PyDriosm

An open-source tool for downloading, reading and PostgreSQL-based I/O of OpenStreetMap data.

Release 2.4.0

Qian Fu

Birmingham Energy Institute

University of Birmingham

First release: **September 2019**

Last updated: **April 2026**

Table of Contents

Chapter 1

About PyDriosm

PyDriosm is an open-source Python package designed to simplify the acquisition and management of OpenStreetMap (OSM) data. It provides automated utilities for downloading and parsing data extracts from Geofabrik and BBBike in several popular formats, including Protocolbuffer Binary Format (PBF), Shapefiles and GeoPackage (GPKG).

Beyond data retrieval, the package integrates a robust I/O interface for PostgreSQL databases. This allows users to import parsed OSM data directly into a relational database, facilitating complex spatial querying and efficient data manipulation. By handling the complexities of source scraping, file parsing and schema mapping, **pydriosm** provides a streamlined workflow for researchers and developers working with large-scale geographic datasets.

Core features:

- **Automated downloads**: Direct access to Geofabrik and BBBike subregion extracts.
- **Format support**: Parse regional data of PBF, Shapefiles and GeoPackage files into standard Pandas DataFrames or GeoDataFrames.
- **PostgreSQL integration**: Streamlined geometry I/O for efficient database storage.
- **Multi-region processing**: Tools for merging regional data layers into unified datasets.

Chapter 2

Installation

The easiest way to install the latest stable release of `pydriosm` is via pip:

```
pip install --upgrade pydriosm
```

To install the development version directly from GitHub:

```
pip install --upgrade git+https://github.com/mikeqfu/pydriosm.git@develop
```

2.1 Dependencies

`pydriosm` requires Python 3.12+ and several core geospatial libraries.

> **Note**
>
> **GDAL Installation**
>
> Parsing PBF data relies on GDAL. While `pip` will attempt to install it, GDAL often requires specific system binaries.
>
> - **Windows users:** If `pip install gdal` fails, it is highly recommended to use pre-built binary wheels. You can find them at the Geospatial library wheels for Python on Windows repository.
> - **Conda users:** Alternatively, installing via `conda install -c conda-forge gdal` often resolves binary dependency issues automatically.

> **Note**
>
> - **Environment:** It is strongly recommended to install `pydriosm` within a virtual environment.
> - **GeoPandas:** This package requires `geopandas >= 1.1.0`. If you have an older version of GeoPandas installed, `pip` will attempt to upgrade it to ensure compatibility with modern GeoPackage (GPKG) parsing.

- **Optimization:** To keep the installation lightweight, non-essential tools (e.g. PyShp) are excluded from the base dependencies. If a feature requires an uninstalled library, a `ModuleNotFoundError` will guide you on what to install.

2.2 Verification

To verify that `pydriosm` has been correctly installed, check the version in a Python interpreter:

```
>>> import pydriosm
>>> pydriosm.__version__  # Check the latest version
```

```
The latest version is: 2.4.0
```

Tip

For more general instructions on installing Python packages, refer to the official Installing Packages guide.

Chapter 3

Quick start

This guide provides a practical overview of how handles OpenStreetMap (OSM) data - specifically downloading, parsing, and performing storage I/O.

> **Note**
>
> - **Work directory:** All tutorial data is saved to `tests/osm_data/` in your current working directory. This folder will be created automatically.
> - **Cleanup:** At the end of the tutorial, you will be prompted to either **retain** or **remove** this directory. Please follow the prompt carefully to avoid accidentally deleting your data.

3.1 Download data

The current release of `pydriosm` supports subregion-based OSM data extracts from free download servers, including Geofabrik and BBBike.

To begin, we use the *`GeofabrikDownloader`* class to interface with the Geofabrik free download server.

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> # Initialize the downloader
>>> gfd = GeofabrikDownloader()

>>> gfd.LONG_NAME  # Name of the data
'Geofabrik OpenStreetMap data extracts'

>>> # View supported file formats
>>> gfd.FILE_FORMATS
{'.gpkg.zip', '.osm.bz2', '.osm.pbf', '.shp.zip'}
```

3.1.1 Exploring the catalogue

To see which regions are available for download, use the *GeofabrikDownloader.get_catalogue()* method:

```
>>> # The download catalogue for all available subregions
>>> geofabrik_download_catalogue = gfd.get_catalogue()
>>> geofabrik_download_catalogue.head()
               subregion  ... .osm.bz2
0                 Africa  ...     None
1             Antarctica  ...     None
2                   Asia  ...     None
3  Australia and Oceania  ...     None
4         Central America  ...     None
[5 rows x 7 columns]
```

3.1.2 Downloading a specific (sub)region

To download a Protocolbuffer Binary Format (PBF) file, specify the subregion name and the format (e.g. `".pbf"` or `".osm.pbf"`). Let's download the data for **London** and save it to our local directory:

```
>>> subregion_name = 'London'  # Name of a (sub)region; case-insensitive
>>> osm_file_format = ".pbf"  # OSM data file format
>>> download_dir = "tests/osm_data"  # Directory where the data is saved

>>> # This will prompt for confirmation before starting
>>> path_to_london_pbf = gfd.download_data(
...     subregion_names=subregion_name, osm_file_formats=osm_file_format,
...     download_dir=download_dir, ret_download_path=True, verbose=True)
Proceed to download data in the format '.osm.pbf' for the following geographic (sub)reg...
    "Greater London"
  to "./tests/osm_data/greater-london/"
? [No]|Yes: yes
Downloading "greater-london-latest.osm.pbf" 100%|■■■■■■■■■■■| 123M/123M | 30.4MB/s...
  Saving "greater-london-latest.osm.pbf" to "./tests/osm_data/greater-london/" ... Done.
```

After the downloading process completes, we can find the downloaded data file at `tests/osm_data/` and the (default) filename is `greater-london-latest.osm.pbf`.

Note

- **Confirmation:** By default, `confirmation_required=True`. Set it to `False` to skip the manual "yes/no" step.
- **Default paths:** If `download_dir=None`, the file is saved to a structured default path, e.g. `geofabrik/europe/united-kingdom/england/greater-london/`.
- **Downloaded files:** Set `ret_download_path=True` to return a list of absolute paths of the downloaded files.
- **Updates:** If a file already exists, it won't be re-downloaded unless you set `update=True`.

Check the file path and the filename of the downloaded data:

```
>>> import os

>>> path_to_london_pbf_ = path_to_london_pbf[0]

>>> # Relative file path:
>>> print(f'Current (relative) path: "{os.path.relpath(path_to_london_pbf_)}"')
Current (relative) path: "tests\osm_data\greater-london\greater-london-latest.osm.pbf"

>>> # Default filename:
>>> london_pbf_filename = os.path.basename(path_to_london_pbf_)
>>> print(f'Default filename: "{london_pbf_filename}"')
Default filename: "greater-london-latest.osm.pbf"
```

We could use the *get_default_pathname()* method to get the information (even if the file does not exist):

```
>>> download_info = gfd.get_valid_download_info(subrgn_name, file_format, dwnld_dir)
>>> subrgn_name_, london_pbf_filename, london_pbf_url, london_pbf_pathname = download_info

>>> print(f'Current (relative) path: "{os.path.relpath(london_pbf_pathname)}"')
Current (relative) path: "tests\osm_data\greater-london\greater-london-latest.osm.pbf"

>>> print(f'Default filename: "{london_pbf_filename}"')
Default filename: "greater-london-latest.osm.pbf"
```

In addition, we can also download the data of multiple (sub)regions at one go. For example, download the PBF data of both `'West Yorkshire'` and `'West Midlands'`, and return the file paths:

```
>>> subregion_names = ['West Yorkshire', 'West Midlands']
>>> paths_to_pbf = gfd.download_data(
...     subregion_names=subregion_names, osm_file_formats=osm_file_format,
...     download_dir=download_dir, ret_download_path=True, verbose=True)
Proceed to download data in the format '.osm.pbf' for the following geographic (sub)reg...
    "West Midlands"
    "West Yorkshire"
  to "./tests/osm_data/"
? [No]|Yes: yes
Downloading "west-yorkshire-latest.osm.pbf" 100%|■■■■■■■■■■■■| 51.0M/51.0M | 31.5MB/s...
  Saving "west-yorkshire-latest.osm.pbf" to "./tests/osm_data/west-yorkshire/" ... Done.
Downloading "west-midlands-latest.osm.pbf" 100%|■■■■■■■■■■■■| 58.4M/58.4M | 31.1MB/s ...
  Saving "west-midlands-latest.osm.pbf" to "./tests/osm_data/west-midlands/" ... Done.
```

Check the pathnames of the data files:

```
>>> for path_to_pbf in paths_to_pbf:
...     print(f"\"{os.path.relpath(path_to_pbf)}\"")
"tests\osm_data\west-yorkshire\west-yorkshire-latest.osm.pbf"
"tests\osm_data\west-midlands\west-midlands-latest.osm.pbf"
```

3.2 Read and parse data

Once downloaded, we can parse OSM data into Python objects using *GeofabrikReader*. This class utilizes GDAL for parsing.

3.2.1 Parsing PBF data

Let's read the Rutland subregion. If the file isn't found locally, the *read_pbf()* method can automatically download it for you:

```
>>> from pydriosm.reader import GeofabrikReader

>>> # Initialize the reader
>>> gfr = GeofabrikReader()

>>> subregion_name = 'Rutland'
>>> data_dir = download_dir  # i.e. "tests/osm_data"

>>> # Read raw features (as GDAL Feature objects)
>>> rutland_pbf_raw = gfr.read_pbf(
...     subregion_name=subregion_name, data_dir=data_dir, verbose=True)
Downloading "rutland-latest.osm.pbf" 100%|■■■■■■■■■■■| 1.89M/1.89M | 1.76MB/s | ...
  Saving "rutland-latest.osm.pbf" to "./tests/osm_data/rutland/" ... Done.
Reading "./tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
```

Check the data types:

```
>>> raw_data_type = type(rutland_pbf_raw)
>>> print(f'Data type of `rutland_pbf_parsed`:\n  {raw_data_type}')
Data type of `rutland_pbf_parsed`:
  <class 'dict'>

>>> raw_data_keys = list(rutland_pbf_raw.keys())
>>> print(f'The "keys" of `rutland_pbf_parsed`:\n  {raw_data_keys}')
The "keys" of `rutland_pbf_parsed`:
  ['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> raw_layer_data_type = type(rutland_pbf_raw['points'])
>>> print(f'Data type of the corresponding layer:\n  {raw_layer_data_type}')
Data type of the corresponding layer:
  <class 'list'>

>>> raw_value_type = type(rutland_pbf_raw['points'][0])
>>> print(f'Data type of the individual feature:\n  {raw_value_type}')
Data type of the individual feature:
  <class 'osgeo.ogr.Feature'>
```

The resulting dictionary contains five layers: `'points'`, `'lines'`, `'multilinestrings'`, `'multipolygons'`, and `'other_relations'`.

Note

- **Performance:** The *read_pbf()* method may take tens of minutes (or even much longer) to parse a PBF data file, depending on the size of the data file.
- **Large data:** If the size of a PBF data file is greater than the specified `chunk_size_limit` (default: 50 MB), the data will be parsed in a chunk-wise manner.

3.2.2 Make raw PBF readable

Raw GDAL features are not easily manipulated in Python. Set `readable=True` to parse them into standard Python dictionaries (GeoJSON-like) or `expand=True` to convert them into a **Pandas DataFrame**.

```
>>> # Parse into a DataFrame with geometry objects
>>> rutland_pbf_parsed_0 = gfr.read_pbf(
...     subregion_name=subregion_name, data_dir=data_dir, readable=True, verbose=True)
Parsing "./tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
```

Check the data types:

```
>>> parsed_data_type = type(rutland_pbf_parsed_0)
>>> print(f'Data type of `rutland_pbf_parsed`:\n  {parsed_data_type}')
Data type of `rutland_pbf_parsed`:
  <class 'dict'>

>>> parsed_data_keys = list(rutland_pbf_parsed_0.keys())
>>> print(f'The "keys" of `rutland_pbf_parsed`:\n  {parsed_data_keys}')
The "keys" of `rutland_pbf_parsed`:
  ['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> parsed_layer_type = type(rutland_pbf_parsed_0['points'])
>>> print(f'Data type of the corresponding layer:\n  {parsed_layer_type}')
Data type of the corresponding layer:
  <class 'pandas.Series'>
```

Let's take a look at the `'points'` layer as an example:

```
>>> rutland_pbf_points_0 = rutland_pbf_parsed_0['points']  # The layer of 'points'
>>> rutland_pbf_points_0.head()
0    {'type': 'Feature', 'geometry': {'type': 'Poin...
1    {'type': 'Feature', 'geometry': {'type': 'Poin...
2    {'type': 'Feature', 'geometry': {'type': 'Poin...
3    {'type': 'Feature', 'geometry': {'type': 'Poin...
4    {'type': 'Feature', 'geometry': {'type': 'Poin...
Name: points, dtype: object

>>> rutland_pbf_points_0_3 = rutland_pbf_points_0[3]  # A feature of the 'points' layer
>>> rutland_pbf_points_0_3
{'type': 'Feature',
 'geometry': {'type': 'Point', 'coordinates': [-0.7266543, 52.669517]},
 'properties': {'osm_id': '14558402',
  'name': None,
  'barrier': None,
```

(continues on next page)

(continued from previous page)

