Datasets API V1

Attributes

api_name
[str] The API's name.

api_version
[str] The API's version number.

Methods

create([name, description])	Creates a new, empty dataset.
delete_dataset(dataset)	Deletes a single dataset, including all of the features that it contains.
delete_feature(dataset, fid)	Removes a feature from a dataset.
handle_http_error(response[, ...])	Converts service errors to Python exceptions
list()	Lists all datasets for a particular account.
list_features(dataset[, reverse, start, limit])	Lists features in a dataset.
read_dataset(dataset)	Retrieves (reads) a single dataset.
read_feature(dataset, fid)	Retrieves (reads) a feature in a dataset.
update_dataset(dataset[, name, description])	Updates a single dataset.
update_feature(dataset, fid, feature)	Inserts or updates a feature in a dataset.

api_name = 'datasets'

api_version = 'v1'

create(*name=None*, *description=None*)

Creates a new, empty dataset.

Parameters

name
[str, optional] The name of the dataset.

description
[str, optional] The description of the dataset.

Returns

request.Response
The response contains the properties of a new dataset as a JSON object.

delete_dataset(*dataset*)

Deletes a single dataset, including all of the features that it contains.

Parameters

dataset
[str] The dataset id.

Returns

HTTP status code.

delete_feature(*dataset*, *fid*)

Removes a feature from a dataset.

Parameters

dataset
[str] The dataset id.

fid
[str] The feature id.

Returns

HTTP status code.

`list`()

Lists all datasets for a particular account.

Returns

request.Response
The response contains a list of JSON objects describing datasets.

`list_features`(*dataset*, *reverse=False*, *start=None*, *limit=None*)

Lists features in a dataset.

Parameters

dataset
[str] The dataset id.

reverse
[str, optional] List features in reverse order.

Possible value is "true".

start
[str, optional] The id of the feature after which to start the list (pagination).

limit
[str, optional] The maximum number of features to list (pagination).

Returns

request.Response
The response contains the features of a dataset as a GeoJSON FeatureCollection.

`read_dataset`(*dataset*)

Retrieves (reads) a single dataset.

Parameters

dataset
[str] The dataset id.

Returns

request.Response
The response contains the properties of the retrieved dataset as a JSON object.

`read_feature`(*dataset*, *fid*)

Retrieves (reads) a feature in a dataset.

Parameters

dataset
[str] The dataset id.

fid
[str] The feature id.

Returns

request.Response
The response contains a GeoJSON representation of the feature.

`update_dataset`(*dataset*, *name=None*, *description=None*)

Updates a single dataset.

Parameters

dataset
[str] The dataset id.

name
[str, optional] The name of the dataset.

description
[str, optional] The description of the dataset.

Returns

request.Response
The response contains the properties of the updated dataset as a JSON object.

`update_feature`(*dataset*, *fid*, *feature*)

Inserts or updates a feature in a dataset.

Parameters

dataset
[str] The dataset id.

fid
[str] The feature id.

If the dataset has no feature with the given feature id, then a new feature will be created.

feature
[dict] The GeoJSON feature object.

This should be one individual GeoJSON feature and not a GeoJSON FeatureCollection.

Returns

request.Response
The response contains a GeoJSON representation of the new or updated feature.

mapbox.services.directions module

`class` mapbox.services.directions.`Directions`(*access_token=None*, *host=None*, *cache=None*)

Bases: *Service*

Access to the Directions v5 API.

Attributes

baseuri
The service's base URI

`username`
The username in the service's access token

Methods

directions(features[, profile, ...])	Request directions for waypoints encoded as GeoJSON features.
handle_http_error(response[, ...])	Converts service errors to Python exceptions

api_name = 'directions'

api_version = 'v5'

property baseuri

The service's base URI

Returns

str

directions(*features*, *profile='mapbox/driving'*, *alternatives=None*, *geometries=None*, *overview=None*, *steps=None*, *continue_straight=None*, *waypoint_snapping=None*, *annotations=None*, *language=None*, ***kwargs*)

Request directions for waypoints encoded as GeoJSON features.