```
  'highway': 'mini_roundabout',
  'ref': None,
  'address': None,
  'is_in': None,
  'place': None,
  'man_made': None,
  'other_tags': '"direction"=>"clockwise"'},
 'id': 14558402}
```

Each row (or, a feature) of `rutland_pbf_points_0` is GeoJSON data, which is a nested dictionary.

The charts (Figure 1 - Figure 5) below illustrate the different geometry types and structures (i.e. all keys within the corresponding GeoJSON data) for each layer:

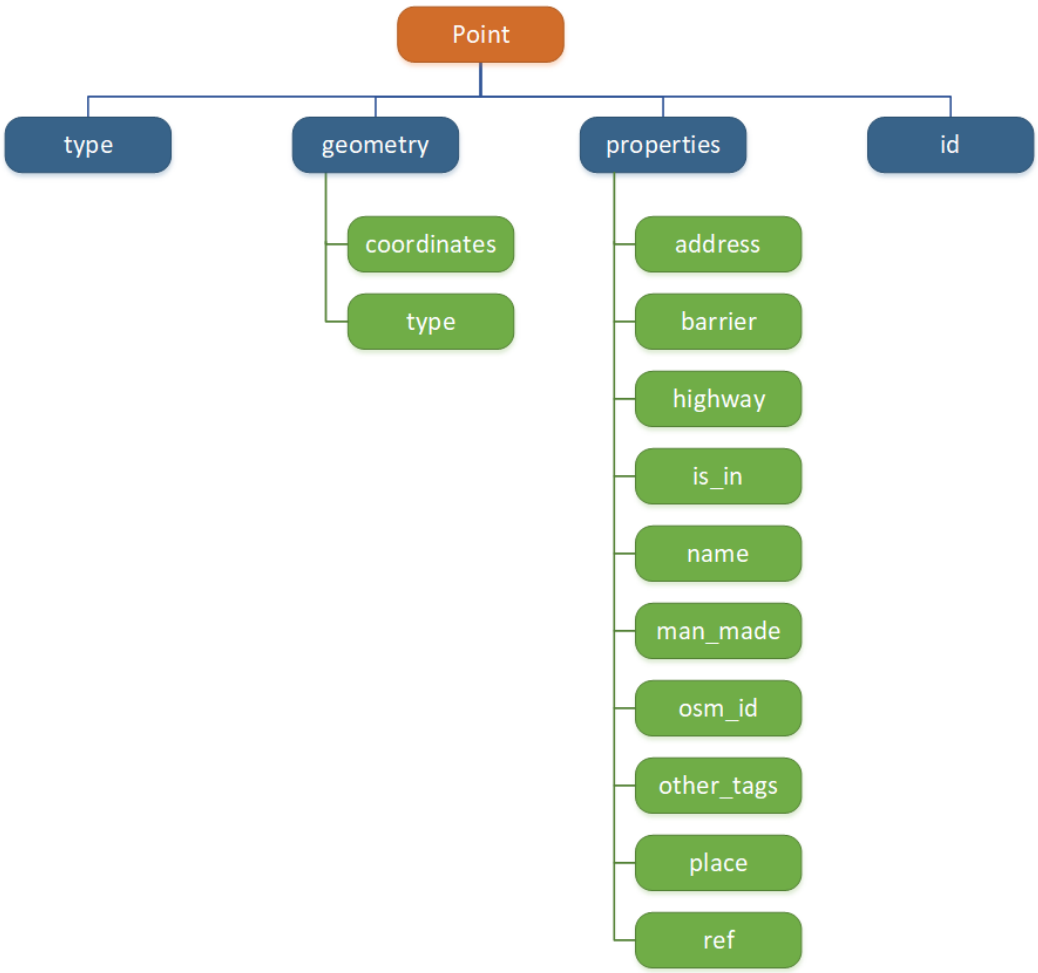

Figure 1: Type of the geometry object and keys within the nested dictionary of `'points'`.

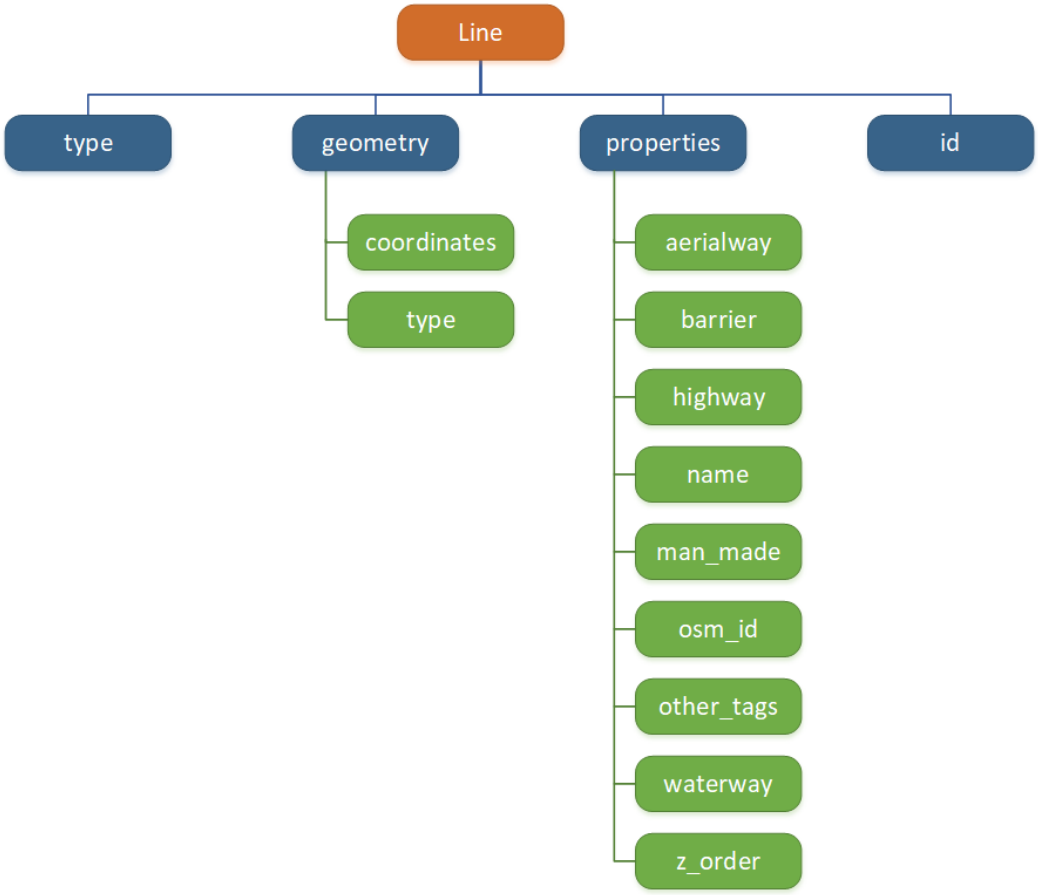

Figure 2: Type of the geometry object and keys within the nested dictionary of `'lines'`.

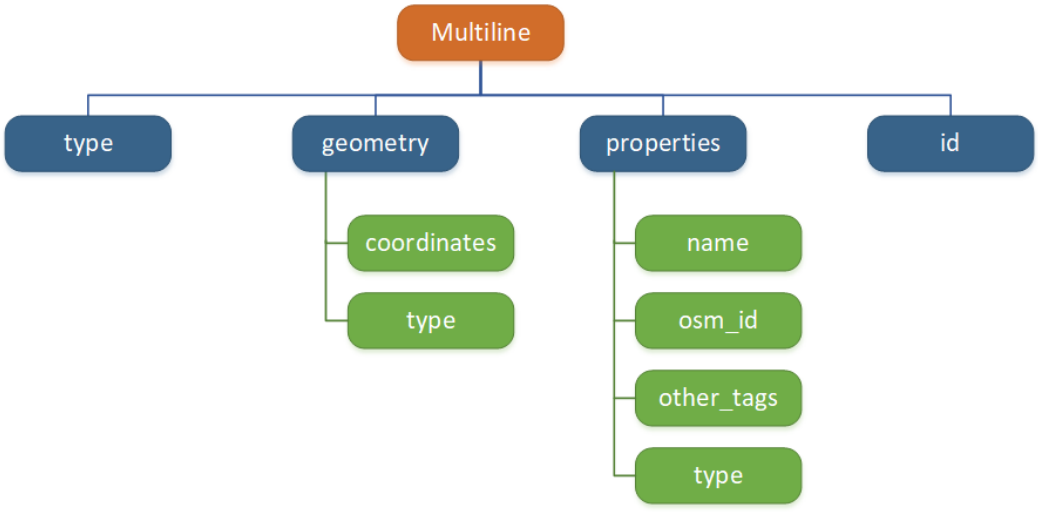

Figure 3: Type of the geometry object and keys within the nested dictionary of `'multilinestrings'`.

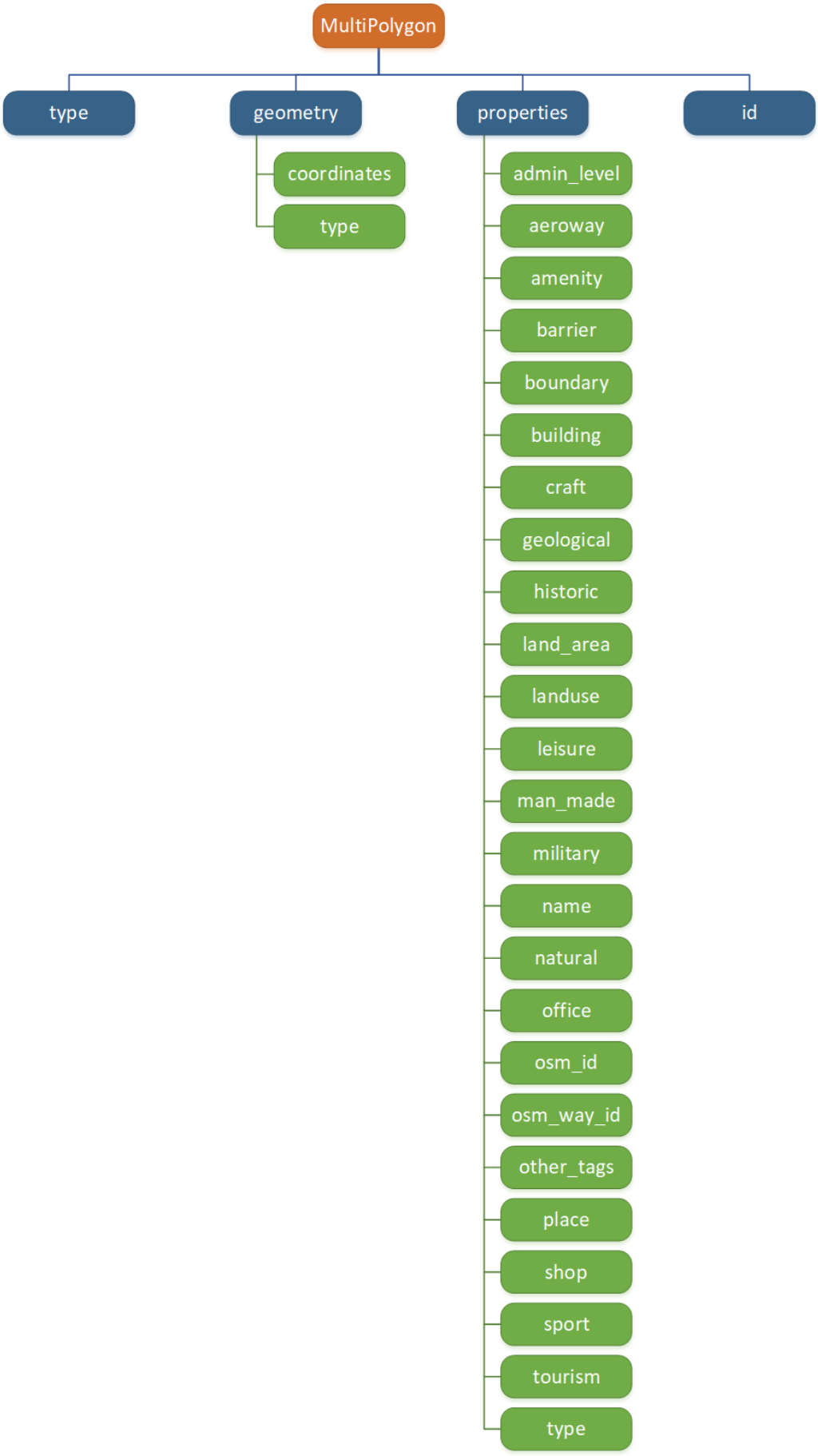

Figure 4: Type of the geometry object and keys within the nested dictionary of `'multipolygons'`.

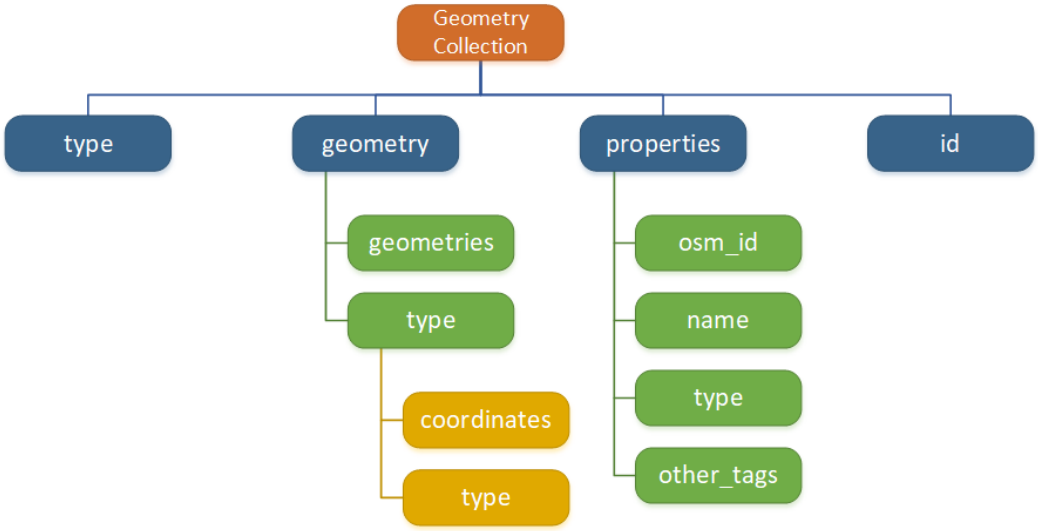

Figure 5: Type of the geometry object and keys within the nested dictionary of `'other_relations'`.

If we set `expand=True`, we can transform the GeoJSON records to dataframe and obtain data of 'visually' (though not virtually) higher level of granularity (*see also how to import the data into a PostgreSQL database*):

```
>>> rutland_pbf_parsed_1 = gfr.read_pbf(
...     subregion_name=subregion_name, data_dir=data_dir, expand=True, verbose=True)
Parsing "./tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
```

Data of the expanded `'points'` layer (*see also the retrieved data from database*):

```
>>> rutland_pbf_points_1 = rutland_pbf_parsed_1['points']
>>> rutland_pbf_points_1.head()
        id  ...                                         properties
0   488658  ...  {'osm_id': '488658', 'name': 'Tickencote Inter...
1  13883868  ...  {'osm_id': '13883868', 'name': None, 'barrier'...
2  14049101  ...  {'osm_id': '14049101', 'name': None, 'barrier'...
3  14558402  ...  {'osm_id': '14558402', 'name': None, 'barrier'...
4  14558409  ...  {'osm_id': '14558409', 'name': None, 'barrier'...
[5 rows x 3 columns]

>>> rutland_pbf_points_1['geometry'].head()
0    {'type': 'Point', 'coordinates': [-0.5313354, ...
1    {'type': 'Point', 'coordinates': [-0.7229332, ...
2    {'type': 'Point', 'coordinates': [-0.7249816, ...
3    {'type': 'Point', 'coordinates': [-0.7266543, ...
4    {'type': 'Point', 'coordinates': [-0.7287807, ...
Name: geometry, dtype: object
```

The data can be further transformed/parsed via three more parameters: `parse_geometry`, `parse_other_tags` and `parse_properties`, which all default to `False`.

For example, let's now try `expand=True` and `parse_geometry=True`:

```
>>> rutland_pbf_parsed_2 = gfr.read_pbf(
...     subregion_name=subregion_name, data_dir=data_dir, expand=True,
```

(continues on next page)

(continued from previous page)

```
...     parse_geometry=True, verbose=True)
Parsing "./tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.

>>> rutland_pbf_points_2 = rutland_pbf_parsed_2['points']
>>> rutland_pbf_points_2.head()
         id  ...                                         properties
0    488658  ...  {'osm_id': '488658', 'name': 'Tickencote Inter...
1  13883868  ...  {'osm_id': '13883868', 'name': None, 'barrier'...
2  14049101  ...  {'osm_id': '14049101', 'name': None, 'barrier'...
3  14558402  ...  {'osm_id': '14558402', 'name': None, 'barrier'...
4  14558409  ...  {'osm_id': '14558409', 'name': None, 'barrier'...
[5 rows x 3 columns]

>>> rutland_pbf_points_2['geometry'].head()
0    POINT (-0.5313354 52.6737716)
1    POINT (-0.7229332 52.5889864)
2    POINT (-0.7249816 52.6748426)
3     POINT (-0.7266543 52.669517)
4    POINT (-0.7287807 52.6696427)
Name: geometry, dtype: object
```

We can see the difference in `'geometry'` column between `rutland_pbf_points_1` and `rutland_pbf_points_2`.

Note

- If only the name of a (sub)region is provided, e.g. `rutland_pbf = gfr.read_pbf(subregion_name='Rutland')`, the method will go to look for the data file at the default file path. Otherwise, you need to specify `data_dir` where the data file is.
- If the data file does not exist at the default or specified directory, the method will by default try to download it first. To give up downloading the data, setting `download=False`.
- When `pickle_it=True`, the parsed data will be saved as a Pickle file. When you run the method next time, it will try to load the Pickle file first, provided that `update=False` (default); if `update=True`, the method will try to download and parse the latest version of the data file. Note that `pickle_it=True` works only when `readable=True` and/or `expand=True`.

3.2.3 Parsing Shapefiles

To demonstrate reading OSM Shapefile data, we switch to the BBBike server. We use the *`read_shp()`* method, which utilizes GeoPandas to return data as a GeoDataFrame.

Note

- PyShp is not required for the installation of pydriosm.

For example, let's now try to read the `'railways'` layer of the shapefile of `'London'` by using *`BBBikeReader.read_shp()`*:

```
>>> from pydriosm.reader import BBBikeReader

>>> bbr = BBBikeReader()

>>> subregion_name = 'London'
>>> layer_name = 'railways'

>>> # Attempt to read
>>> london_shp = bbr.read_shp(
...     subregion_name=subregion_name, layer_names=layer_name, data_dir=data_dir,
...     verbose=True)
Traceback (most recent call last):
  ...
FileNotFoundError: The shapefile "London.osm.shp.zip" is not available.
  Set `download=True` to download it.

>>> # If the file is missing, set download=True
>>> london_shp = bbr.read_shp(
...     subregion_name=subregion_name, layer_names=layer_name, data_dir=data_dir,
...     download=True, verbose=True)
Downloading "London.osm.shp.zip" 100%|■■■■■■■■■■| 248M/248M | 32.8MB/s
  Saving "London.osm.shp.zip" to "./tests/osm_data/london/" ......
Extracting the following layer(s):
    'railways'
  from: "./tests/osm_data/london/London.osm.shp.zip" ...
    to: "./tests/osm_data/london/" ... Done.
Reading "./tests/osm_data/london/London-shp/shape/railways.shp" ... Done.
```

Check the data:

```
>>> data_type = type(london_shp)
>>> print(f'Data type of `london_shp`:\n  {data_type}')
Data type of `london_shp`:
  <class 'dict'>

>>> data_keys = list(london_shp.keys())
>>> print(f'The "keys" of `london_shp`:\n  {data_keys}')
The "keys" of `london_shp`:
  ['railways']

>>> layer_type = type(london_shp[lyr_name])
>>> print(f"Data type of the '{lyr_name}' layer:\n  {layer_type}")
Data type of the 'railways' layer:
  <class 'geopandas.geodataframe.GeoDataFrame'>
```

Similar to the parsed PBF data, `london_shp` is also a dictionary with the `layer_name` being its key by default.

```
>>> london_railways_shp = london_shp[layer_name]  # london_shp['railways']
>>> london_railways_shp.head()
   osm_id  ...                                           geometry
0   30804  ...     LINESTRING (0.00486 51.62793, 0.0062 51.62927)
1  101298  ...  LINESTRING (-0.22499 51.4937, -0.22516 51.4945...
2  101486  ...  LINESTRING (-0.20555 51.51954, -0.20514 51.519...
3  101511  ...  LINESTRING (-0.2119 51.52419, -0.21081 51.5239...
4  282898  ...   LINESTRING (-0.1862 51.61592, -0.18687 51.61386)
[5 rows x 4 columns]
```

Note

- **Layer selection:** If `layer_names=None` (default), all available layers in the shapefile will be read.
- **Automatic workflow:** The reader is "smart" - it will try to find the `.shp` file first, then look for a `.shp.zip` to extract from, and finally download from the server if `download=True`.
- **Cleanup:** You can automatically delete intermediate files after reading by setting `rm_extracts=True` and/or `rm_shp_zip=True`.

3.2.4 Merging subregion shapefiles

If you need to analyze multiple regions together, you can merge layers from different subregions into a single Shapefile using *`merge_shp_layers()`*.

For example, let's merge the railways of **London** and **Birmingham**:

```
>>> subregion_names = ['London', 'Birmingham']
>>> layer_name = 'railways'

>>> path_to_merged_shp = bbr.merge_shp_layers(
...     subregion_names=subregion_names, layer_name=layer_name, data_dir=data_dir,
...     ret_merged_shp_path=True, verbose=True)
"London.osm.shp.zip" already exists in "./tests/osm_data/london/".
Proceed to download data in the format '.shp.zip' for the following geographic (sub)reg...
    "Birmingham"
  to "./tests/osm_data/"
? [No]|Yes: yes
Downloading "Birmingham.osm.shp.zip" 100%|■■■■■■■■■■■| 79.1M/79.1M | 18.1MB/s | ETA...
  Saving "Birmingham.osm.shp.zip" to "./tests/osm_data/birmingham/" ... Done.
Merging the following shapefiles:
  "london_railways.shp"
  "birmingham_railways.shp"
  In progress ... Done.
    Find the merged shapefile in "./tests/osm_data/lon-bir-railways/".

>>> # Relative path of the merged shapefile
>>> print(f"\"{os.path.relpath(path_to_merged_shp[0])}\"")
"tests\osm_data\lon-bir-railways\lon-bir-railways.shp"
```

We can then read this merged data back into Python using *`SHP.read_shp()`* or *`SHP.read_layer_shps()`*, or use the internal `SHP` utility:

```
>>> # Optional
>>> # from pydriosm.reader import SHP

>>> lon_bir_railways = bbr.SHP.read_layer_shps(path_to_merged_shp)
>>> lon_bir_railways.head()
   osm_id  ...                                           geometry
0   30804  ...     LINESTRING (0.00486 51.62793, 0.0062 51.62927)
1  101298  ...  LINESTRING (-0.22499 51.4937, -0.22516 51.4945...
2  101486  ...  LINESTRING (-0.20555 51.51954, -0.20514 51.519...
3  101511  ...  LINESTRING (-0.2119 51.52419, -0.21081 51.5239...
4  282898  ...   LINESTRING (-0.1862 51.61592, -0.18687 51.61386)
[5 rows x 4 columns]
```

For more details, also check out *SHP.merge_shps()* and *SHP.merge_layers()*.

3.3 Import data into / fetch data from a PostgreSQL server

After downloading and reading the OSM data, PyDriosm further provides a practical solution - the module *pydriosm.ios* - to managing the storage I/O of the data through database. Specifically, the class *PostgresOSM*, which inherits from pyhelpers.dbms.PostgreSQL, can assist us with importing the OSM data into, and retrieving it from, a PostgreSQL server. To establish a connection with a PostgreSQL server, we need to specify the host address, port, username, password and a database name of the server. For example, let's connect/create to a database named `'osmdb_test'` in a local PostgreSQL server (as is installed with the default configuration):

```
>>> from pydriosm.ios import PostgresOSM

>>> host = 'localhost'
>>> port = 5432
>>> username = 'postgres'
>>> password = None  # You need to type it in manually if `password=None`
>>> database_name = 'osmdb_test'

>>> # Create an instance of a running PostgreSQL server
>>> osmdb = PostgresOSM(
...     host=host, port=port, username=username, password=password,
...     database_name=database_name, data_source='Geofabrik', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.
```

The example is illustrated in Figure 6:

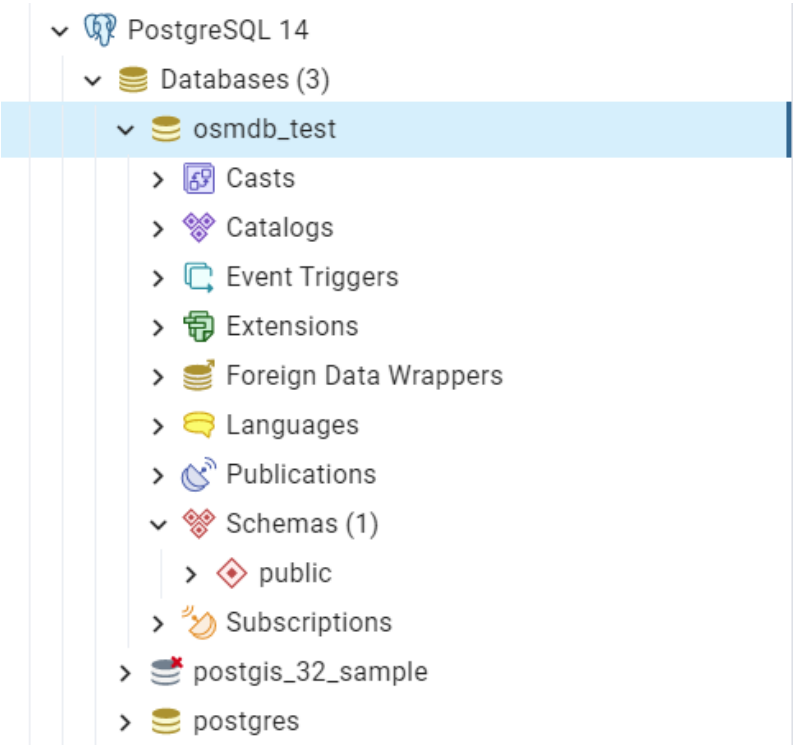

Figure 6: An illustration of the database named *'osmdb_test'*.

Note

- The parameter `password` is by default `None`. If we don't specify a password for creating an instance, we'll need to manually type in the password to the PostgreSQL server.
- The class *`PostgresOSM`* incorporates the classes for downloading and reading OSM data from the modules *`downloader`* and *`reader`* as properties. In the case of the above instance, `osmdb.downloader` is equivalent to the class *`GeofabrikDownloader`*, as the parameter `data_source='Geofabrik'` by default.
- To relate the instance `osmdb_test` to BBBike data, we could just run `osmdb.data_source = 'BBBike'`.
- See also the example of *reading Birmingham shapefile data*.

3.3.1 Import data into the database

To import any of the above OSM data to a database in the connected PostgreSQL server, we can use the method *`import_osm_data()`*.

For example, let's now try to import `rutland_pbf_parsed_1` (*see also the parsed PBF data of Rutland above* that we've got from previous *PBF data (.pbf / .osm.pbf)* section:

```
>>> subregion_name = 'Rutland'

>>> osmdb.import_osm_data(
```

(continues on next page)

(continued from previous page)

```
...     osm_data=rutland_pbf_parsed_1, table_name=subregion_name, schema_names=None,
...     verbose=True)
Proceed to import data into the table "Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
  "points" ... Done: <total of rows> features.
  "lines" ... Done: <total of rows> features.
  "multilinestrings" ... Done: <total of rows> features.
  "multipolygons" ... Done: <total of rows> features.
  "other_relations" ... Done: <total of rows> features.
```

Note

- The parameter `schema_names` is by default `None`, meaning that we import all the available layers of the PBF data into the database.

In the example above, the schemas are *'points'*, *'lines'*, *'multilinestrings'*, *'multipolygons'* and *'other_relations'*. If they do not exist, they are created in the database *'osmdb_test'* when running the method *`import_osm_data()`*. Each of the schemas corresponds to a key (i.e. name of a layer) of `rutland_pbf_parsed_1` (as illustrated in Figure 7); the data of each layer is imported into a table named as "Rutland" under the corresponding schema (as illustrated in Figure 8).

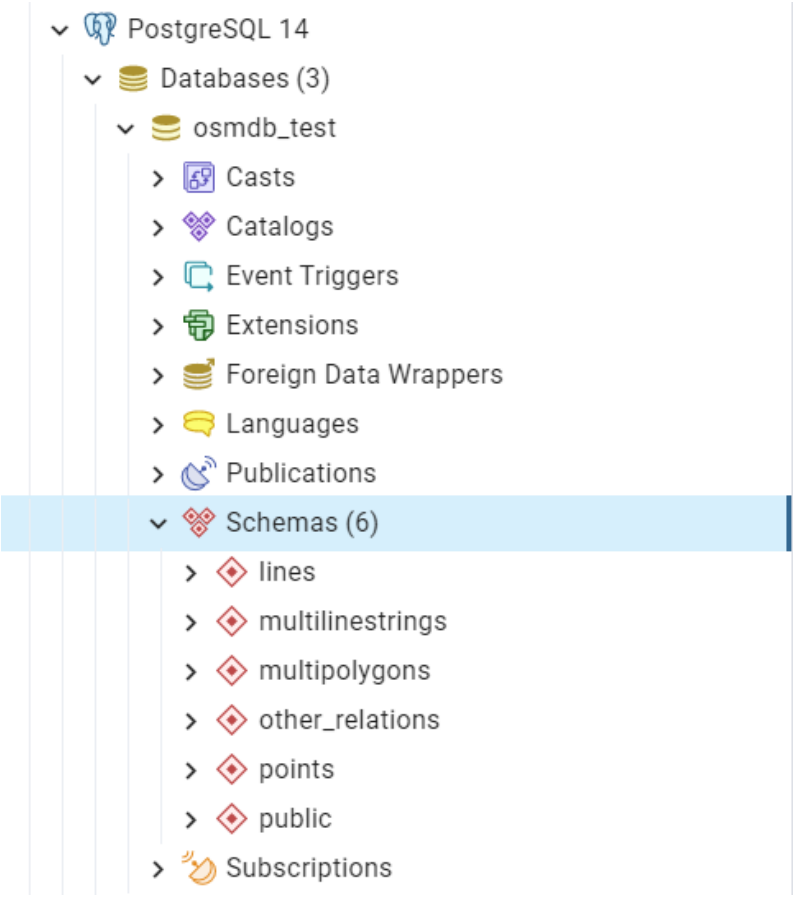

Figure 7: An illustration of schemas for importing OSM PBF data into a PostgreSQL database.

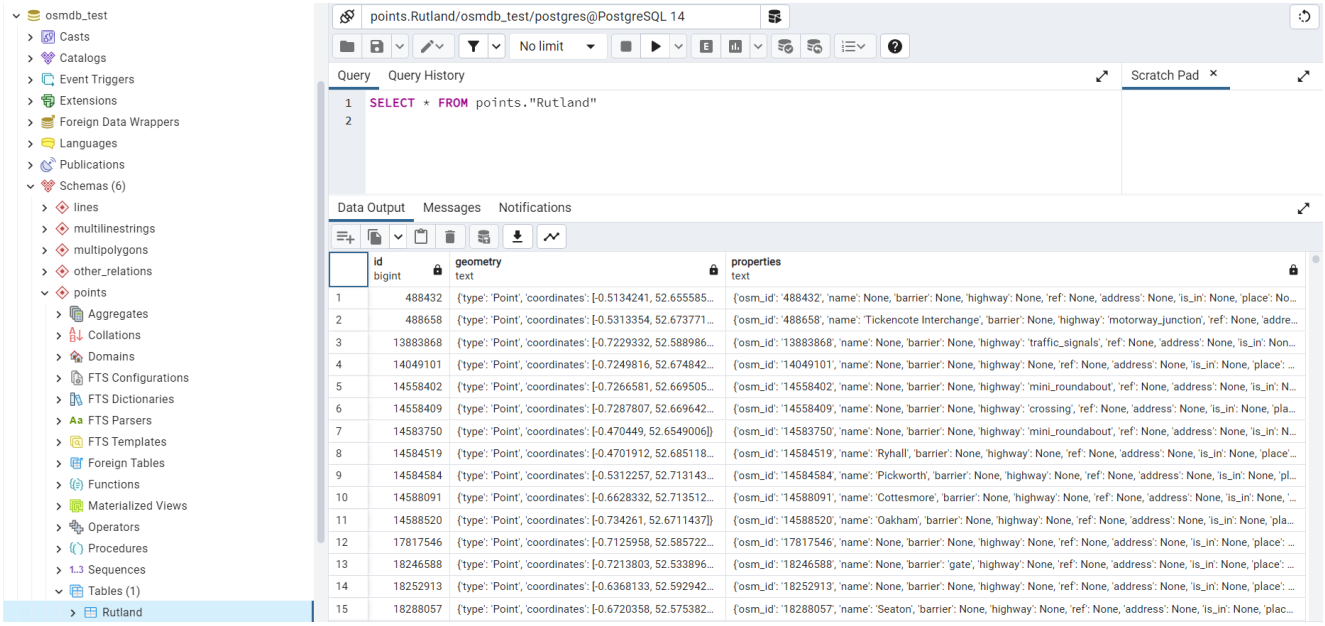

Figure 8: An illustration of table name for storing the *'points'* layer of the OSM PBF data of Rutland.

3.3.2 Fetch data from the database

To retrieve all or specific layers of the imported data, we can use the `fetch_data()` method:

```
>>> # Retrieve all the PBF data being just imported
>>> rutland_pbf_parsed_1_ = osmdb.fetch_data(subregion_name, verbose=True)
Fetching the data of "Rutland" ...
  "points" ... Done.
  "lines" ... Done.
  "multilinestrings" ... Done.
  "multipolygons" ... Done.
  "other_relations" ... Done.
```

Check the data `rutland_pbf_parsed_1_` we just retrieved:

```
>>> retr_data_type = type(rutland_pbf_parsed_1_)
>>> print(f'Data type of `rutland_pbf_parsed_1_`:\n  {retr_data_type}')
Data type of `rutland_pbf_parsed_1_`:
  <class 'dict'>

>>> retr_data_keys = list(rutland_pbf_parsed_1_.keys())
>>> print(f'The "keys" of `rutland_pbf_parsed_1_`:\n  {retr_data_keys}')
The "keys" of `rutland_pbf_parsed_1_`:
  ['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> retr_layer_type = type(rutland_pbf_parsed_1_['points'])
>>> print(f'Data type of the corresponding layer:\n  {retr_layer_type}')
Data type of the corresponding layer:
  <class 'pandas.DataFrame'>
```

Take a quick look at the data of the *'points'*:

```
>>> rutland_pbf_parsed_1_points_ = rutland_pbf_parsed_1_['points']
>>> rutland_pbf_parsed_1_points_.head()
         id  ...                                         properties
0    488658  ...  {'osm_id': '488658', 'name': 'Tickencote Inter...
1  13883868  ...  {'osm_id': '13883868', 'name': None, 'barrier'...
2  14049101  ...  {'osm_id': '14049101', 'name': None, 'barrier'...
3  14558402  ...  {'osm_id': '14558402', 'name': None, 'barrier'...
4  14558409  ...  {'osm_id': '14558409', 'name': None, 'barrier'...
[5 rows x 3 columns]
```

Check whether `rutland_pbf_parsed_1_` is equal to `rutland_pbf_parsed_1` (see also *the parsed data*):

```
>>> # Check each of the layers:
>>> #   'points', 'lines', 'multilinestrings', 'multipolygons' or 'other_relations'
>>> check_equivalence = all(
...     rutland_pbf_parsed_1[lyr_name].equals(rutland_pbf_parsed_1_[lyr_name])
...     for lyr_name in rutland_pbf_parsed_1.keys())
>>> print(f"`rutland_pbf_parsed_1_` is equivalent to `rutland_pbf_parsed_1`: "
...       f"{check_equivalence}")
`rutland_pbf_parsed_1_` is equivalent to `rutland_pbf_parsed_1`: True
```

Note

- The parameter `layer_names` is `None` by default, meaning that we fetch data of all layers available from the database.
- The data stored in the database was parsed by the method *`GeofabrikReader.read_pbf()`* given `expand=True` (see *the parsed data*). When it is being imported in the PostgreSQL server, the data type of the column `'coordinates'` is converted from list to str. Therefore, to retrieve the same data in the above example for the method *`fetch_data()`*, the parameter `decode_geojson` is by default `True`.

3.3.3 Specific layers of shapefile

Below is another example of importing/fetching data of multiple layers in a customised order. Let's firstly import the transport-related layers of Leeds shapefile data.

Note

- `'Leeds'` is not listed on the free download catalogue of Geofabrik but that of BBBike. We need to change the data source to `'BBBike'` for the instance `osmdb` (see also the *note above*).

```
>>> osmdb.data_source = 'BBBike'  # Change to 'BBBike'

>>> subregion_name = 'Leeds'

>>> leeds_shp = osmdb.reader.read_shp(
...     subregion_name=subregion_name, data_dir=data_dir, download=True, verbose=True)
Downloading "Leeds.osm.shp.zip" 100%|■■■■■■■■■■| 57.8M/57.8M | 18.0MB/s | ETA: 00:00
```

(continues on next page)

(continued from previous page)

```
  Saving "Leeds.osm.shp.zip" to "./tests/osm_data/leeds/" ... Done.
Extracting "./tests/osm_data/leeds/Leeds.osm.shp.zip"
  to "./tests/osm_data/leeds/" ... Done.
Reading the shapefile(s) at "./tests/osm_data/leeds/Leeds-shp/shape/" ... Done.
```

Check the data *leeds_shp*:

```
>>> retr_data_type = type(leeds_shp)
>>> print(f'Data type of `leeds_shp`:\n  {retr_data_type}')
Data type of `leeds_shp`:
  <class 'dict'>

>>> retr_data_keys = list(leeds_shp.keys())
>>> print(f'The "keys" of `leeds_shp`:\n  {\'\n  \'.join(retr_data_keys)}')
The "keys" of `leeds_shp`:
  buildings
  landuse
  natural
  places
  points
  railways
  roads
  waterways

>>> leeds_shp_railways = leeds_shp['railways']
>>> retr_layer_type = type(leeds_shp_railways)
>>> print(f'Data type of the \'railways\' layer:\n  {retr_layer_type}')
Data type of the 'railways' layer:
  <class 'geopandas.geodataframe.GeoDataFrame'>
```

We could import the data of a list of selected layers. For example, let's import the data of `'railways'`, `'roads'` and `'waterways'`:

```
>>> layer_names = ['railways', 'roads', 'waterways']

>>> osmdb.import_osm_data(
...     leeds_shp, table_name=subregion_name, schema_names=layer_names, verbose=True)
Proceed to import data into the table "Leeds" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
  "railways" ... Done: <total of rows> features.
  "roads" ... Done: <total of rows> features.
  "waterways" ... Done: <total of rows> features.
```

As illustrated in Figure 9, three schemas: *'railways'*, *'roads'* and *'waterways'* are created in the *'osmdb_test'* database for storing the data of the three shapefile layers of Leeds.

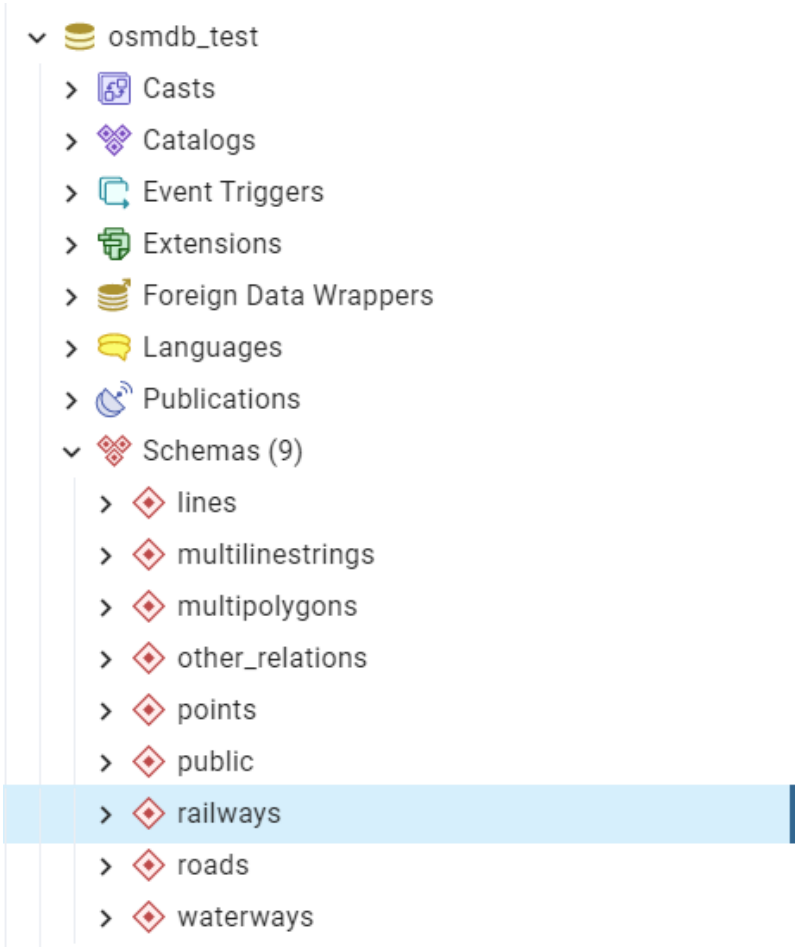

Figure 9: An illustration of the newly created schemas for the selected layers of Leeds shapefile data.

Now let's fetch only the *'railways'* data of Leeds from the *'osmdb_test'* database:

```
>>> layer_name = 'railways'

>>> leeds_shp_ = osmdb.fetch_data(
...     subregion_name, layer_names=layer_name, sort_by='osm_id', verbose=True)
Fetching the data of "Leeds" ...
    "railways" ... Done.
```

Check the data *leeds_shp_*:

```
>>> retr_data_type = type(leeds_shp_)
>>> print(f'Data type of `leeds_shp_`:\n  {retr_data_type}')
Data type of `leeds_shp_`:
  <class 'dict'>

>>> retr_data_keys = list(leeds_shp_.keys())
>>> print(f'The "keys" of `leeds_shp_`:\n  {retr_data_keys}')
The "keys" of `leeds_shp_`:
  ['railways']

>>> # Data frame of the 'railways' layer
>>> leeds_shp_railways_ = leeds_shp_[layer_name]
```

(continues on next page)

(continued from previous page)

```
>>> leeds_shp_railways_.head()
    osm_id  ...                                           geometry
0  3666100  ...  LINESTRING (-1.4935085 53.6772284, -1.4941684 ...
1  3688274  ...  LINESTRING (-1.5321838 53.6588828, -1.5316487 ...
2  3688277  ...  LINESTRING (-1.4361755 53.6908246, -1.4365117 ...
3  3688278  ...  LINESTRING (-1.4265919 53.6975115, -1.4268996 ...
4  3688279  ...  LINESTRING (-1.3564814 53.7237694, -1.3569774 ...
[5 rows x 4 columns]
```

Note

- The original `leeds_shp_railways` is a GeoDataFrame, however the retrieved `leeds_shp_railways_` is a standard Pandas DataFrame.
- It must be noted that empty strings, `''`, may be automatically saved as `None` when importing `leeds_shp` into the PostgreSQL database.
- The data retrieved from a PostgreSQL database may not be in the same order as it is in the database; the retrieved `leeds_shp_railways_` may not be exactly equal to *leeds_shp_railways*. However, they contain the same information. We can sort the data by `'osm_id'` or `'id'` and convert geometry to WKT/WKB (or vice versa) to make a comparison (see the test code below).

Check whether `leeds_shp_railways_` is equivalent to `leeds_shp_railways`:

```
>>> import shapely.wkt
>>> import geopandas as gpd

>>> # Convert `leeds_shp_railways_` to a GeoDataFrame
>>> leeds_shp_railways_geo = leeds_shp_railways_.copy()
>>> leeds_shp_railways_geo['geometry'] = leeds_shp_railways_geo['geometry'].map(
...     shapely.wkt.loads)
>>> leeds_shp_railways_geo = gpd.GeoDataFrame(
...     leeds_shp_railways_geo, crs=osmdb.reader.SHP.EPSG4326_WGS84_PROJ4)

>>> check_eq = leeds_shp_railways_geo.equals(leeds_shp_railways)
>>> print(f"`leeds_shp_railways_geo` is equivalent to `leeds_shp_railways`: {check_eq}")
`leeds_shp_railways_geo` is equivalent to `leeds_shp_railways`: True
```

3.3.4 Drop data

To drop the data of all or selected layers that have been imported for one or multiple geographic regions, we can use the method *drop_subregion_tables()*.

For example, let's now drop the *'railways'* schema for Leeds:

```
>>> # Recall that: subrgn_name == 'Leeds'; lyr_name == 'railways'
>>> osmdb.drop_subregion_tables(subregion_name, schema_names=layer_name, verbose=True)
Proceed to drop table "railways"."Leeds"
  from postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
```

(continues on next page)

(continued from previous page)

```
Dropping the table ...
  "railways"."Leeds" ... Done.
```

Then drop the *'waterways'* schema for Leeds, and both the *'lines'* and *'multilinestrings'* schemas for Rutland:

```
>>> subregion_names = ['Leeds', 'Rutland']
>>> layer_names = ['waterways', 'lines', 'multilinestrings']
>>> osmdb.drop_subregion_tables(subregion_names, schema_names=layer_names, verbose=True)
Proceed to drop tables from postgres:***@localhost:5432/osmdb_test:
    "Leeds"
    "Rutland"
  under the schemas:
    "multilinestrings"
    "waterways"
    "lines"
? [No]|Yes: yes
Dropping the tables ...
  "multilinestrings"."Rutland" ... Done.
  "waterways"."Leeds" ... Done.
  "lines"."Rutland" ... Done.
```

We could also easily drop the whole database *'osmdb_test'* if we don't need it anymore:

```
>>> osmdb.drop_database(verbose=True)
Drop the database "osmdb_test" from postgres:***@localhost:5432?
 [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

3.4 Clear up 'the mess' in here

Now we are approaching the end of this tutorial. The final task we may want to do is to remove all the data files that have been downloaded and generated. Those data are all stored in the directory **"tests/osm_data/"**. Let's take a quick look at what's in here:

```
>>> os.listdir(data_dir)  # Recall that dat_dir == "tests/osm_data"
['birmingham',
 'greater-london',
 'leeds',
 'lon-bir-railways',
 'london',
 'rutland',
 'west-midlands',
 'west-yorkshire']
```

Let's delete the directory **"tests/osm_data/"**:

```
>>> from pyhelpers.dirs import delete_dir

>>> delete_dir(data_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

(continues on next page)

(continued from previous page)

```
>>> os.path.isdir(data_dir)  # Check if the directory still exists
False
```

This is the end of the *quick-start tutorial*.

Any issues regarding the use of the package are all welcome and should be logged/reported onto the Issue Tracker.

For more details and examples, check *subpackages* and *modules*.

Chapter 4

Subpackages

1anti-flashwhitewhite

downloader	Download OpenStreetMap (OSM) data from free download servers: Geofabrik and BBBike.
reader	Read the OSM data extracts in various file formats.
ios	Implement storage I/O of (parsed) OSM data extracts with PostgreSQL.

4.1 downloader

Download OpenStreetMap (OSM) data from free download servers: Geofabrik and BBBike.

4.1.1 Download OSM data

1anti-flashwhitewhite

GeofabrikDownloader([download_dir, update])	Download OSM data from Geofabrik free download server.
BBBikeDownloader([download_dir, update])	Download OSM data from BBBike free download server.

GeofabrikDownloader

class pydriosm.downloader.**GeofabrikDownloader**(*download_dir=None*, *update=False*, ***kwargs*)

Download OSM data from Geofabrik free download server.

Parameters

download_dir (*str | os.PathLike | pathlib.Path | None*) – name or pathname of a directory for saving downloaded data files, defaults to None; when download_dir=None, downloaded data files are saved to a folder named 'osm_data' under the current working directory

Variables

- **valid_subregion_names** (*set*) – names of (sub)regions available on the free download server
- **valid_file_formats** – filename extensions of the data files available
- **download_index** – index of downloads for all available (sub)regions
- **continent_tables** (*dict*) – download catalogues for each continent
- **region_subregion_tier** (*dict*) – region-subregion tier
- **having_no_subregions** (*list*) – all (sub)regions that have no subregions
- **catalogue** (*pandas.DataFrame*) – a catalogue (index) of all available downloads (similar to `download_index`)
- **download_dir** (*str | None*) – name or pathname of a directory for saving downloaded data files
- **data_pathnames** (*list*) – list of pathnames of all downloaded data files

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> import os
>>> gfd = GeofabrikDownloader()
>>> gfd.NAME
'Geofabrik'
>>> gfd.URL
'https://download.geofabrik.de/'
>>> gfd.DOWNLOAD_INDEX_URL
'https://download.geofabrik.de/index-v1.json'
>>> os.path.relpath(gfd.download_dir)
'osm_data\geofabrik'
>>> gfd = GeofabrikDownloader(download_dir="tests/osm_data")
>>> os.path.relpath(gfd.download_dir)
'tests\osm_data'
```

Attributes

1anti-flashwhitewhite

DEFAULT_DOWNLOAD_DIR	Default download directory.
DOWNLOAD_INDEX_URL	URL of the official download index.
FILE_FORMATS	Valid file formats.
LONG_NAME	Full name of the data resource.
NAME	Name of the free download server.
URL	URL of the homepage to the free download server.