Parameters

features
[iterable] An collection of GeoJSON features

profile
[str] Name of a Mapbox profile such as 'mapbox.driving'

alternatives
[bool] Whether to try to return alternative routes, default: False

geometries
[string] Type of geometry returned (geojson, polyline, polyline6)

overview
[string or False] Type of returned overview geometry: 'full', 'simplified', or False

steps
[bool] Whether to return steps and turn-by-turn instructions, default: False

continue_straight
[bool] Direction of travel when departing intermediate waypoints

radiuses
[iterable of numbers or 'unlimited'] Must be same length as features

waypoint_snapping
[list] Controls snapping of waypoints

The list is zipped with the features collection and must have the same length. Elements of the list must be one of:

- A number (interpretted as a snapping radius)
- The string 'unlimited' (unlimited snapping radius)
- A 3-element tuple consisting of (radius, angle, range)
- None (no snapping parameters specified for that waypoint)

annotations
[str] Whether or not to return additional metadata along the route

Possible values are: 'duration', 'distance', 'speed', and 'congestion'. Several annotations can be used by joining them with ','.

language
[str] Language of returned turn-by-turn text instructions, default: 'en'

Returns

requests.Response
The response object has a geojson() method for access to the route(s) as a GeoJSON-like FeatureCollection dictionary.

```
valid_annotations = ['duration', 'distance', 'speed']

valid_geom_encoding = ['geojson', 'polyline', 'polyline6']

valid_geom_overview = ['full', 'simplified', False]

valid_profiles = ['mapbox/driving', 'mapbox/driving-traffic', 'mapbox/walking',
'mapbox/cycling']
```

mapbox.services.geocoding module

class mapbox.services.geocoding.**Geocoder**(*name='mapbox.places'*, *access_token=None*, *cache=None*, *host=None*)

Bases: *Service*

Access to the Geocoding API V5

Attributes

baseuri
The service's base URI

country_codes
A list of valid country codes

place_types
A mapping of place type names to descriptions

username
The username in the service's access token

Methods

forward(address[, types, lon, lat, country, ...])	Returns a Requests response object that contains a GeoJSON collection of places matching the given address.
handle_http_error(response[, ...])	Converts service errors to Python exceptions
reverse(lon, lat[, types, limit])	Returns a Requests response object that contains a GeoJSON collection of places near the given longitude and latitude.

api_name = 'geocoding'

api_version = 'v5'

property country_codes

A list of valid country codes

forward(*address*, *types=None*, *lon=None*, *lat=None*, *country=None*, *bbox=None*, *limit=None*, *languages=None*)

Returns a Requests response object that contains a GeoJSON collection of places matching the given address.

response.geojson() returns the geocoding result as GeoJSON. *response.status_code* returns the HTTP API status code.

Place results may be constrained to those of one or more types or be biased toward a given longitude and latitude.

See: https://www.mapbox.com/api-documentation/search/#geocoding.

property place_types

A mapping of place type names to descriptions

precision = {'proximity': 3, 'reverse': 5}

reverse(*lon*, *lat*, *types=None*, *limit=None*)

Returns a Requests response object that contains a GeoJSON collection of places near the given longitude and latitude.

response.geojson() returns the geocoding result as GeoJSON. *response.status_code* returns the HTTP API status code.

See: https://www.mapbox.com/api-documentation/search/#reverse-geocoding.

mapbox.services.mapmatching module

class mapbox.services.mapmatching.**MapMatcher**(*access_token=None*, *host=None*, *cache=None*)

Bases: *Service*

Access to the Map Matching API V4

Attributes

baseuri
The service's base URI

username
The username in the service's access token

Methods

handle_http_error(response[, ...])	Converts service errors to Python exceptions
match(feature[, gps_precision, profile])	Match features to OpenStreetMap data.

api_name = 'matching'

api_version = 'v4'

match(*feature*, *gps_precision=None*, *profile='mapbox.driving'*)

Match features to OpenStreetMap data.

valid_profiles = ['mapbox.driving', 'mapbox.cycling', 'mapbox.walking']

mapbox.services.matrix module

Matrix API V1

class mapbox.services.matrix.**DirectionsMatrix**(*access_token=None*, *host=None*, *cache=None*)