GeofabrikDownloader.DEFAULT_DOWNLOAD_DIR

GeofabrikDownloader.**DEFAULT_DOWNLOAD_DIR: str = 'osm_data/geofabrik'**

Default download directory.

GeofabrikDownloader.DOWNLOAD_INDEX_URL

GeofabrikDownloader.**DOWNLOAD_INDEX_URL: str = 'https://download.geofabrik.de/index-v1.json'**

URL of the official download index.

GeofabrikDownloader.FILE_FORMATS

GeofabrikDownloader.**FILE_FORMATS: set = {'.gpkg.zip', '.osm.bz2', '.osm.pbf', '.shp.zip'}**

Valid file formats.

GeofabrikDownloader.LONG_NAME

GeofabrikDownloader.**LONG_NAME: str = 'Geofabrik OpenStreetMap data extracts'**

Full name of the data resource.

GeofabrikDownloader.NAME

GeofabrikDownloader.**NAME: str = 'Geofabrik'**

Name of the free download server.

GeofabrikDownloader.URL

GeofabrikDownloader.**URL: str = 'https://download.geofabrik.de/'**

URL of the homepage to the free download server.

Methods

1anti-flashwhitewhite

cdd(*sub_dir[, mkdir])	Change directory to default download directory and its subdirectories or a specific file.
download_data(subregion_names, osm_file_formats)	Download OSM data (in a specific format) of one (or multiple) geographic (sub)region(s).
file_exists(subregion_name, osm_file_format)	Check whether a data file of a geographic (sub)region already exists locally, given its default filename.

continues on next page

Table 4 – continued from previous page

file_exists_and_more(subregion_names, ...[, ...])	Check if a requested data file already exists and compile information for downloading the data.
format_confirmation_prompt([data_name, ...])	Compose a short message to be printed for confirmation.
get_catalogue([update, ...])	Get a catalogue (index) of all available downloads.
get_continent_tables([update, ...])	Get download catalogues for each continent.
get_default_filename(subregion_name, ...[, ...])	get a default filename for a geograpic (sub)region.
get_default_pathname(subregion_name, ...[, ...])	Get the default pathname of a local directory for storing a downloaded data file.
get_default_sub_path(subregion_name_, ...)	Get default sub path for saving OSM data file of a geographic (sub)region.
get_download_index([update, ...])	Get the official index of downloads for all available geographic (sub)regions.
get_prepacked_data(meth[, data_name, ...])	Get auxiliary data (that is to be prepacked in the package).
get_raw_directory_index(url[, save_path, ...])	Get a raw directory index (including download information of older file logs).
get_region_subregion_tiers([update, ...])	Get region-subregion tier and all (sub)regions that have no subregions.
get_subregion_download_url(subregion_na ...)	Get a download URL of a geographic (sub)region.
get_subregion_table(url[, verbose, raise_error])	Get download information of all geographic (sub)regions on a web page.
get_subregions(*subregion_name[, deep])	Retrieve names of all subregions (if any) of the given geographic (sub)region(s).
get_valid_download_info(subregion_name, ...)	Get information of downloading (or downloaded) data file.
get_valid_subregion_names([update, ...])	Get names of all available geographic (sub)regions.
make_subregion_dirname(subregion_name_)	Make the name of the directory one level up from an OSM data file of a geographic (sub)region.
print_action_prompt([data_name, verbose, ...])	Print a short message showing the action as a function runs.
print_status([data_name, path_to_file, ...])	Print a short message for an otherwise situation.
specify_sub_download_dir(subregion_name ...)	Specify a directory for downloading data of all subregions of a geographic (sub)region.
validate_file_format(osm_file_format[, ...])	Validate an input file format of OSM data.
validate_subregion_name(subregion_name[...])	Validate an input name of a geographic (sub)region.

continues on next page

Table 4 – continued from previous page

verify_download_dir([download_dir, ...])	Verify the pathname of the current download directory.

GeofabrikDownloader.cdd

classmethod GeofabrikDownloader.**cdd**(**sub_dir*, *mkdir=False*, ***kwargs*)

Change directory to default download directory and its subdirectories or a specific file.

Parameters

- **sub_dir** (*str | os.PathLike[str]*) – name of directory; names of directories (and/or a filename)
- **mkdir** (*bool*) – whether to create a directory, defaults to False
- **kwargs** – [optional] parameters of pyhelpers.dirs.cd()

Returns

an absolute pathname to a directory (or a file)

Return type

str | os.PathLike[str]

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> import os
>>> os.path.relpath(BaseDownloader.cdd())
'osm_data'
```

GeofabrikDownloader.download_data

GeofabrikDownloader.**download_data**(*subregion_names*, *osm_file_formats*, *download_dir=None*, *update=False*, *confirmation_required=True*, *deep=False*, *interval=None*, *verify_download_dir=True*, *verbose=False*, *ret_download_path=False*, ***kwargs*)

Download OSM data (in a specific format) of one (or multiple) geographic (sub)region(s).

Parameters

- **subregion_names** (*str | list*) – name of a geographic (sub)region (or names of multiple geographic (sub)regions) available on Geofabrik.
- **osm_file_formats** (*str | list*) – file format/extension of the OSM data available on the download server
- **download_dir** (*str | None*) – directory for saving the downloaded file(s), defaults to None; when download_dir=None, it refers to *cdd_geofabrik()*.
- **update** (*bool*) – whether to update the data if it already exists, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True

- **deep** (*bool*) – whether to further check availability of sub-subregions data, defaults to `False`
- **interval** (*int | float | None*) – interval (in sec) between downloading two subregions, defaults to `None`
- **verify_download_dir** (*bool*) – whether to verify the pathname of the current download directory, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **ret_download_path** (*bool*) – whether to return the path(s) to the downloaded file(s), defaults to `False`
- **kwargs** – optional parameters of pyhelpers.ops.download_file_from_url()

Returns

absolute path(s) to downloaded file(s) when `ret_download_path` is `True`

Return type

list | str

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pyhelpers.dirs import delete_dir
>>> import os
```

Example 1:

```
>>> gfd = GeofabrikDownloader()
>>> # Download PBF data file of 'Greater London' and 'Rutland'
>>> subregion_names = ['Isle of Wight', 'rutland']  # Case-insensitive
>>> osm_file_format = ".pbf"
>>> gfd.download_data(subregion_names, osm_file_format, verbose=True)
Proceed to download data in the format '.osm.pbf' for the following geographic (sub...
    "Isle of Wight"
    "Rutland"
  to "./osm_data/geofabrik/europe/united-kingdom/england/"
? [No]|Yes: yes
Downloading "isle-of-wight-latest.osm.pbf" 100%|■■■■■■■■■■■■| 8.83M/8.83M | 16....
  Saving "isle-of-wight-latest.osm.pbf" to "./osm_data/geofabrik/europe/united-king...
Downloading "rutland-latest.osm.pbf" 100%|■■■■■■■■■■■■| 1.89M/1.89M | 4.58MB/s ...
  Saving "rutland-latest.osm.pbf" to "./osm_data/geofabrik/europe/united-kingdom/en...
>>> len(gfd.data_paths)
2
>>> for file_path in gfd.data_paths: print(os.path.basename(file_path))
isle-of-wight-latest.osm.pbf
rutland-latest.osm.pbf
>>> # Since `download_dir` was not specified when instantiating the class,
>>> #   the data is now in the default download directory
>>> os.path.relpath(gfd.download_dir)  # (on Windows)
'osm_data\geofabrik'
>>> download_dir_ = os.path.dirname(gfd.download_dir)

>>> # Download shapefiles of West Midlands (to a given directory "tests/osm_data")
```

(continues on next page)

(continued from previous page)

```
>>> subregion_name = 'west midlands'  # Case-insensitive
>>> osm_file_format = [".shp", ".gpkg", "pbf"]
>>> download_dir = "tests/osm_data"
>>> gfd.download_data(subregion_name, osm_file_format, download_dir, verbose=True)
Proceed to download data in the formats ('.shp.zip', '.gpkg.zip', '.osm.pbf') for t...
    "West Midlands"
? [No]|Yes: yes
No '.shp.zip' data is available for "West Midlands".
Try to download the data of its subregions instead
? [No]|Yes: no
Downloading "west-midlands-latest-free.gpkg.zip" 100%|■■■■■■■■■■■| 103M/103M |...
  Saving "west-midlands-latest-free.gpkg.zip" to "./tests/osm_data/west-midlands/" ...
Downloading "west-midlands-latest.osm.pbf" 100%|■■■■■■■■■■■| 58.4M/58.4M | 10....
  Saving "west-midlands-latest.osm.pbf" to "./tests/osm_data/west-midlands/" ... Done.
>>> len(gfd.data_paths)
4
>>> os.path.relpath(gfd.data_paths[-1])  # (on Windows)
'tests\osm_data\west-midlands\west-midlands-latest.osm.pbf'
>>> # Now the `.download_dir` variable has changed to the given one `download_dir`
>>> os.path.relpath(gfd.download_dir)  # (on Windows)
'tests\osm_data'
>>> # while `.cdd()` remains the default one
>>> os.path.relpath(gfd.cdd())  # (on Windows)
'osm_data\geofabrik'
>>> # Delete the above downloaded directories
>>> delete_dir([download_dir_, gfd.download_dir], verbose=True)
To delete the following directories:
  "./osm_data/" (Not empty)
  "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting:
  "./osm_data/" ... Done.
  "./tests/osm_data/" ... Done.
```

Example 2:

```
>>> # Create a new instance with a pre-specified download directory
>>> gfd = GeofabrikDownloader(download_dir="tests/osm_data")
>>> os.path.relpath(gfd.download_dir)  # (on Windows)
'tests\osm_data'
>>> # Download shapefiles of UK (to the directory specified by instantiation)
>>> # (Note that .shp.zip data is not available for "United Kingdom".)
>>> subregion_name = 'United Kingdom'  # Case-insensitive
>>> osm_file_format = ".shp"
>>> # By default, `deep_retry=False`
>>> gfd.download_data(subregion_name, osm_file_format, verbose=True)
Proceed to download data in the format '.shp.zip' for the following geographic (sub...
    "United Kingdom"
? [No]|Yes: yes
No '.shp.zip' data is available for "United Kingdom".
Try to download the data of its subregions instead
? [No]|Yes: yes
Downloading "bermuda-latest-free.shp.zip" 100%|■■■■■■■■■■■| 4.04M/4.04M | 9.08...
  Saving "bermuda-latest-free.shp.zip" to "./tests/osm_data/europe/united-kingdom/u...
Downloading "england-latest-free.shp.zip" 100%|■■■■■■■■■■■| 2.78G/2.78G | 36.2...
  Saving "england-latest-free.shp.zip" to "./tests/osm_data/europe/united-kingdom/u...
```

(continues on next page)

(continued from previous page)

```
Downloading "falklands-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 11.0M/11.0M | 13...
  Saving "falklands-latest-free.shp.zip" to "./tests/osm_data/europe/united-kingdom...
Downloading "scotland-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 561M/561M | 33.2M...
  Saving "scotland-latest-free.shp.zip" to "./tests/osm_data/europe/united-kingdom/...
Downloading "wales-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 252M/252M | 33.6MB/s...
  Saving "wales-latest-free.shp.zip" to "./tests/osm_data/europe/united-kingdom/uni...
>>> len(gfd.data_paths)
5
>>> # Now set `deep_retry=True`
>>> gfd.download_data(subregion_name, osm_file_format, verbose=1, deep=True)
Proceed to download data in the format '.shp.zip' for the following geographic (sub...
    "Lancashire"
    "Scotland"
    "Merseyside"
    ...
    "West Sussex"
  to "./tests/osm_data/europe/united-kingdom/"
? [No]|Yes: yes
Downloading "wales-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 252M/252M | 28.7MB/s...
  Saving "wales-latest-free.shp.zip" to "./tests/osm_data/europe/united-kingdom/wal...
Downloading "scotland-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 561M/561M | 31.3M...
  Saving "scotland-latest-free.shp.zip" to "./tests/osm_data/europe/united-kingdom/...
Downloading "falklands-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 11.0M/11.0M | 5....
  Saving "falklands-latest-free.shp.zip" to "./tests/osm_data/europe/united-kingdom...
...
    ...
Downloading "rutland-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 2.71M/2.71M | 7.33...
  Saving "rutland-latest-free.shp.zip" to "./tests/osm_data/europe/united-kingdom/e...
...
    ...
Downloading "west-yorkshire-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 87.3M/87.3M...
  Saving "west-yorkshire-latest-free.shp.zip" to "./tests/osm_data/europe/united-ki...
Downloading "wiltshire-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 62.7M/62.7M | 28...
  Saving "wiltshire-latest-free.shp.zip" to "./tests/osm_data/europe/united-kingdom...
Downloading "worcestershire-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 35.0M/35.0M...
  Saving "worcestershire-latest-free.shp.zip" to "./tests/osm_data/europe/united-ki...
>>> # Check the file paths
>>> len(gfd.data_paths)
56
>>> # Check the current default `download_dir`
>>> os.path.relpath(gfd.download_dir)  # (on Windows)
'tests\osm_data'
>>> # Delete all the downloaded files
>>> delete_dir(gfd.download_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

GeofabrikDownloader.file_exists

GeofabrikDownloader.**file_exists**(*subregion_name*, *osm_file_format*, *data_dir=None*, *update=False*, *verbose=False*, *ret_file_path=False*)

Check whether a data file of a geographic (sub)region already exists locally, given its default filename.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on Geofabrik free download server
- **osm_file_format** (*str*) – file format/extension of the OSM data available on the download server
- **data_dir** (*str | None*) – directory where the data file (or files) is (or are) stored, defaults to `None`; when `data_dir=None`, it refers to *cdd_geofabrik()*.
- **update** (*bool*) – whether to (check and) update the data, defaults to `False`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **ret_file_path** (*bool*) – whether to return the path to the data file (if it exists), defaults to `False`

Returns

whether the requested data file exists; or the path to the data file

Return type

bool | str

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pyhelpers.dirs import delete_dir
>>> import os
>>> gfd = GeofabrikDownloader(download_dir="tests/osm_data")
>>> # Download the PBF data of London (to the default directory)
>>> subregion_name = 'london'
>>> osm_file_format = ".geopackage"
>>> gfd.download_data(subregion_name, osm_file_format, verbose=True)
Proceed to download data in the format '.gpkg.zip' for the following geographic (su...
    "Greater London"
  to "./tests/osm_data/europe/united-kingdom/england/greater-london/"
? [No]|Yes: yes
Downloading "greater-london-latest-free.gpkg.zip" 100%|■■■■■■■■■■| 206M/206M ...
  Saving "greater-london-latest-free.gpkg.zip" to "./tests/osm_data/europe/united-k...
>>> # Check whether the PBF data file exists; `ret_file_path` is by default `False`
>>> gpkg_exists = gfd.file_exists(subregion_name, osm_file_format)
>>> gpkg_exists  # If the data file exists at the default directory
True
>>> # Set `ret_file_path=True`
>>> path_to_gpkg = gfd.file_exists(
...     subregion_name, osm_file_format, ret_file_path=True)
>>> os.path.relpath(path_to_gpkg)  # If the data file exists at the default directory
'tests\osm_data\europe\united-kingdom\england\greater-london\greater-london-l...
>>> # Remove the download directory:
>>> delete_dir(gfd.download_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
>>> # Check if the data file still exists at the specified download directory
>>> gfd.file_exists(subregion_name, osm_file_format)
False
```

GeofabrikDownloader.file_exists_and_more

GeofabrikDownloader.**file_exists_and_more**(*subregion_names*, *osm_file_formats*, *data_dir=None*, *update=False*, *confirmation_required=True*, *verbose=True*, *deep=False*)

Check if a requested data file already exists and compile information for downloading the data.

Parameters

- **subregion_names** (*str | list*) – name(s) of geographic (sub)region(s) available on a free download server
- **osm_file_formats** (*str*) – file format of the OSM data available on the free download server
- **data_dir** (*str | None*) – directory where the data file (or files) is (or are) stored, defaults to None
- **update** (*bool*) – whether to (check on and) update the data, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to True
- **deep** (*bool*) – whether to further check availability of sub-subregions data, defaults to False

Returns

whether the requested data file exists; or the path to the data file

Return type

tuple

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader, BBBikeDownloader
>>> gfd = GeofabrikDownloader()
>>> gfd.file_exists_and_more('London', ".pbf")
(['Greater London'],
 ['.osm.pbf'],
 True,
 'Proceed to download data in the format '.osm.pbf' for the following geographic ...
 [])
>>> gfd.file_exists_and_more(['london', 'rutland'], ".pbf")
(['Greater London', 'Rutland'],
 ['.osm.pbf'],
 True,
 'Proceed to download data in the format '.osm.pbf' for the following geographic ...
 [])
>>> gfd.file_exists_and_more(['london', 'rutland'], ["shp", ".pbf"])
(['Greater London', 'Rutland'],
```

(continues on next page)

(continued from previous page)

```
 ['.shp.zip', '.osm.pbf'],
 True,
 'Proceed to download data in the formats ('.shp.zip', '.osm.pbf') for the foll...
 [])
>>> bbd = BBBikeDownloader()
>>> bbd.file_exists_and_more('London', ".pbf")
(['London'],
 ['.pbf'],
 True,
 'Proceed to download data in the format '.pbf' for the following geographic (sub...
 [])
>>> bbd.file_exists_and_more(['birmingham', 'leeds'], ".pbf")
(['Birmingham', 'Leeds'],
 ['.pbf'],
 True,
 'Proceed to download data in the format '.pbf' for the following geographic (sub...
 [])
```

GeofabrikDownloader.format_confirmation_prompt

classmethod GeofabrikDownloader.**format_confirmation_prompt**(*data_name='<data_name>'*, *file_path='<file_path>'*, *update=False*, *note=''*)

Compose a short message to be printed for confirmation.

Parameters

- **data_name** (*str*) – name of the prepacked data. Defaults to `'<data_name>'`.
- **file_path** (*str | os.PathLike*) – pathname of the prepacked data file. Defaults to `"<file_path>"`.
- **update** (*bool*) – whether to (check on and) update the prepacked data. Defaults to `False`.
- **note** (*str*) – additional message. Defaults to `""`.

Returns

a short message to be printed for confirmation.

Return type

str

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> BaseDownloader.format_confirmation_prompt()
'Proceed to retrieve/compile data of <data_name>\n?'
>>> BaseDownloader.format_confirmation_prompt(update=True)
'Proceed to update the data of <data_name>\n?'
```

GeofabrikDownloader.get_catalogue

GeofabrikDownloader.**get_catalogue**(*update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Get a catalogue (index) of all available downloads.

Similar to *get_download_index()*.

Parameters

- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to False
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Returns

a catalogue for all subregion downloads

Return type

pandas.DataFrame | None

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()
>>> # A download catalogue for all subregions
>>> dwnld_catalog = gfd.get_catalogue()
>>> type(dwnld_catalog)
pandas.DataFrame
>>> dwnld_catalog.shape
(543, 7)
>>> dwnld_catalog.head()
               subregion  ... .osm.bz2
0                 Africa  ...     None
1             Antarctica  ...     None
2                   Asia  ...     None
3  Australia and Oceania  ...     None
4        Central America  ...     None
[5 rows x 7 columns]
>>> dwnld_catalog.columns.to_list()
['subregion',
 'subregion-url',
 '.osm.pbf',
 '.osm.pbf-size',
 '.gpkg.zip',
 '.shp.zip',
 '.osm.bz2']
```

Note

- Information of London/Enfield is not directly available from the web page of Greater London.
- Two subregions have the same name - 'Georgia': Europe/Georgia and US/Georgia; In the latter case, a suffix ' (US)' is appended.

GeofabrikDownloader.get_continent_tables

`GeofabrikDownloader.`**`get_continent_tables`**(*update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*, ***kwargs*)

Get download catalogues for each continent.

Parameters

- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

download catalogues for each continent

Return type

dict | None

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()

>>> # Download information of subregions for each continent
>>> continent_tables = gfd.get_continent_tables()
>>> type(continent_tables)
dict
>>> list(continent_tables.keys())
['Africa',
 'Antarctica',
 'Asia',
 'Australia and Oceania',
 'Central America',
 'Europe',
 'North America',
 'South America']

>>> # Information about the data of subregions in Asia
```

(continues on next page)

(continued from previous page)

```
>>> asia_table = continent_tables['Asia']
>>> asia_table.shape
(40, 7)
>>> asia_table.head()
     subregion  ... .osm.bz2
0  Afghanistan  ...     None
1      Armenia  ...     None
2   Azerbaijan  ...     None
3   Bangladesh  ...     None
4       Bhutan  ...     None
[5 rows x 7 columns]
>>> asia_table.columns.to_list()
['subregion',
 'subregion-url',
 '.osm.pbf',
 '.osm.pbf-size',
 '.gpkg.zip',
 '.shp.zip',
 '.osm.bz2']
```

GeofabrikDownloader.get_default_filename

GeofabrikDownloader.**get_default_filename**(*subregion_name*, *osm_file_format*, *update=False*)

get a default filename for a geograpic (sub)region.

The default filename is derived from the download URL of the requested data file.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on Geofabrik free download server
- **osm_file_format** (*str*) – file format/extension of the OSM data available on the download server
- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to False

Returns

default OSM filename for the subregion_name

Return type

str | None

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()
>>> # Default filename of the PBF data of London
>>> gfd.get_default_filename(subregion_name='london', osm_file_format=".pbf")
'greater-london-latest.osm.pbf'
>>> # Default filename of the shapefile data of Great Britain
>>> gfd.get_default_filename(subregion_name='britain', osm_file_format=".gpkg")
No ".gpkg.zip" data is available to download for "Great Britain".
```

GeofabrikDownloader.get_default_pathname

GeofabrikDownloader.**get_default_pathname**(*subregion_name*, *osm_file_format*, *mkdir=False*, *update=False*, *verbose=False*)

Get the default pathname of a local directory for storing a downloaded data file.

The default file path is derived from the download URL of the requested data file.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on Geofabrik free download server
- **osm_file_format** (*str*) – file format/extension of the OSM data available on the download server
- **mkdir** (*bool*) – whether to create a directory, defaults to `False`
- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`

Returns

default filename of the subregion and default (absolute) path to the file

Return type

Tuple[str, str]

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> import os
>>> gfd = GeofabrikDownloader()

>>> # Default filename and download path of the PBF data of London
>>> subregion_name, osm_file_format = 'london', ".geopackage"
>>> pathname, filename = gfd.get_default_pathname(subregion_name, osm_file_format)
>>> os.path.relpath(os.path.dirname(pathname))
'osm_data\geofabrik\europe\united-kingdom\england\greater-london'
>>> filename
'greater-london-latest-free.gpkg.zip'
```

GeofabrikDownloader.get_default_sub_path

classmethod GeofabrikDownloader.**get_default_sub_path**(*subregion_name_*, *download_url*)

Get default sub path for saving OSM data file of a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – validated name of a (sub)region available on a free download server
- **download_url** (*str*) – download URL of a geographic (sub)region

Returns
default sub path

Return type
str | os.PathLike[str]

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader

>>> subrgn_name_ = 'London'
>>> dwnld_url = 'https://download.bbbike.org/osm/bbbike/London/London.osm.pbf'
>>> BaseDownloader.get_default_sub_path(subrgn_name_, dwnld_url)
'\london'
```

GeofabrikDownloader.get_download_index

GeofabrikDownloader.**get_download_index**(*update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*, ***kwargs*)

Get the official index of downloads for all available geographic (sub)regions.

Similar to *get_catalogue()*.

Parameters

- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
the official index of all downloads

Return type
pandas.DataFrame | None

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()
>>> download_index = gfd.get_download_index()
>>> type(download_index)
pandas.DataFrame
>>> download_index.shape
(544, 12)
>>> download_index.head()
           id  ...                                            updates
0         act  ...  https://download.geofabrik.de/australia-oceani...
1  afghanistan  ...  https://download.geofabrik.de/asia/afghanistan...
```

(continues on next page)

(continued from previous page)

```
2       africa  ...       https://download.geofabrik.de/africa-updates
3      albania  ...  https://download.geofabrik.de/europe/albania-u...
4      alberta  ...  https://download.geofabrik.de/north-america/ca...
[5 rows x 12 columns]
>>> download_index.columns.to_list()
['id',
 'parent',
 'name',
 'iso3166-1:alpha2',
 'iso3166-2',
 'geometry',
 '.osm.pbf',
 '.shp.zip',
 'pbf-internal',
 'history',
 'taginfo',
 'updates']
```

GeofabrikDownloader.get_prepacked_data

classmethod GeofabrikDownloader.**get_prepacked_data**(*meth*, *data_name='<data_name>'*, *file_stem=None*, *ext='.pkl.xz'*, *update=False*, *confirmation_required=True*, *dump_backup=True*, *verbose=False*, *confirmation_prompt_note=''*, *action_prompt_note=''*, *action_prompt_end=' ... '*, *ending_message='Done.'*, *raise_error=False*, ***kwargs*)

Get auxiliary data (that is to be prepacked in the package).

Parameters

- **meth** (*Callable*) – name of a class method for getting (auxiliary) prepacked data
- **data_name** (*str*) – name of the prepacked data, defaults to `'<data_name>'`
- **file_stem** – The filename (without file extension) that overrides `data_name` for determining file path. Defaults to `None`.
- **ext** (*str*) – File extension of the filename of prepacked data; defaults to `".pkl"`.
- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **dump_backup** (*bool*) – If `True`, save a cached data file.
- **ending_message** (*str*) – The ending message to print.

- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **confirmation_prompt_note** (*str*) – additional message for the method `compose_cfm_msg()`, defaults to `""`
- **action_prompt_note** (*str*) – equivalent of the parameter `note` of the method `print_action_msg()`, defaults to `""`
- **action_prompt_end** (*str*) – equivalent of the parameter `end` of the method `print_action_msg()`, defaults to `" ... "`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
auxiliary data

Return type
Any

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> data = BaseDownloader.get_prepacked_data(print, verbose=True, raise_error=True)
Proceed to retrieve/compile data of <data_name>
? [No]|Yes: yes
Retrieving/compiling the data ...
Done.
>>> data is None
True
```

GeofabrikDownloader.get_raw_directory_index

classmethod GeofabrikDownloader.**get_raw_directory_index**(*url*, *save_path=None*, *verbose=False*, *raise_error=False*)

Get a raw directory index (including download information of older file logs).

Parameters

- **url** (*str*) – URL of a web page of a data resource (e.g. a subregion)
- **save_path** (*str | pathlib.Path | None*)
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
information of raw directory index

Return type
pandas.DataFrame | None

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()
>>> raw_directory_index = gfd.get_raw_directory_index(gfd.URL, verbose=True)
Collecting the raw directory index on 'https://download.geofabrik.de/' ... Failed. ...
>>> raw_directory_index is None
True
>>> url = 'https://download.geofabrik.de/europe/great-britain.html'
>>> raw_directory_index = gfd.get_raw_directory_index(url)
>>> type(raw_directory_index)
pandas.DataFrame
>>> raw_directory_index.columns.tolist()
['file', 'date', 'size', 'metric_file_size', 'url']
```

GeofabrikDownloader.get_region_subregion_tiers

GeofabrikDownloader.**get_region_subregion_tiers**(*update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Get region-subregion tier and all (sub)regions that have no subregions.

This includes all geographic (sub)regions for which data of subregions is unavailable.

Parameters

- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

region-subregion tier and all (sub)regions that have no subregions

Return type

tuple[dict, list] | tuple[None, None]

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()
>>> # region-subregion tiers, and all regions that have no subregions
>>> region_subregion_tiers, having_no_subregions = gfd.get_region_subregion_tiers()
>>> type(region_subregion_tiers)
dict
>>> # Keys of the region-subregion tier
>>> list(region_subregion_tiers)
['Africa',
 'Antarctica',
```

(continues on next page)

(continued from previous page)

```
 'Asia',
 'Australia and Oceania',
 'Central America',
 'Europe',
 'North America',
 'South America']
>>> type(having_no_subregions)
list
>>> len(having_no_subregions)
513
>>> # Example: five regions that have no subregions
>>> having_no_subregions[0:5]
['Antarctica', 'Algeria', 'Angola', 'Benin', 'Botswana']
```

GeofabrikDownloader.get_subregion_download_url

GeofabrikDownloader.**get_subregion_download_url**(*subregion_name*, *osm_file_format*, *update=False*, *verbose=False*, *raise_error=True*)

Get a download URL of a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on Geofabrik free download server
- **osm_file_format** (*str*) – file format/extension of the OSM data available on the download server
- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – (if the input fails to match a valid name) whether to raise an *pydriosm.errors.InvalidSubregionNameError* or *InvalidFileFormatError*; defaults to `True`

Returns

name and URL of the subregion

Return type

tuple

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()
>>> subregion_name = 'England'
>>> osm_file_format = ".pbf"
>>> subregion_name_, download_url = gfd.get_subregion_download_url(
...     subregion_name, osm_file_format)
>>> subregion_name_  # The name of the subregion on the free downloader server
```

(continues on next page)

(continued from previous page)

```
'England'
>>> download_url  # The URL of the PBF data file
'https://download.geofabrik.de/europe/united-kingdom/england-latest.osm.pbf'
>>> subregion_name = 'britain'
>>> osm_file_format = ".geopackage"  # Or, osm_file_format = ".gpkg"
>>> subregion_name_, download_url = gfd.get_subregion_download_url(
...     subregion_name, osm_file_format)
>>> subregion_name_
'Great Britain'
>>> download_url is None  # The URL of the shapefile for Great Britain is not available
True
```

GeofabrikDownloader.get_subregion_table

classmethod GeofabrikDownloader.**get_subregion_table**(*url*, *verbose=False*, *raise_error=False*)

Get download information of all geographic (sub)regions on a web page.

Parameters

- **url** (*str*) – URL of a subregion's web page
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

download information of all available subregions on the given `url`

Return type

pandas.DataFrame | None

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()

>>> # Download information on the homepage
>>> subregion_table = gfd.get_subregion_table(url=gfd.URL)
>>> subregion_table
               subregion  ... .osm.bz2
0                 Africa  ...     None
1             Antarctica  ...     None
2                   Asia  ...     None
3  Australia and Oceania  ...     None
4         Central America  ...     None
5                 Europe  ...     None
6          North America  ...     None
7          South America  ...     None
[8 rows x 7 columns]
>>> subregion_table.columns.to_list()
['subregion',
 'subregion-url',
```

(continues on next page)

(continued from previous page)

```
 '.osm.pbf',
 '.osm.pbf-size',
 '.gpkg.zip',
 '.shp.zip',
 '.osm.bz2']

>>> # Download information about 'Great Britain'
>>> url = 'https://download.geofabrik.de/europe/united-kingdom.html'
>>> subregion_table = gfd.get_subregion_table(url)
>>> subregion_table
          subregion  ... .osm.bz2
0           Bermuda  ...     None
1           England  ...     None
2  Falkland Islands  ...     None
3          Scotland  ...     None
4             Wales  ...     None
[5 rows x 7 columns]

>>> # Download information about 'Antarctica'
>>> url = 'https://download.geofabrik.de/antarctica.html'
>>> subregion_table = gfd.get_subregion_table(url, verbose=True)
Compiling a subregion list of "Antarctica" ... Failed.
  No data is available for 'Antarctica'.
>>> subregion_table is None
True
>>> # To get more information about the above failure, set `verbose=2`
>>> subregion_table = gfd.get_subregion_table(url, verbose=2)
Compiling a subregion list of "Antarctica" ... Failed.
  No data is available for 'Antarctica'.
  Errors: 'NoneType' object has no attribute 'empty'.
>>> subregion_table is None
True
```

GeofabrikDownloader.get_subregions

`GeofabrikDownloader.`**`get_subregions`**`(`*subregion_name*, *deep=False*`)`

Retrieve names of all subregions (if any) of the given geographic (sub)region(s).

The returned result is based on the region-subregion tier structured by *get_region_subregion_tiers()*.

See also [RNS-1].

Parameters

- **subregion_name** (*str | None*) – name of a (sub)region, or names of (sub)regions, available on Geofabrik free download server
- **deep** (*bool*) – whether to get subregion names of the subregions, defaults to `False`

Returns

name(s) of subregion(s) of the given geographic (sub)region or (sub)regions; when `subregion_name=None`, it returns all (sub)regions that have subregions

Return type
list

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()

>>> # Names of all subregions
>>> all_subrgn_names = gfd.get_subregions()
>>> type(all_subrgn_names)
list

>>> # Names of all subregions of England and North America
>>> e_na_subrgn_names = gfd.get_subregions('england', 'n america')
>>> type(e_na_subrgn_names)
list

>>> # Names of all subregions of North America
>>> na_subrgn_names = gfd.get_subregions('n america', deep=True)
>>> type(na_subrgn_names)
list

>>> # Names of subregions of Great Britain
>>> gb_subrgn_names = gfd.get_subregions('britain')
>>> len(gb_subrgn_names) == 1
True

>>> # Names of all subregions of Great Britain's subregions
>>> gb_subrgn_names_ = gfd.get_subregions('unitied kingdom', deep=True)
>>> len(gb_subrgn_names_) >= len(gb_subrgn_names)
True
```

GeofabrikDownloader.get_valid_download_info

GeofabrikDownloader.**get_valid_download_info**(*subregion_name*, *osm_file_format*, *download_dir=None*, ***kwargs*)

Get information of downloading (or downloaded) data file.

The information includes a valid subregion name, a default filename, a URL and an absolute path where the data file is (to be) saved locally.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on GeofabrikDownloader free download server
- **osm_file_format** (*str*) – file format/extension of the OSM data available on the download server
- **download_dir** (*str | None*) – directory for saving the downloaded file(s), defaults to None; when download_dir=None, it refers to *cdd_geofabrik()*.
- **kwargs** – [optional] parameters of pyhelpers.dirs.cd(), including mkdir``(default: ``False)

Returns

valid subregion name, filename, download url and absolute file path

Return type

Tuple[str, str, str, str]

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> import os
>>> gfd = GeofabrikDownloader()

>>> # valid subregion name, filename, download url and absolute file path
>>> subregion_name = 'london'
>>> osm_file_format = "pbf"
>>> info_1 = gfd.get_valid_download_info(subregion_name, osm_file_format)
>>> subregion_name_, osm_filename, download_url, file_pathname = info_1
>>> subregion_name_
'Greater London'
>>> osm_filename
'greater-london-latest.osm.pbf'
>>> os.path.dirname(download_url)
'https://download.geofabrik.de/europe/united-kingdom/england'
>>> os.path.relpath(os.path.dirname(file_pathname))
'osm_data\geofabrik\europe\united-kingdom\england\greater-london'

>>> # Specify a new directory for downloaded data
>>> download_dir = "tests/osm_data"
>>> info_2 = gfd.get_valid_download_info(subregion_name, osm_file_format, download_dir)
>>> _, _, _, file_pathname2 = info_2
>>> os.path.relpath(os.path.dirname(file_pathname2))
'tests\osm_data\greater-london'

>>> gfd_ = GeofabrikDownloader(download_dir=download_dir)
>>> info_3 = gfd_.get_valid_download_info(subregion_name, osm_file_format)
>>> _, _, _, file_pathname3 = info_3
>>> os.path.relpath(os.path.dirname(file_pathname3))
'tests\osm_data\europe\united-kingdom\england\greater-london'
```

GeofabrikDownloader.get_valid_subregion_names

GeofabrikDownloader.**get_valid_subregion_names**(*update=False*, *confirmation_required=True*, *verbose=False*)

Get names of all available geographic (sub)regions.

Parameters

- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`

Returns
names of all geographic (sub)regions available on Geofabrik free download server

Return type
set | None

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()
>>> # A list of the names of available geographic (sub)regions
>>> valid_subrgn_names = gfd.get_valid_subregion_names()
>>> type(valid_subrgn_names)
set
```

GeofabrikDownloader.make_subregion_dirname

classmethod GeofabrikDownloader.**make_subregion_dirname**(*subregion_name_*)

Make the name of the directory one level up from an OSM data file of a geographic (sub)region.

Parameters
subregion_name (*str*) – validated name of a (sub)region available on a free download server

Returns
name of the directory one level up from a downloaded OSM data file

Return type
str

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> subrgn_name_ = 'England'
>>> BaseDownloader.make_subregion_dirname(subrgn_name_)
'england'
>>> subrgn_name_ = 'Greater London'
>>> BaseDownloader.make_subregion_dirname(subrgn_name_)
'greater-london'
```

GeofabrikDownloader.print_action_prompt

classmethod GeofabrikDownloader.**print_action_prompt**(*data_name='<data_name>'*, *verbose=False*, *confirmation_required=True*, *note=''*, *end=' ... '*)

Print a short message showing the action as a function runs.

Parameters

- **data_name** (*str*) – name of the prepacked data, defaults to `'<data_name>'`

- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **note** (*str*) – additional message, defaults to `""`
- **end** (*str*) – end string after printing the status message, defaults to `" ... "`

Returns
None.

Return type
None

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> BaseDownloader.print_action_prompt(verbose=False) is None  # Nothing is printed.
True
>>> BaseDownloader.print_action_prompt(verbose=True)
... print("Done.")
Retrieving/compiling the data ... Done.
>>> BaseDownloader.print_action_prompt(verbose=True, note="(Some notes)")
... print("Done.")
Retrieving/compiling the data (Some notes) ... Done.
>>> BaseDownloader.print_action_prompt(verbose=True, confirmation_required=False)
... print("Done.")
Retrieving/compiling data of <data_name> ... Done.
```

GeofabrikDownloader.print_status

classmethod GeofabrikDownloader.**print_status**(*data_name='<data_name>', path_to_file='<file_path>', verbose=False, error_message=None, update=False, raise_error=False*)

Print a short message for an otherwise situation.