Bases: *Service*

Access to the Matrix API V1

Attributes

baseuri
The service's base URI

username
The username in the service's access token

Methods

handle_http_error(response[, ...])	Converts service errors to Python exceptions
matrix(coordinates[, profile, sources, ...])	Request a directions matrix for trips between coordinates

api_name = 'directions-matrix'

api_version = 'v1'

property baseuri

The service's base URI

Returns

str

matrix(*coordinates*, *profile='mapbox/driving'*, *sources=None*, *destinations=None*, *annotations=None*)

Request a directions matrix for trips between coordinates

In the default case, the matrix returns a symmetric matrix, using all input coordinates as sources and destinations. You may also generate an asymmetric matrix, with only some coordinates as sources or destinations:

Parameters

coordinates
[sequence] A sequence of coordinates, which may be represented as GeoJSON features, GeoJSON geometries, or (longitude, latitude) pairs.

profile
[str] The trip travel mode. Valid modes are listed in the class's valid_profiles attribute.

annotations
[list] Used to specify the resulting matrices. Possible values are listed in the class's valid_annotations attribute.

sources
[list] Indices of source coordinates to include in the matrix. Default is all coordinates.

destinations
[list] Indices of destination coordinates to include in the matrix. Default is all coordinates.

Returns

requests.Response
Note: the directions matrix itself is obtained by calling the response's json() method. The resulting mapping has a code, the destinations and the sources, and depending of the annotations specified, it can also contain a durations matrix, a distances matrix or both of them (by default, only the durations matrix is provided).
code
[str] Status of the response

sources
[list] Results of snapping selected coordinates to the nearest addresses.

destinations
[list] Results of snapping selected coordinates to the nearest addresses.

durations
[list] An array of arrays representing the matrix in row-major order. durations[i][j] gives the travel time from the i-th source to the j-th destination. All values are in seconds. The duration between the same coordinate is always 0. If a duration can not be found, the result is null.

distances
[list] An array of arrays representing the matrix in row-major order. distances[i][j] gives the distance from the i-th source to the j-th destination. All values are in meters. The distance between the same coordinate is always 0. If a distance can not be found, the result is null.

```
valid_annotations = ['duration', 'distance']

valid_profiles = ['mapbox/driving', 'mapbox/cycling', 'mapbox/walking',
'mapbox/driving-traffic']
```

mapbox.services.static module

class mapbox.services.static.**Static**(*access_token=None*, *host=None*, *cache=None*)

Bases: *Service*

Access to the Static Map API V4

Attributes

api_name

baseuri

The service's base URI

username

The username in the service's access token

Methods

handle_http_error(response[, ...])	Converts service errors to Python exceptions

image	

api_name = None

api_version = 'v4'

property baseuri

The service's base URI

Returns

str

image(*mapid*, *lon=None*, *lat=None*, *z=None*, *features=None*, *width=600*, *height=600*, *image_format='png256'*, *sort_keys=False*, *retina=False*)

mapbox.services.static_style module

class mapbox.services.static_style.**StaticStyle**(*access_token=None*, *host=None*, *cache=None*)

Bases: *Service*

Access to the Static Map API V1

Attributes

baseuri

The service's base URI

username

The username in the service's access token

Methods

handle_http_error(response[, ...])	Converts service errors to Python exceptions
tile(username, style_id, z, x, y[, ...])	/styles/v1/{username}/{style_id}/tiles/{tileSize}/{z}/{x}/{y}{@2x}

image	
wmts	

api_name = 'styles'

api_version = 'v1'

image(*username*, *style_id*, *lon=None*, *lat=None*, *zoom=None*, *features=None*, *pitch=0*, *bearing=0*, *width=600*, *height=600*, *retina=None*, *sort_keys=False*, *attribution=None*, *logo=None*, *before_layer=None*, *twox=None*)

tile(*username*, *style_id*, *z*, *x*, *y*, *tile_size=512*, *retina=False*)

/styles/v1/{username}/{style_id}/tiles/{tileSize}/{z}/{x}/{y}{@2x}

wmts(*username*, *style_id*)

mapbox.services.static_style.**validate_bearing**(*val*)

mapbox.services.static_style.**validate_image_size**(*val*)

mapbox.services.static_style.**validate_lat**(*val*)

mapbox.services.static_style.**validate_lon**(*val*)

mapbox.services.static_style.**validate_overlay**(*val*)

mapbox.services.static_style.**validate_pitch**(*val*)

mapbox.services.tilequery module

The Tilequery class provides access to Mapbox's Tilequery API.

class mapbox.services.tilequery.**Tilequery**(*access_token=None*, *host=None*, *cache=None*)

Bases: *Service*

Access to Tilequery API V4

Attributes

api_name
[str] The API's name.

api_version
[str] The API's version number.

valid_geometries
[list] The possible values for geometry.

base_uri
[str] Forms base URI.