Parameters

- **data_name** (*str*) – name of the prepacked data, defaults to `'<name_of_data>'`
- **path_to_file** (*str | os.PathLike[str]*) – pathname of the prepacked data file, defaults to `"<file_path>"`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **error_message** (*Exception | str | None*) – message of an error detected during execution of a function, defaults to `None`
- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`

- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
None.

Return type
None

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> BaseDownloader.print_status() is None  # Nothing will be printed.
True
>>> BaseDownloader.print_status(verbose=True)
Cancelled.
>>> BaseDownloader.print_status(verbose=2)
The collecting of <data_name> is cancelled, or no data is available.
>>> BaseDownloader.print_status(verbose=True, error_message="Errors.")
Failed. Errors.
```

GeofabrikDownloader.specify_sub_download_dir

`GeofabrikDownloader.`**`specify_sub_download_dir`**(*subregion_name*, *osm_file_format*, *download_dir=None*, ***kwargs*)

Specify a directory for downloading data of all subregions of a geographic (sub)region.

This is useful when the specified format of the data of a geographic (sub)region is not available at Geofabrik free download server.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on Geofabrik free download server
- **osm_file_format** (*str*) – file format/extension of the OSM data available on the download server
- **download_dir** (*str | None*) – directory for saving the downloaded file(s), defaults to `None`; when `download_dir=None`, it refers to *cdd_geofabrik()*. *cdd_geofabrik()*
- **kwargs** – [optional] parameters of pyhelpers.dirs.cd(), including `mkdir``(default: ``False`)

Returns
pathname of a download directory for downloading data of all subregions of the specified (sub)region and format

Return type
str

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> import os
>>> gfd = GeofabrikDownloader()
>>> subregion_name = 'london'
>>> osm_file_format = ".pbf"

>>> # Default download directory (if the requested data file is not available)
>>> download_pathname = gfd.specify_sub_download_dir(subregion_name, osm_file_format)
>>> os.path.dirname(os.path.relpath(download_pathname))
'osm_data\geofabrik\europe\united-kingdom\england\greater-london'

>>> # When a download directory is specified
>>> subregion_name = 'britain'
>>> osm_file_format = ".shp"
>>> download_dir = "tests/osm_data"
>>> download_pathname = gfd.specify_sub_download_dir(
...     subregion_name, osm_file_format, download_dir)
>>> os.path.relpath(download_pathname)
'tests\osm_data\great-britain-shp-zip'

>>> gfd_ = GeofabrikDownloader(download_dir=download_dir)
>>> download_pathname = gfd_.specify_sub_download_dir(subregion_name, osm_file_format)
>>> os.path.relpath(download_pathname)
'tests\osm_data\europe\great-britain\great-britain-shp-zip'
```

GeofabrikDownloader.validate_file_format

GeofabrikDownloader.**validate_file_format**(*osm_file_format*, *valid_formats=None*, *raise_error=True*, ***kwargs*)

Validate an input file format of OSM data.

The validation is done by matching the input to a filename extension available on Geofabrik free download server.

Parameters

- **osm_file_format** (*str*) – file format/extension of the OSM data on the free download server
- **valid_formats** (*Iterable*) – fil extensions of the data available on a free download server
- **raise_error** (*bool*) – (if the input fails to match a valid name) whether to raise the error `pydriosm.downloader.InvalidFileFormatError`, defaults to `True`
- **kwargs** – [optional] parameters of pyhelpers.text.find_similar_str()

Returns

formal file format

Return type

str

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()
>>> osm_file_format = ".pbf"
>>> gfd.validate_file_format(osm_file_format)
'.osm.pbf'
>>> osm_file_format = "shp"
>>> gfd.validate_file_format(osm_file_format)
'.shp.zip'
>>> osm_file_format = "geopackage"
>>> gfd.validate_file_format(osm_file_format)
'.gpkg.zip'
```

GeofabrikDownloader.validate_subregion_name

GeofabrikDownloader.**validate_subregion_name**(*subregion_name*, *valid_names=None*, *raise_error=False*, ***kwargs*)

Validate an input name of a geographic (sub)region.

The validation is done by matching the input to a name of a geographic (sub)region available on Geofabrik free download server.

Parameters

- **subregion_name** (*str*) – name/URL of a (sub)region available on Geofabrik free download server
- **valid_names** (*Iterable*) – names of all (sub)regions available on a free download server
- **raise_error** (*bool*) – (if the input fails to match a valid name) whether to raise the error `pydriosm.downloader.InvalidSubregionName`, defaults to `True`
- **kwargs** – [optional] parameters of pyhelpers.text.find_similar_str()

Returns

valid subregion name that matches (or is the most similar to) the input

Return type

str

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader
>>> gfd = GeofabrikDownloader()
>>> subregion_name = 'london'
>>> gfd.validate_subregion_name(subregion_name)
'Greater London'
>>> subregion_name = 'https://download.geofabrik.de/europe/united-kingdom.html'
>>> gfd.validate_subregion_name(subregion_name)
'United Kingdom'
```

GeofabrikDownloader.verify_download_dir

GeofabrikDownloader.**verify_download_dir**(*download_dir=None*, *verify_download_dir=True*)

Verify the pathname of the current download directory.

Parameters

- **download_dir** (*str | os.PathLike | None*) – directory for saving the downloaded file(s)
- **verify_download_dir** (*bool*) – whether to verify the pathname of the current download directory

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> import os
>>> bdl = BaseDownloader()
>>> os.path.relpath(bdl.download_dir)
'osm_data'
>>> bdl.verify_download_dir(download_dir='tests', verify_download_dir=True)
>>> os.path.relpath(bdl.download_dir)
'tests'
```

BBBikeDownloader

class pydriosm.downloader.**BBBikeDownloader**(*download_dir=None*, *update=False*, ***kwargs*)

Download OSM data from BBBike free download server.

Parameters

download_dir (*str | None*) – (a path or a name of) a directory for saving downloaded data files; if `download_dir=None` (default), the downloaded data files are saved into a folder named `'osm_data'` under the current working directory

Variables

- **valid_subregion_names** (*set*) – names of (sub)regions available on BBBike free download server
- **valid_file_formats** (*set*) – filename extensions of the data files available on BBBike free download server
- **subregion_index** (*pandas.DataFrame*) – index of download pages for all available (sub)regions
- **catalogue** (*pandas.DataFrame*) – a catalogue (index) of all available BBBike downloads
- **download_dir** (*str | None*) – name or pathname of a directory for saving downloaded data files (in accordance with the parameter `download_dir`)
- **data_pathnames** (*list*) – list of pathnames of all downloaded data files

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> import os
>>> bbd = BBBikeDownloader()
>>> bbd.NAME
'BBBike'
>>> bbd.LONG_NAME
'BBBike exports of OpenStreetMap data'
>>> bbd.URL
'https://download.bbbike.org/osm/bbbike/'
>>> os.path.relpath(bbd.download_dir)
'osm_data\bbbike'
>>> bbd = BBBikeDownloader(download_dir="tests\osm_data")
>>> os.path.relpath(bbd.download_dir)
'tests\osm_data'
```

Attributes

1anti-flashwhitewhite

DEFAULT_DOWNLOAD_DIR	Default download directory.
FILE_FORMATS	Valid file formats.
LONG_NAME	Full name of the data resource.
NAME	Name of the free downloader server.
URL	URL of the homepage to the free download server.

BBBikeDownloader.DEFAULT_DOWNLOAD_DIR

BBBikeDownloader.**DEFAULT_DOWNLOAD_DIR: str = 'osm_data/bbbike'**

Default download directory.

BBBikeDownloader.FILE_FORMATS

BBBikeDownloader.**FILE_FORMATS: set = {'.csv.xz', '.garmin-onroad-latin1.zip', '.garmin-ontrail-latin1.zip', '.garmin-opentopo-latin1.zip', '.garmin-osm.zip', '.geojson.xz', '.gz', '.mapsforge-osm.zip', '.mbtiles-openmaptiles.zip', '.pbf', '.shp.zip'}**

Valid file formats.

BBBikeDownloader.LONG_NAME

BBBikeDownloader.**LONG_NAME: str = 'BBBike exports of OpenStreetMap data'**

Full name of the data resource.

BBBikeDownloader.NAME

BBBikeDownloader.**NAME: str = 'BBBike'**

Name of the free downloader server.

BBBikeDownloader.URL

BBBikeDownloader.**URL: str = 'https://download.bbbike.org/osm/bbbike/'**

URL of the homepage to the free download server.

Methods

1anti-flashwhitewhite

cdd(*sub_dir[, mkdir])	Change directory to default download directory and its subdirectories or a specific file.
download_data(subregion_names, osm_file_formats)	Download OSM data (of a specific file format) of one (or multiple) geographic (sub)region(s).
file_exists(subregion_name, osm_file_format)	Check if a requested data file of a geographic (sub)region already exists locally, given its default filename.
file_exists_and_more(subregion_names, ...[, ...])	Check if a requested data file already exists and compile information for downloading the data.
format_confirmation_prompt([data_name, ...])	Compose a short message to be printed for confirmation.
get_bbbike_cities([update, ...])	Get the names of all the available cities.
get_bbbike_cities_poly([update, ...])	Get location information of all cities available on the download server.
get_catalogue([update, ...])	Get a dict-type index of available formats, data types and a download catalogue.
get_default_sub_path(subregion_name_, ...)	Get default sub path for saving OSM data file of a geographic (sub)region.
get_prepacked_data(meth[, data_name, ...])	Get auxiliary data (that is to be prepacked in the package).
get_sub_catalogue(subregion_name[, update, ...])	Get a download catalogue of OSM data available for a given geographic (sub)region.
get_subregion_download_url(subregion_na ...)	Get a valid URL for downloading OSM data of a specific file format for a geographic (sub)region.
get_subregion_index([update, ...])	Get a catalogue for geographic (sub)regions.
get_valid_download_info(subregion_name, ...)	Get information of downloading (or downloaded) data file.
get_valid_subregion_names([update, ...])	Get a list of names of all geographic (sub)regions.
make_subregion_dirname(subregion_name_)	Make the name of the directory one level up from an OSM data file of a geographic (sub)region.
print_action_prompt([data_name, verbose, ...])	Print a short message showing the action as a function runs.

continues on next page

Table 6 – continued from previous page

print_status([data_name, path_to_file, ...])	Print a short message for an otherwise situation.
validate_file_format(osm_file_format[, ...])	Validate an input file format of OSM data.
validate_subregion_name(subregion_name[...])	Validate an input name of a geographic (sub)region.
verify_download_dir([download_dir, ...])	Verify the pathname of the current download directory.

BBBikeDownloader.cdd

classmethod BBBikeDownloader.**cdd**(**sub_dir*, *mkdir=False*, ***kwargs*)

Change directory to default download directory and its subdirectories or a specific file.

Parameters

- **sub_dir** (*str | os.PathLike[str]*) – name of directory; names of directories (and/or a filename)
- **mkdir** (*bool*) – whether to create a directory, defaults to False
- **kwargs** – [optional] parameters of pyhelpers.dirs.cd()

Returns

an absolute pathname to a directory (or a file)

Return type

str | os.PathLike[str]

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> import os
>>> os.path.relpath(BaseDownloader.cdd())
'osm_data'
```

BBBikeDownloader.download_data

BBBikeDownloader.**download_data**(*subregion_names*, *osm_file_formats*, *download_dir=None*, *update=False*, *confirmation_required=True*, *interval=None*, *verify_download_dir=True*, *verbose=False*, *ret_download_path=False*, ***kwargs*)

Download OSM data (of a specific file format) of one (or multiple) geographic (sub)region(s).

Parameters

- **subregion_names** (*str | list*) – name of a geographic (sub)region (or names of multiple geographic (sub)regions) available on BBBike free download server

- **osm_file_formats** (*str*) – file format/extension of the OSM data available on the download server
- **download_dir** (*str | None*) – directory for saving the downloaded file(s), defaults to None; when download_dir=None, it refers to *cdd_bbbike()*.
- **update** (*bool*) – whether to update the data if it already exists, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True
- **interval** (*int | float | None*) – interval (in second) between downloading two subregions, defaults to None
- **verify_download_dir** (*bool*) – whether to verify the pathname of the current download directory, defaults to True
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to False
- **ret_download_path** (*bool*) – whether to return the path(s) to the downloaded file(s), defaults to False

Returns

the path(s) to the downloaded file(s) when ret_download_path is True

Return type

list | str

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> from pyhelpers.dirs import delete_dir
>>> import os
>>> bbd = BBBikeDownloader()
>>> # Download BBBike PBF data of London
>>> subregion_name = 'London'
>>> osm_file_format = "pbf"
>>> bbd.download_data(subregion_name, osm_file_format, verbose=True)
Proceed to download data in the format '.pbf' for the following geographic (sub)reg...
    "London"
  to "./osm_data/bbbike/london/"
? [No]|Yes: yes
Downloading "London.osm.pbf" 100%|■■■■■■■■■■■| 188M/188M | 1.06MB/s | ETA: 00:00
  Saving "London.osm.pbf" to "./osm_data/bbbike/london/" ... Done.
>>> len(bbd.data_paths)
1
>>> os.path.relpath(bbd.data_paths[0])  # (on Windows)
'osm_data\bbbike\london\London.osm.pbf'
>>> london_download_dir = os.path.relpath(bbd.download_dir)
>>> london_download_dir  # (on Windows)
'osm_data\bbbike'

>>> # Download PBF data of Leeds and Birmingham to a given directory
>>> subregion_names = ['Leeds', 'Birmingham']
>>> osm_file_format = ['shp', 'pbf']
```

(continues on next page)

(continued from previous page)

```
>>> download_dir = "tests/osm_data"
>>> download_paths = bbd.download_data(
...     subregion_names, osm_file_format, download_dir, verbose=True,
...     ret_download_path=True)
Proceed to download data in the formats ('.shp.zip', '.pbf') for the following geog...
    "Birmingham"
    "Leeds"
  to "./tests/osm_data/"
? [No]|Yes: >? yes
Downloading "Leeds.osm.shp.zip" 100%|■■■■■■■■■■■| 57.8M/57.8M | 18.3MB/s | ETA...
  Saving "Leeds.osm.shp.zip" to "./tests/osm_data/leeds/" ... Done.
Downloading "Leeds.osm.pbf" 100%|■■■■■■■■■■■| 38.2M/38.2M | 14.9MB/s | ETA: 00:00
  Saving "Leeds.osm.pbf" to "./tests/osm_data/leeds/" ... Done.
Downloading "Birmingham.osm.shp.zip" 100%|■■■■■■■■■■■| 79.1M/79.1M | 18.0MB/s ...
  Saving "Birmingham.osm.shp.zip" to "./tests/osm_data/birmingham/" ... Done.
Downloading "Birmingham.osm.pbf" 100%|■■■■■■■■■■■| 56.0M/56.0M | 17.2MB/s | ET...
  Saving "Birmingham.osm.pbf" to "./tests/osm_data/birmingham/" ... Done.
>>> len(download_paths)
4
>>> len(bbd.data_paths)
5
>>> os.path.relpath(bbd.download_dir) == os.path.relpath(download_dir)
True
>>> os.path.relpath(os.path.commonpath(download_paths))  # (on Windows)
'tests\osm_data'

>>> # Delete the above download directories
>>> delete_dir([os.path.dirname(london_download_dir), download_dir], verbose=True)
To delete the following directories:
  "./osm_data/" (Not empty)
  "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting:
  "./osm_data/" ... Done.
  "./tests/osm_data/" ... Done.
```

BBBikeDownloader.file_exists

BBBikeDownloader.**file_exists**(*subregion_name*, *osm_file_format*, *data_dir=None*, *update=False*, *verbose=False*, *ret_file_path=False*)

Check if a requested data file of a geographic (sub)region already exists locally, given its default filename.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on BBBike free download server
- **osm_file_format** (*str*) – file format/extension of the OSM data available on the download server
- **data_dir** (*str | None*) – directory where the data file (or files) is (or are) stored, defaults to None; when data_dir=None, it refers to *cdd_bbbike()*.
- **update** (*bool*) – whether to (check and) update the data, defaults to False

- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **ret_file_path** (*bool*) – whether to return the path to the data file (if it exists), defaults to `False`

Returns
whether the requested data file exists; or the path to the data file

Return type
bool | str

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> from pyhelpers.dirs import delete_dir
>>> import os
>>> bbd = BBBikeDownloader()
>>> subregion_name = 'birmingham'
>>> osm_file_format = ".shp"
>>> data_dir = "tests/osm_data"
>>> # Check whether the PBF data file exists; `ret_file_path` is by default `False`
>>> pbf_exists = bbd.file_exists(subregion_name, osm_file_format, data_dir)
>>> pbf_exists
False
>>> # Download the PBF data of Birmingham (to the default directory)
>>> bbd.download_data(subregion_name, osm_file_format, data_dir, verbose=True)
Proceed to download data in the format '.shp.zip' for the following geographic (sub...
    "Birmingham"
  to "./tests/osm_data/birmingham/"
? [No]|Yes: yes
Downloading "Birmingham.osm.shp.zip" 100%|■■■■■■■■■■| 79.0M/79.0M | 949kB/s |...
  Saving "Birmingham.osm.shp.zip" to "./tests/osm_data/birmingham/" ... Done.
>>> bbd.file_exists(subregion_name, osm_file_format, data_dir)
True
>>> # Set `ret_file_path=True`
>>> pbf_pathname = bbd.file_exists(subregion_name, osm_file_format, ret_file_path=True)
>>> os.path.relpath(pbf_pathname)
'tests\osm_data\birmingham\Birmingham.osm.pbf'
>>> os.path.relpath(data_dir) == os.path.relpath(bbd.download_dir)
True
>>> # Remove the directory or the PBF file and check again:
>>> delete_dir(bbd.download_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
>>> # Since the default download directory has been deleted
>>> bbd.file_exists(subregion_name, osm_file_format, data_dir)
False
```

BBBikeDownloader.file_exists_and_more

BBBikeDownloader.**file_exists_and_more**(*subregion_names*, *osm_file_formats*, *data_dir=None*, *update=False*, *confirmation_required=True*, *verbose=True*, *deep=False*)

Check if a requested data file already exists and compile information for downloading the

data.

Parameters

- **subregion_names** (*str | list*) – name(s) of geographic (sub)region(s) available on a free download server
- **osm_file_formats** (*str*) – file format of the OSM data available on the free download server
- **data_dir** (*str | None*) – directory where the data file (or files) is (or are) stored, defaults to `None`
- **update** (*bool*) – whether to (check on and) update the data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `True`
- **deep** (*bool*) – whether to further check availability of sub-subregions data, defaults to `False`

Returns

whether the requested data file exists; or the path to the data file

Return type

tuple

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader, BBBikeDownloader
>>> gfd = GeofabrikDownloader()
>>> gfd.file_exists_and_more('London', ".pbf")
(['Greater London'],
 ['.osm.pbf'],
 True,
 'Proceed to download data in the format '.osm.pbf' for the following geographic ...
 [])
>>> gfd.file_exists_and_more(['london', 'rutland'], ".pbf")
(['Greater London', 'Rutland'],
 ['.osm.pbf'],
 True,
 'Proceed to download data in the format '.osm.pbf' for the following geographic ...
 [])
>>> gfd.file_exists_and_more(['london', 'rutland'], ["shp", ".pbf"])
(['Greater London', 'Rutland'],
 ['.shp.zip', '.osm.pbf'],
 True,
 'Proceed to download data in the formats ('.shp.zip', '.osm.pbf') for the foll...
 [])
>>> bbd = BBBikeDownloader()
>>> bbd.file_exists_and_more('London', ".pbf")
(['London'],
 ['.pbf'],
 True,
```

(continues on next page)

(continued from previous page)

```
 'Proceed to download data in the format '.pbf' for the following geographic (sub...
 [])
>>> bbd.file_exists_and_more(['birmingham', 'leeds'], ".pbf")
(['Birmingham', 'Leeds'],
 ['.pbf'],
 True,
 'Proceed to download data in the format '.pbf' for the following geographic (sub...
 [])
```

BBBikeDownloader.format_confirmation_prompt

classmethod BBBikeDownloader.**format_confirmation_prompt**(*data_name='<data_name>'*, *file_path='<file_path>'*, *update=False*, *note=''*)

Compose a short message to be printed for confirmation.