Methods

handle_http_error(response[, ...])	Converts service errors to Python exceptions
tilequery(map_id[, lon, lat, radius, limit, ...])	Returns data about specific features from a vector tileset.

api_name = 'tilequery'

api_version = 'v4'

property base_uri

Forms base URI.

tilequery(*map_id*, *lon=None*, *lat=None*, *radius=None*, *limit=None*, *dedupe=None*, *geometry=None*, *layers=None*)

Returns data about specific features from a vector tileset.

Parameters

map_id
[str or list] The tileset's unique identifier in the format username.id.

map_id may be either a str with one value or a list with multiple values.

lon
[float] The longitude to query, where -180 is the minimum value and 180 is the maximum value.

lat
[float] The latitude to query, where -85.0511 is the minimum value and 85.0511 is the maximum value.

radius
[int, optional] The approximate distance in meters to query, where 0 is the minimum value. (There is no maximum value.)

If None, the default value is 0.

limit
[int, optional] The number of features to return, where 1 is the minimum value and 50 is the maximum value.

If None, the default value is 5.

dedupe
[bool, optional] Whether to remove duplicate results.

If None, the default value is True.

geometry
[str, optional] The geometry type to query.

layers
[list, optional] The list of layers to query.

If a specified layer does not exist, then the Tilequery API will skip it. If no layers exist, then the API will return an empty GeoJSON FeatureCollection.

Returns

request.Response
The response object with a GeoJSON FeatureCollection of features at or near the specified longitude and latitude.

`valid_geometries = ['linestring', 'point', 'polygon']`

mapbox.services.uploads module

Mapbox Uploads API

`class mapbox.services.uploads.Uploader(access_token=None, host=None, cache=None)`

Bases: *Service*

Access to the Upload API V1

Example usage:

from mapbox import Uploader

u = Uploader() url = u.stage(open('test.tif', 'rb')) job = u.create(url, 'test1').json()

assert job in u.list().json()

... wait unti finished ... finished = u.status(job).json()['complete']

u.delete(job) assert job not in u.list().json()

Attributes

baseuri
The service's base URI

username
The username in the service's access token

Methods

create(stage_url, tileset[, name, patch, bypass])	Create a tileset
delete(upload[, account, username])	Delete the specified upload
handle_http_error(response[, ...])	Converts service errors to Python exceptions
list([account, username])	List of all uploads
stage(fileobj[, creds, callback])	Stages data in a Mapbox-owned S3 bucket
status(upload[, account, username])	Check status of upload
upload(fileobj, tileset[, name, patch, ...])	Upload data and create a Mapbox tileset

`api_name = 'uploads'`

`api_version = 'v1'`

`create(stage_url, tileset, name=None, patch=False, bypass=False)`

Create a tileset

Note: this step is refered to as "upload" in the API docs; This class's upload() method is a high-level function which acts like the Studio upload form.

Returns a response object where the json() contents are an upload dict. Completion of the tileset may take several seconds or minutes depending on size of the data. The status() method of this class may be used to poll the API endpoint for tileset creation status.

Parameters

stage_url: str
URL to resource on S3, typically provided in the response of this class's stage() method.

tileset: str
The id of the tileset set to be created. Username will be prefixed if not present. For example, 'my-tileset' becomes '{username}.my-tileset'.

name: str
A short name for the tileset that will appear in Mapbox studio.

patch: bool
Optional patch mode which requires a flag on the owner's account.

bypass: bool
Optional bypass validation mode for MBTiles which requires a flag on the owner's account.