Parameters

- **data_name** (*str*) – name of the prepacked data. Defaults to '<data_name>'.
- **file_path** (*str | os.PathLike*) – pathname of the prepacked data file. Defaults to "<file_path>".
- **update** (*bool*) – whether to (check on and) update the prepacked data. Defaults to False.
- **note** (*str*) – additional message. Defaults to "".

Returns

a short message to be printed for confirmation.

Return type

str

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> BaseDownloader.format_confirmation_prompt()
'Proceed to retrieve/compile data of <data_name>\n?'
>>> BaseDownloader.format_confirmation_prompt(update=True)
'Proceed to update the data of <data_name>\n?'
```

BBBikeDownloader.get_bbbike_cities

classmethod BBBikeDownloader.**get_bbbike_cities**(*update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Get the names of all the available cities.

Parameters

- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to False

- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
list of names of cities available on BBBike free download server

Return type
list | None

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> bbd = BBBikeDownloader()
>>> bbbike_cities = bbd.get_bbbike_cities()
>>> bbbike_cities[:5]
['Aachen', 'Aarhus', 'Adelaide', 'Albuquerque', 'Alexandria']
```

BBBikeDownloader.get_bbbike_cities_poly

classmethod BBBikeDownloader.**get_bbbike_cities_poly**(*update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Get location information of all cities available on the download server.

Parameters

- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
location information of BBBike cities, i.e. geographic (sub)regions

Return type
pandas.DataFrame | None

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> bbd = BBBikeDownloader()
>>> bbbike_city_polygons = bbd.get_bbbike_cities_poly()
```

(continues on next page)

(continued from previous page)

```
>>> type(bbbike_city_polygons)
pandas.DataFrame
>>> bbbike_city_polygons.shape
(238, 2)
>>> bbbike_city_polygons.head()
          name                                           geometry
0       Aachen  POLYGON ((5.88 50.6, 6.58 50.6, 6.58 50.99, 5....
1       Aarhus  POLYGON ((9.82 55.99, 10.37 55.99, 10.37 56.29...
2     Adelaide  POLYGON ((138.46 -35.03, 138.74 -35.03, 138.74...
3  Albuquerque  POLYGON ((-106.8 35, -106.47 35, -106.47 35.22...
4   Alexandria  POLYGON ((29.7 31.02, 30.21 31.02, 30.21 31.34...
```

BBBikeDownloader.get_catalogue

classmethod BBBikeDownloader.**get_catalogue**(*update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Get a dict-type index of available formats, data types and a download catalogue.

Parameters

- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

a list of available formats, a list of available data types and a dictionary of download catalogue

Return type

dict[str, list | dict[str, pandas.DataFrame]]

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> bbd = BBBikeDownloader()
>>> # Index for downloading OSM data available on the BBBike free download server
>>> bbbike_catalogue = bbd.get_catalogue()
>>> list(bbbike_catalogue.keys())
['FileFormat', 'DataType', 'Catalogue']
>>> catalogue = bbbike_catalogue['Catalogue']
>>> type(catalogue)
dict
>>> bham_catalogue = catalogue['Birmingham']
>>> type(bham_catalogue)
pandas.core.frame.DataFrame
>>> bham_catalogue.shape
```

(continues on next page)

(continued from previous page)

```
(13, 5)
>>> bham_catalogue.head()
                                  filename  ...                last_update
0                       Birmingham.osm.pbf  ...  2025-09-27 18:51:39+00:00
1                        Birmingham.osm.gz  ...  2025-09-28 05:37:57+00:00
2                  Birmingham.osm.shp.zip   ...  2025-09-28 05:54:16+00:00
3  Birmingham.osm.garmin-ontrail-latin1.zip  ...  2025-09-28 07:05:18+00:00
4   Birmingham.osm.garmin-onroad-latin1.zip  ...  2025-09-28 07:04:54+00:00
[5 rows x 5 columns]
```

BBBikeDownloader.get_default_sub_path

classmethod BBBikeDownloader.**get_default_sub_path**(*subregion_name_*, *download_url*)

Get default sub path for saving OSM data file of a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – validated name of a (sub)region available on a free download server
- **download_url** (*str*) – download URL of a geographic (sub)region

Returns

default sub path

Return type

str | os.PathLike[str]

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader

>>> subrgn_name_ = 'London'
>>> dwnld_url = 'https://download.bbbike.org/osm/bbbike/London/London.osm.pbf'
>>> BaseDownloader.get_default_sub_path(subrgn_name_, dwnld_url)
'\london'
```

BBBikeDownloader.get_prepacked_data

classmethod BBBikeDownloader.**get_prepacked_data**(*meth*, *data_name='<data_name>'*, *file_stem=None*, *ext='.pkl.xz'*, *update=False*, *confirmation_required=True*, *dump_backup=True*, *verbose=False*, *confirmation_prompt_note=''*, *action_prompt_note=''*, *action_prompt_end=' ... '*, *ending_message='Done.'*, *raise_error=False*, ***kwargs*)

Get auxiliary data (that is to be prepacked in the package).

Parameters

- **meth** (*Callable*) – name of a class method for getting (auxiliary) prepacked data
- **data_name** (*str*) – name of the prepacked data, defaults to '<data_name>'
- **file_stem** – The filename (without file extension) that overrides data_name for determining file path. Defaults to None.
- **ext** (*str*) – File extension of the filename of prepacked data; defaults to ".pkl".
- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True
- **dump_backup** (*bool*) – If True, save a cached data file.
- **ending_message** (*str*) – The ending message to print.
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to False
- **confirmation_prompt_note** (*str*) – additional message for the method compose_cfm_msg(), defaults to ""
- **action_prompt_note** (*str*) – equivalent of the parameter note of the method print_action_msg(), defaults to ""
- **action_prompt_end** (*str*) – equivalent of the parameter end of the method print_action_msg(), defaults to " ... "
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Returns
auxiliary data

Return type
Any

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> data = BaseDownloader.get_prepacked_data(print, verbose=True, raise_error=True)
Proceed to retrieve/compile data of <data_name>
? [No]|Yes: yes
Retrieving/compiling the data ...
Done.
>>> data is None
True
```

BBBikeDownloader.get_sub_catalogue

BBBikeDownloader.**get_sub_catalogue**(*subregion_name*, *update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Get a download catalogue of OSM data available for a given geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on BBBike free download server
- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

a catalogues for subregion downloads

Return type

pandas.DataFrame | None

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> bbd = BBBikeDownloader()
>>> subregion_name = 'birmingham'
>>> # A download catalogue for Leeds
>>> bham_catalogue = bbd.get_sub_catalogue(subregion_name, verbose=True)
To retrieve/compile data of a download catalogue for "Birmingham"
? [No]|Yes: yes
Retrieving/compiling the data ... Done.
>>> bham_catalogue.shape
(15, 5)
>>> bham_catalogue.head()
                                  filename  ...                last_update
0                       Birmingham.osm.pbf  ...  2026-02-28 18:54:17+00:00
1                        Birmingham.osm.gz  ...  2026-03-01 04:52:14+00:00
2                  Birmingham.osm.shp.zip  ...  2026-03-01 05:40:06+00:00
3  Birmingham.osm.garmin-ontrail-latin1.zip  ...  2026-03-01 06:15:16+00:00
4   Birmingham.osm.garmin-onroad-latin1.zip  ...  2026-03-01 06:14:56+00:00
[5 rows x 5 columns]
>>> bham_catalogue.columns.tolist()
['filename', 'url', 'data_type', 'size', 'last_update']
```

BBBikeDownloader.get_subregion_download_url

BBBikeDownloader.**get_subregion_download_url**(*subregion_name*, *osm_file_format*, *update=False*, *verbose=False*, *raise_error=True*)

Get a valid URL for downloading OSM data of a specific file format for a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on BBBike free download server
- **osm_file_format** (*str*) – file format/extension of the OSM data available on the download server
- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – (if the input fails to match a valid name) whether to raise an *pydriosm.errors.InvalidSubregionNameError* or *InvalidFileFormatError*; defaults to `True`

Returns

a valid name of `subregion_name` and a download URL for the given `osm_file_format`

Return type

tuple

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> bbd = BBBikeDownloader()
>>> subregion_name, osm_file_format = 'birmingham', "pbf"
>>> # Get a valid subregion name and its download URL
>>> subregion_name_, download_url = bbd.get_subregion_download_url(
...     subregion_name, osm_file_format)
>>> subregion_name_
'Birmingham'
>>> download_url
'https://download.bbbike.org/osm/bbbike/Birmingham/Birmingham.osm.pbf'
>>> osm_file_format = "csv.xz"
>>> subregion_name_, download_url = bbd.get_subregion_download_url(
...     subregion_name, osm_file_format)
>>> subregion_name_
'Birmingham'
>>> download_url
'https://download.bbbike.org/osm/bbbike/Birmingham/Birmingham.osm.csv.xz'
```

BBBikeDownloader.get_subregion_index

classmethod BBBikeDownloader.**get_subregion_index**(*update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Get a catalogue for geographic (sub)regions.

Parameters

- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

catalogue for subregions of BBBike data

Return type

pandas.DataFrame | None

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> bbd = BBBikeDownloader()
>>> # A BBBike catalogue of geographic (sub)regions
>>> subregion_index = bbd.get_subregion_index()
>>> subregion_index.shape
(238, 3)
>>> subregion_index.head()
          name  ...                                                url
0       Aachen  ...     https://download.bbbike.org/osm/bbbike/Aachen/
1       Aarhus  ...     https://download.bbbike.org/osm/bbbike/Aarhus/
2     Adelaide  ...   https://download.bbbike.org/osm/bbbike/Adelaide/
3  Albuquerque  ...  https://download.bbbike.org/osm/bbbike/Albuque...
4   Alexandria  ...  https://download.bbbike.org/osm/bbbike/Alexand...
[5 rows x 3 columns]
>>> subregion_index.columns.to_list()
['name', 'last_modified', 'url']
```

BBBikeDownloader.get_valid_download_info

BBBikeDownloader.**get_valid_download_info**(*subregion_name*, *osm_file_format*, *download_dir=None*, ***kwargs*)

Get information of downloading (or downloaded) data file.

The information includes a valid subregion name, a default filename, a URL and an absolute path where the data file is (to be) saved locally.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on BBBike free download server
- **osm_file_format** (*str*) – file format/extension of the OSM data available on the download server
- **download_dir** (*str | None*) – directory for saving the downloaded file(s), defaults to None; when download_dir=None, it refers to *cdd_bbbike()*.

Returns
valid subregion name, filename, download url and absolute file path

Return type
tuple

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> import os
>>> bbd = BBBikeDownloader()
>>> subregion_name = 'birmingham'
>>> osm_file_format = "shp"
>>> # valid subregion name, filename, download url and absolute file path
>>> subregion_name_, pbf_filename, download_url, pbf_pathname =             ...     bbd.
↪get_valid_download_info(subregion_name, osm_file_format)
>>> subregion_name_
'Birmingham'
>>> pbf_filename
'Birmingham.osm.shp.zip'
>>> download_url
'https://download.bbbike.org/osm/bbbike/Birmingham/Birmingham.osm.shp.zip'
>>> os.path.relpath(pbf_pathname)  # (on Windows)
'osm_data\bbbike\birmingham\Birmingham.osm.shp.zip'
>>> # Create a new instance with a given download directory
>>> bbd = BBBikeDownloader(download_dir="tests/osm_data")
>>> _, _, _, pbf_pathname = bbd.get_valid_download_info(subregion_name, osm_file_format)
>>> os.path.relpath(pbf_pathname)  # (Returns the same pathname on Windows)
'tests\osm_data\birmingham\Birmingham.osm.shp.zip'
```

BBBikeDownloader.get_valid_subregion_names

classmethod BBBikeDownloader.**get_valid_subregion_names**(*update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Get a list of names of all geographic (sub)regions.

Parameters

- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True

- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to False
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Returns
a list of geographic (sub)region names available on BBBike free download server

Return type
list | None

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> bbd = BBBikeDownloader()
>>> # A list of names of all BBBike geographic (sub)regions
>>> subregion_names = bbd.get_valid_subregion_names()
>>> type(subregion_names)
list
>>> len(subregion_names)
237
```

BBBikeDownloader.make_subregion_dirname

classmethod BBBikeDownloader.**make_subregion_dirname**(*subregion_name_*)

Make the name of the directory one level up from an OSM data file of a geographic (sub)region.

Parameters
subregion_name (*str*) – validated name of a (sub)region available on a free download server

Returns
name of the directory one level up from a downloaded OSM data file

Return type
str

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> subrgn_name_ = 'England'
>>> BaseDownloader.make_subregion_dirname(subrgn_name_)
'england'
>>> subrgn_name_ = 'Greater London'
>>> BaseDownloader.make_subregion_dirname(subrgn_name_)
'greater-london'
```

BBBikeDownloader.print_action_prompt

classmethod BBBikeDownloader.**print_action_prompt**(*data_name='<data_name>'*, *verbose=False*, *confirmation_required=True*, *note=''*, *end=' ... '*)

Print a short message showing the action as a function runs.

Parameters

- **data_name** (*str*) – name of the prepacked data, defaults to '<data_name>'
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True
- **note** (*str*) – additional message, defaults to ""
- **end** (*str*) – end string after printing the status message, defaults to " ... "

Returns
None.

Return type
None

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> BaseDownloader.print_action_prompt(verbose=False) is None  # Nothing is printed.
True
>>> BaseDownloader.print_action_prompt(verbose=True)
... print("Done.")
Retrieving/compiling the data ... Done.
>>> BaseDownloader.print_action_prompt(verbose=True, note="(Some notes)")
... print("Done.")
Retrieving/compiling the data (Some notes) ... Done.
>>> BaseDownloader.print_action_prompt(verbose=True, confirmation_required=False)
... print("Done.")
Retrieving/compiling data of <data_name> ... Done.
```

BBBikeDownloader.print_status

classmethod BBBikeDownloader.**print_status**(*data_name='<data_name>'*, *path_to_file='<file_path>'*, *verbose=False*, *error_message=None*, *update=False*, *raise_error=False*)

Print a short message for an otherwise situation.

Parameters

- **data_name** (*str*) – name of the prepacked data, defaults to '<name_of_data>'

- **path_to_file** (*str | os.PathLike[str]*) – pathname of the prepacked data file, defaults to "<file_path>"
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to False
- **error_message** (*Exception | str | None*) – message of an error detected during execution of a function, defaults to None
- **update** (*bool*) – whether to (check on and) update the prepacked data, defaults to False
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Returns
None.

Return type
None

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> BaseDownloader.print_status() is None  # Nothing will be printed.
True
>>> BaseDownloader.print_status(verbose=True)
Cancelled.
>>> BaseDownloader.print_status(verbose=2)
The collecting of <data_name> is cancelled, or no data is available.
>>> BaseDownloader.print_status(verbose=True, error_message="Errors.")
Failed. Errors.
```

BBBikeDownloader.validate_file_format

BBBikeDownloader.**validate_file_format**(*osm_file_format*, *valid_formats=None*, *raise_error=True*, ***kwargs*)

Validate an input file format of OSM data.

The validation is done by matching the input osm_file_format to a filename extension available on BBBike free download server.

Parameters

- **osm_file_format** (*str*) – file format/extension of the OSM data available on BBBike free download server
- **valid_formats** (*Collection | None*) – fil extensions of the data available on a free download server
- **raise_error** (*bool*) – (if the input fails to match a valid name) whether to raise the error pydriosm.downloader.InvalidFileFormatError, defaults to True

Returns
valid file format (file extension)

Return type
str

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> bbd = BBBikeDownloader()
>>> osm_file_format_ = bbd.validate_file_format(osm_file_format='PBF')
>>> osm_file_format_
'.pbf'
>>> osm_file_format_ = bbd.validate_file_format(osm_file_format='.osm.pbf')
>>> osm_file_format_
'.pbf'
```

BBBikeDownloader.validate_subregion_name

BBBikeDownloader.**validate_subregion_name**(*subregion_name*, *valid_names=None*, *raise_error=True*, ***kwargs*)

Validate an input name of a geographic (sub)region.

The validation is done by matching the input `subregion_name` to a name of a geographic (sub)region available on BBBike free download server.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on BBBike free download server
- **valid_names** (*Collection | None*) – names of all (sub)regions available on a free download server
- **raise_error** (*bool*) – (if the input fails to match a valid name) whether to raise the error `pydriosm.downloader.InvalidSubregionName`, defaults to `True`

Returns
valid (sub)region name that matches, or is the most similar to, the input

Return type
str

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> bbd = BBBikeDownloader()
>>> subregion_name = 'birmingham'
>>> subregion_name_ = bbd.validate_subregion_name(subregion_name)
>>> subregion_name_
'Birmingham'
```

BBBikeDownloader.verify_download_dir

BBBikeDownloader.**verify_download_dir**(*download_dir=None*, *verify_download_dir=True*)

Verify the pathname of the current download directory.

Parameters

- **download_dir** (*str | os.PathLike | None*) – directory for saving the downloaded file(s)
- **verify_download_dir** (*bool*) – whether to verify the pathname of the current download directory

Examples:

```
>>> from pydriosm.downloader._base import BaseDownloader
>>> import os
>>> bdl = BaseDownloader()
>>> os.path.relpath(bdl.download_dir)
'osm_data'
>>> bdl.verify_download_dir(download_dir='tests', verify_download_dir=True)
>>> os.path.relpath(bdl.download_dir)
'tests'
```

4.2 reader

Read the OSM data extracts in various file formats.

4.2.1 Parse OSM data

1anti-flashwhitewhite

SHP()	Read/parse Shapefile data.
PBF()	Read/parse PBF data.
VAR()	Read/parse OSM data of various formats (other than PBF and Shapefile).

SHP

class pydriosm.reader._shp.**SHP**

Read/parse Shapefile data.

Examples:

```
>>> from pydriosm.reader._shp import SHP

>>> SHP.EPSG4326_WGS84_PROJ4
'+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs'

>>> SHP.EPSG4326_WGS84_PROJ4_
{'proj': 'longlat', 'ellps': 'WGS84', 'datum': 'WGS84', 'no_defs': True}
```

Attributes

1anti-flashwhitewhite

ENCODING	The encoding method applied to create an OSM shapefile.
EPSG4326_WGS84_ESRI_WKT	The metadata associated with the shapefiles coordinate and projection system.
EPSG4326_WGS84_PROJ4	Proj4 of EPSG Projection 4326 - WGS 84 (EPSG:4326) for the setting of CRS for shapefile data.
*EPSG4326_WGS84_PROJ4_*	A dict-type representation of EPSG Projection 4326 - WGS 84 (EPSG:4326) for the setting of CRS for shapefile data.
LAYER_NAMES	Valid layer names for an OSM shapefile.
SHAPE_TYPE_GEOM	Shape type codes of shapefiles and their corresponding geometric objects defined in Shapely.
SHAPE_TYPE_GEOM_NAME	Shape type codes of shapefiles and their corresponding geometry object names
SHAPE_TYPE_NAME_LOOKUP	Shape type codes of shapefiles and their corresponding names for an OSM shapefile.
VECTOR_DRIVER	Name of the vector driver for writing shapefile data; see also the parameter `driver` of geopandas.GeoDataFrame.to_file().

SHP.ENCODING

SHP.**ENCODING: str = 'UTF-8'**

The encoding method applied to create an OSM shapefile. This is for writing .cpg (code page) file.

SHP.EPSG4326_WGS84_ESRI_WKT

SHP.**EPSG4326_WGS84_ESRI_WKT: str = 'GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]**

The metadata associated with the shapefiles coordinate and projection system. ESRI WKT of EPSG Projection 4326 - WGS 84 (EPSG:4326) for shapefile data.

SHP.EPSG4326_WGS84_PROJ4

SHP.**EPSG4326_WGS84_PROJ4: str = '+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs'**

Proj4 of EPSG Projection 4326 - WGS 84 (EPSG:4326) for the setting of CRS for shapefile data.

SHP.EPSG4326_WGS84_PROJ4_

SHP.**EPSG4326_WGS84_PROJ4_: dict = {'datum': 'WGS84', 'ellps': 'WGS84', 'no_defs': True, 'proj': 'longlat'}**

A dict-type representation of EPSG Projection 4326 - WGS 84 (EPSG:4326) for the setting of CRS for shapefile data.