Returns

requests.Response

`delete`(*upload*, *account=None*, *username=None*)

Delete the specified upload

Parameters

upload: str
The id of the upload or a dict with key 'id'.

username
[str] Account username, defaults to the service's username.

account
[str, **deprecated**] Alias for username. Will be removed in version 1.0.

Returns

requests.Response

`list`(*account=None*, *username=None*)

List of all uploads

Returns a Response object, the json() method of which returns a list of uploads

Parameters

username
[str] Account username, defaults to the service's username.

account
[str, **deprecated**] Alias for username. Will be removed in version 1.0.

Returns

requests.Response

`stage`(*fileobj*, *creds=None*, *callback=None*)

Stages data in a Mapbox-owned S3 bucket

If creds are not provided, temporary credentials will be generated using the Mapbox API.

Parameters

fileobj: file object or filename
A Python file object opened in binary mode or a filename.

creds: dict
AWS credentials allowing uploads to the destination bucket.

callback: func
A function that takes a number of bytes processed as its sole argument.

Returns

str
The URL of the staged data

`status`(*upload*, *account=None*, *username=None*)

Check status of upload

Parameters

upload: str
The id of the upload or a dict with key 'id'.

username
[str] Account username, defaults to the service's username.

account
[str, **deprecated**] Alias for username. Will be removed in version 1.0.

Returns

requests.Response

`upload`(*fileobj*, *tileset*, *name=None*, *patch=False*, *callback=None*, *bypass=False*)

Upload data and create a Mapbox tileset

Effectively replicates the Studio upload feature. Returns a Response object, the json() of which returns a dict with upload metadata.

Parameters

fileobj: file object or str
A filename or a Python file object opened in binary mode.

tileset: str
A tileset identifier such as '{owner}.my-tileset'.

name: str
A short name for the tileset that will appear in Mapbox studio.

patch: bool
Optional patch mode which requires a flag on the owner's account.

bypass: bool
Optional bypass validation mode for MBTiles which requires a flag on the owner's account.

callback: func
A function that takes a number of bytes processed as its sole argument. May be used with a progress bar.

Returns

requests.Response

Module contents

5.1.2 Submodules

5.1.3 mapbox.encoding module

mapbox.encoding.**encode_coordinates_json**(*features*)

Given an iterable of features return a JSON string to be used as the request body for the distance API: a JSON object, with a key coordinates, which has an array of [Longitude, Lattitude] pairs

mapbox.encoding.**encode_polyline**(*features*)

Encode and iterable of features as a polyline

mapbox.encoding.**encode_waypoints**(*features*, *min_limit=None*, *max_limit=None*, *precision=6*)

Given an iterable of features return a string encoded in waypoint-style used by certain mapbox APIs ("lon,lat" pairs separated by ";")

mapbox.encoding.**read_points**(*features*)

Iterable of features to a sequence of point tuples Where "features" can be either GeoJSON mappings or objects implementing the geo_interface

5.1.4 mapbox.errors module

exception mapbox.errors.**HTTPError**

Bases: *ValidationError*

exception mapbox.errors.**ImageSizeError**

Bases: *ValidationError*

exception mapbox.errors.**InputSizeError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidColorError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidColumnError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidCoordError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidCountryCodeError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidFeatureError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidFeatureFormatError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidFileError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidFileFormatError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidId**

Bases: *ValidationError*

exception mapbox.errors.**InvalidLabelError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidMarkerNameError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidParameterError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidPeriodError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidPlaceTypeError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidProfileError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidResourceTypeError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidRowError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidUsernameError**

Bases: *ValidationError*

exception mapbox.errors.**InvalidZoomError**

Bases: *ValidationError*

exception mapbox.errors.**MapboxDeprecationWarning**

Bases: UserWarning

exception mapbox.errors.**TokenError**

Bases: *ValidationError*

exception mapbox.errors.**ValidationError**

Bases: ValueError

5.1.5 mapbox.utils module

mapbox.utils.**normalize_geojson_featurecollection**(*obj*)

Takes a geojson-like mapping representing geometry, Feature or FeatureCollection (or a sequence of such objects) and returns a FeatureCollection-like dict

5.1.6 Module contents

5.2 mapbox

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

P

R

S

T

U

V

W