SHP.LAYER_NAMES

SHP.**LAYER_NAMES: set = {'adminareas', 'buildings', 'landuse', 'natural', 'places', 'pofw', 'points', 'pois', 'protected_areas', 'railways', 'roads', 'traffic', 'transport', 'water', 'waterways'}**

Valid layer names for an OSM shapefile.

SHP.SHAPE_TYPE_GEOM

SHP.**SHAPE_TYPE_GEOM = {1: <class 'shapely.geometry.point.Point'>, 3: <class 'shapely.geometry.linestring.LineString'>, 5: <class 'shapely.geometry.polygon.Polygon'>, 8: <class 'shapely.geometry.multipoint.MultiPoint'>}**

Shape type codes of shapefiles and their corresponding geometric objects defined in Shapely.

Type
dict

SHP.SHAPE_TYPE_GEOM_NAME

SHP.**SHAPE_TYPE_GEOM_NAME: dict = {1: 'Point', 3: 'LineString', 5: 'Polygon', 8: 'MultiPoint'}**

Shape type codes of shapefiles and their corresponding geometry object names

SHP.SHAPE_TYPE_NAME_LOOKUP

SHP.**SHAPE_TYPE_NAME_LOOKUP: dict = {0: None, 1: 'Point', 3: 'Polyline', 5: 'Polygon', 8: 'MultiPoint', 11: 'PointZ', 13: 'PolylineZ', 15: 'PolygonZ', 18: 'MultiPointZ', 21: 'PointM', 23: 'PolylineM', 25: 'PolygonM', 28: 'MultiPointM', 31: 'MultiPatch'}**

Shape type codes of shapefiles and their corresponding names for an OSM shapefile.

SHP.VECTOR_DRIVER

SHP.**VECTOR_DRIVER: str = 'ESRI Shapefile'**

Name of the vector driver for writing shapefile data; see also the parameter `driver` of geopandas.GeoDataFrame.to_file().

Methods

1anti-flashwhitewhite

get_layer_name(shp_filename)	Find the layer name of OSM shapefile given its filename.
merge_layers(shp_zip_pathnames, layer_name)	Merge shapefiles over a layer for multiple geographic regions.
merge_shps(shp_pathnames, path_to_merged_dir)	Merge multiple shapefiles.
read_layer_shps(shp_pathnames[, ...])	Read a layer of OSM shapefile data.
read_shp(shp_pathname[, engine, emulate_gpd])	Read a shapefile.
unzip_shp_zip(shp_zip_pathname[, ...])	Unzip a zipped shapefile.
validate_layer_names(layer_names)	Validate the input of layer name(s) for reading shapefiles.
write_to_shapefile(data, write_to[, ...])	Save .shp data as a shapefile by PyShp.

SHP.get_layer_name

classmethod SHP.**get_layer_name**(*shp_filename*)

Find the layer name of OSM shapefile given its filename.

Parameters
shp_filename (*str*) – filename of a shapefile (.shp)

Returns
layer name of the shapefile

Return type
str

Examples:

```
>>> from pydriosm.reader._shp import SHP

>>> SHP.get_layer_name("") is None
True

>>> SHP.get_layer_name("gis_osm_railways_free_1.shp")
'railways'

>>> SHP.get_layer_name("gis_osm_transport_a_free_1.shp")
'transport'
```

SHP.merge_layers

classmethod SHP.**merge_layers**(*shp_zip_pathnames*, *layer_name*, *engine='geopandas'*, *rm_zip_extracts=True*, *output_dir=None*, *rm_shp_temp=True*, *ret_shp_pathname=False*, *verbose=False*, *raise_error=False*)

Merge shapefiles over a layer for multiple geographic regions.

Parameters

- **shp_zip_pathnames** (*list*) – list of paths to data of shapefiles (in .shp.zip format)
- **layer_name** (*str*) – name of a layer (e.g. 'railways')
- **engine** (*str*) – the open-source package used to merge/save shapefiles; options include: `'pyshp'` and `'geopandas'` (default) (or `'gpd'`) if `engine='geopandas'`, this function relies on geopandas.GeoDataFrame.to_file(); otherwise, it by default uses shapefile.Writer()
- **rm_zip_extracts** (*bool*) – whether to delete the extracted files, defaults to `False`
- **rm_shp_temp** (*bool*) – whether to delete temporary layer files, defaults to `False`
- **output_dir** (*str | None*) – if `None` (default), use the layer name as the name of the folder where the merged .shp files will be saved
- **ret_shp_pathname** (*bool*) – whether to return the pathname of the merged .shp file, defaults to `False`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

the path to the merged file when `ret_merged_shp_path=True`

Return type

list

Note

- This function does not create projection (.prj) for the merged map. See also [MMS-1].
- For valid `layer_name`, check the function `valid_shapefile_layer_names()`.

Examples:

```
>>> # To merge 'railways' layers of Greater Manchester and West Yorkshire"

>>> from pydriosm.reader._shp import SHP
>>> from pydriosm.downloader import BBBikeDownloader
>>> from pyhelpers.dirs import delete_dir
>>> import os

>>> # Download the .shp.zip file of Manchester and West Yorkshire
```

(continues on next page)

(continued from previous page)

```
>>> subrgn_names = ['London', 'Birmingham']
>>> file_fmt = ".shp"
>>> data_dir = "tests/osm_data"

>>> bbd = BBBikeDownloader()

>>> bbd.download_data(subrgn_names, file_fmt, data_dir, verbose=True)
Proceed to download data in the format '.shp.zip' for the following geographic (sub...
    "London"
    "Birmingham"
  to "./tests/osm_data/"
? [No]|Yes: yes
Downloading "London.osm.shp.zip" 100%|■■■■■■■■■■■| 248M/248M | 18.1MB/s | ETA:...
  Saving "London.osm.shp.zip" to "./tests/osm_data/london/" ... Done.
Downloading "Birmingham.osm.shp.zip" 100%|■■■■■■■■■■■| 79.1M/79.1M | 16.0MB/s ...
  Saving "Birmingham.osm.shp.zip" to "./tests/osm_data/birmingham/" ... Done.

>>> os.path.relpath(bbd.download_dir)
'tests\osm_data'
>>> len(bbd.data_paths)
2

>>> # Merge the layers of 'railways' of the two subregions
>>> merged_shp_path = SHP.merge_shp_layers(
...     bbd.data_paths, layer_name='railways', verbose=True, ret_shp_pathname=True)
Merging the following shapefiles:
  "london_railways.shp"
  "birmingham_railways.shp"
  In progress ... Done.
    Find the merged shapefile in "./tests/osm_data/lon-bir-railways/".

>>> # Check the pathname of the merged shapefile
>>> type(merged_shp_path)
list
>>> len(merged_shp_path)
1
>>> os.path.relpath(merged_shp_path[0])
'tests\osm_data\lon-bir-railways\lon-bir-railways.shp'

>>> # Read the merged .shp file
>>> merged_shp_data = SHP.read_shp(merged_shp_path[0])
>>> merged_shp_data.head()
   osm_id  ...                                          geometry
0   30804  ...     LINESTRING (0.00486 51.62793, 0.0062 51.62927)
1  101298  ...  LINESTRING (-0.22499 51.4937, -0.22516 51.4945...
2  101486  ...  LINESTRING (-0.20555 51.51954, -0.20514 51.519...
3  101511  ...  LINESTRING (-0.2119 51.52419, -0.21081 51.5239...
4  282898  ...   LINESTRING (-0.1862 51.61592, -0.18687 51.61386)
[5 rows x 4 columns]

>>> # Delete the test data directory
>>> delete_dir(bbd.download_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

See also

- Examples for the method *GeofabrikReader.merge_shp_layers()*.

SHP.merge_shps

classmethod SHP.**merge_shps**(*shp_pathnames*, *path_to_merged_dir*, *engine='geopandas'*, ***kwargs*)

Merge multiple shapefiles.

Parameters

- **shp_pathnames** (*list*) – list of paths to shapefiles (in .shp format)
- **path_to_merged_dir** (*str*) – path to a directory where the merged files are to be saved
- **engine** (*str*) – the open-source package that is used to merge/save shapefiles; options include: `'pyshp'` (default) and `'geopandas'` (or `'gpd'`) when `engine='geopandas'`, this function relies on geopandas.GeoDataFrame.to_file(); otherwise, it by default uses shapefile.Writer()

Note

- When `engine='geopandas'` (or `engine='gpd'`), the implementation of this function requires that GeoPandas is installed.

See also

- Examples for *merge_layers()*.
- Resource: https://github.com/GeospatialPython/pyshp

SHP.read_layer_shps

classmethod SHP.**read_layer_shps**(*shp_pathnames*, *feature_names=None*, *save_feat_shp=False*, *ret_feat_shp_path=False*, ***kwargs*)

Read a layer of OSM shapefile data.

Parameters

- **shp_pathnames** (*str | list*) – pathname of a .shp file, or pathnames of multiple shapefiles
- **feature_names** (*str | list | None*) – class name(s) of feature(s), defaults to `None`

- **save_feat_shp** (*bool*) – (when `fclass` is not `None`) whether to save data of the `fclass` as shapefile, defaults to `False`
- **ret_feat_shp_path** (*bool*) – (when `save_fclass_shp=True`) whether to return the path to the saved data of `fclass`, defaults to `False`
- **kwargs** – [optional] parameters of the method *SHP.read_shp()*

Returns

parsed shapefile data; and optionally, pathnames of the shapefiles of the specified features (when `ret_feat_shp_path=True`)

Return type

pandas.DataFrame | geopandas.GeoDataFrame | tuple

Examples:

```
>>> from pydriosm.reader._shp import SHP
>>> from pydriosm.downloader import BBBikeDownloader
>>> from pyhelpers.dirs import cd, delete_dir
>>> import os

>>> # Download the shapefile data of London as an example
>>> subrgn_name = 'Birmingham'
>>> file_format = ".shp"
>>> dwnld_dir = "tests/osm_data"

>>> bbd = BBBikeDownloader()

>>> bbd.download_data(subrgn_name, file_format, dwnld_dir, verbose=True)
Proceed to download data in the format '.shp.zip' for the following geographic (sub...
    "Birmingham"
  to "./tests/osm_data/birmingham/"
? [No]|Yes: >? yes
Downloading "Birmingham.osm.shp.zip" 100%|■■■■■■■■■■■| 79.1M/79.1M | 17.7MB/s ...
  Saving "Birmingham.osm.shp.zip" to "./tests/osm_data/birmingham/" ... Done.

>>> bham_shp_zip = bbd.data_paths[0]
>>> os.path.relpath(bham_shp_zip)
'tests\osm_data\birmingham\Birmingham.osm.shp.zip'

>>> # Extract the downloaded .shp.zip file
>>> bham_shp_dir = SHP.unzip_shp_zip(
...     bham_shp_zip, layer_names='railways', ret_extract_dir=True)
>>> os.listdir(cd(bham_shp_dir, "Birmingham-shp/shape"))
['railways.cpg',
 'railways.dbf',
 'railways.prj',
 'railways.shp',
 'railways.shx']
>>> bham_railways_shp_path = cd(bham_shp_dir, "Birmingham-shp/shape", "railways.shp")

>>> # Read the 'railways' layer
>>> bham_railways_shp = SHP.read_layer_shps(bham_railways_shp_path)
>>> bham_railways_shp.head()
    osm_id  ...                                           geometry
0      740  ...  LINESTRING (-1.81789 52.5701, -1.81793 52.5698...
1     2148  ...  LINESTRING (-1.87303 52.50542, -1.8727 52.5051...
```

(continues on next page)

(continued from previous page)

```
2  2950000  ...  LINESTRING (-1.87933 52.48138, -1.87962 52.481...
3  3491845  ...  LINESTRING (-1.7406 52.51858, -1.73942 52.5186...
4  3981454  ...  LINESTRING (-1.77412 52.52249, -1.77376 52.522...
[5 rows x 4 columns]

>>> # Extract only the features labelled 'rail' and save the extracted data to file
>>> railways_rail_shp, railways_rail_shp_path = SHP.read_layer_shps(
...     bham_railways_shp_path, feature_names='rail', save_feat_shp=True,
...     ret_feat_shp_path=True)
>>> railways_rail_shp['type'].unique()
<StringArray>
['rail']
Length: 1, dtype: str

>>> type(railways_rail_shp_path)
list
>>> len(railways_rail_shp_path)
1
>>> os.path.basename(railways_rail_shp_path[0])
'railways_rail.shp'

>>> # Delete the download/data directory
>>> delete_dir(dwnld_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

SHP.read_shp

classmethod SHP.**read_shp**(*shp_pathname*, *engine='geopandas'*, *emulate_gpd=False*, ***kwargs*)

Read a shapefile.

Parameters

- **shp_pathname** (*str*) – pathname of a shape format file (.shp)
- **engine** (*str*) – method used to read shapefiles; options include: `'pyshp'` and `'geopandas'` (default) (or `'gpd'`) this function by default relies on shapefile.reader(); when `engine='geopandas'` (or `engine='gpd'`), it relies on geopandas.read_file();
- **emulate_gpd** (*bool*) – whether to emulate the data format produced by geopandas.read_file() when `engine='pyshp'`.
- **kwargs** – [optional] parameters of the function geopandas.read_file() or shapefile.reader()

Returns

data frame of the shapefile data

Return type

pandas.DataFrame | geopandas.GeoDataFrame

Note

- **If `engine` is set to be `'geopandas'` (or `'gpd'`), it requires that** GeoPandas is installed.

Examples:

```
>>> from pydriosm.reader._shp import SHP
>>> from pydriosm.downloader import BBBikeDownloader
>>> from pyhelpers.dirs import cd, delete_dir
>>> import os
>>> import glob

>>> # Download the shapefile data of London as an example
>>> subregion_name = 'birmingham'
>>> osm_file_format = ".shp"
>>> download_dir = "tests/osm_data"

>>> bbd = BBBikeDownloader()

>>> bbd.download_data(subregion_name, osm_file_format, download_dir, verbose=True)
Proceed to download data in the format '.shp.zip' for the following geographic (sub...
    "Birmingham"
  to "./tests/osm_data/birmingham/"
? [No]|Yes: yes
Downloading "Birmingham.osm.shp.zip" 100%|■■■■■■■■■■■| 79.1M/79.1M | 17.4MB/s ...
  Saving "Birmingham.osm.shp.zip" to "./tests/osm_data/birmingham/" ... Done.

>>> bham_shp_zip = bbd.data_paths[0]
>>> os.path.relpath(bham_shp_zip)
'tests\osm_data\birmingham\Birmingham.osm.shp.zip'

>>> # Extract all
>>> bham_shp_dir = SHP.unzip_shp_zip(bham_shp_zip, ret_extract_dir=True)

>>> # Get the pathname of the .shp data of 'railways'
>>> path_to_railways_shp = glob.glob(
...     cd(bham_shp_dir, "Birmingham-shp", "shape", "*railways*.shp"))[0]
>>> os.path.relpath(path_to_railways_shp)  # Check the pathname of the .shp file
'tests\osm_data\birmingham\Birmingham-osm-shp\Birmingham-shp\shape\railways.shp'

>>> # Read the data of 'railways'
>>> bham_railways = SHP.read_shp(path_to_railways_shp)
>>> bham_railways.head()
    osm_id  ...                                           geometry
0      740  ...  LINESTRING (-1.81789 52.5701, -1.81793 52.5698...
1     2148  ...  LINESTRING (-1.87303 52.50542, -1.8727 52.5051...
2  2950000  ...  LINESTRING (-1.87933 52.48138, -1.87962 52.481...
3  3491845  ...  LINESTRING (-1.7406 52.51858, -1.73942 52.5186...
4  3981454  ...  LINESTRING (-1.77412 52.52249, -1.77376 52.522...
[5 rows x 4 columns]

>>> # Set `emulate_gpd=True` to return data of similar format to what GeoPandas does
>>> bham_railways_ = SHP.read_shp(
...     path_to_railways_shp, engine='pyshp', emulate_gpd=True)
```

(continues on next page)

(continued from previous page)

```
>>> bham_railways_.head()
    osm_id  ...                                           geometry
0      740  ...  LINESTRING (-1.8178905 52.5700974, -1.8179287 ...
1     2148  ...  LINESTRING (-1.873028 52.5054182, -1.8726964 5...
2  2950000  ...  LINESTRING (-1.8793303 52.4813778, -1.8796237 ...
3  3491845  ...  LINESTRING (-1.7406017 52.5185831, -1.7394216 ...
4  3981454  ...  LINESTRING (-1.7741212 52.5224935, -1.7737563 ...
[5 rows x 4 columns]

>>> # Check the data types of `london_railways` and `london_railways_`
>>> railways_data = [bham_railways, bham_railways_]
>>> list(map(type, railways_data))
[geopandas.geodataframe.GeoDataFrame, pandas.DataFrame]
>>> # Check the geometry data of `london_railways` and `london_railways_`
>>> geom1, geom2 = map(lambda x: x['geometry'].map(lambda y: y.wkb), railways_data)
>>> geom1.equals(geom2)
True

>>> # Delete the download/data directory
>>> delete_dir(bbd.download_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

SHP.unzip_shp_zip

classmethod SHP.**unzip_shp_zip**(*shp_zip_pathname*, *extract_to=None*, *layer_names=None*, *separate=False*, *ret_extract_dir=False*, *verbose=False*, *raise_error=False*)

Unzip a zipped shapefile.

Parameters

- **shp_zip_pathname** (*str | os.PathLike[str]*) – path to a zipped shapefile data (.shp.zip)
- **extract_to** (*str | None*) – path to a directory where extracted files will be saved; when `extract_to=None` (default), the same directory where the .shp.zip file is saved
- **layer_names** (*str | list | None*) – name of a .shp layer, e.g. 'railways', or names of multiple layers; when `layer_names=None` (default), all available layers
- **separate** (*bool*) – whether to put the data files of different layer in respective folders, defaults to `False`
- **ret_extract_dir** (*bool*) – whether to return the pathname of the directory where extracted files are saved, defaults to `False`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

the path to the directory of extracted files when `ret_extract_dir=True`

Return type

str

Examples:

```
>>> from pydriosm.reader._shp import SHP
>>> from pydriosm.downloader import BBBikeDownloader
>>> from pyhelpers.dirs import cd, delete_dir
>>> import os

>>> # Download the shapefile data of London as an example
>>> subrgn_name = 'birmingham'
>>> file_format = ".shp"
>>> dwnld_dir = "tests/osm_data"

>>> bbd = BBBikeDownloader()

>>> bbd.download_data(subrgn_name, file_format, dwnld_dir, verbose=True)
Proceed to update the data in the format '.shp.zip' for the following geographic (s...
    "Birmingham"
  in "./tests/osm_data/birmingham/"
? [No]|Yes: yes
Downloading "Birmingham.osm.shp.zip" 100%|■■■■■■■■■■■| 79.1M/79.1M | 17.7MB/s ...
  Updating "Birmingham.osm.shp.zip" in "./tests/osm_data/birmingham/" ... Done.

>>> path_to_shp_zip = bbd.data_paths[0]
>>> os.path.relpath(path_to_shp_zip)
'tests\osm_data\birmingham\Birmingham.osm.shp.zip'

>>> # To extract data of a specific layer 'railways'
>>> bham_railways_dir = SHP.unzip_shp_zip(
...     path_to_shp_zip, layer_names='railways', verbose=True, ret_extract_dir=True)
Extracting the following layer(s):
    'railways'
    from "tests\osm_data\greater-london\greater-london-latest-free.shp.zip"
      to "tests\osm_data\greater-london\greater-london-latest-free-shp\" ... Done.

>>> os.path.relpath(bham_railways_dir)  # Check the directory
'tests\osm_data\birmingham\Birmingham-osm-shp'

>>> # When multiple layer names are specified, the extracted files for each of the
>>> # layers can be put into a separate subdirectory by setting `separate=True`:
>>> lyr_names = ['railways', 'landuse']
>>> dirs_of_layers = SHP.unzip_shp_zip(
...     shp_zip_pathname=path_to_shp_zip, layer_names=lyr_names, separate=True,
...     verbose=2, ret_extract_dir=True)
Extracting the following layer(s):
  'railways'
  'landuse'
  from: "./tests/osm_data/birmingham/Birmingham.osm.shp.zip" ...
    to: "./tests/osm_data/birmingham/Birmingham-osm-shp/" ... Done.
Grouping files by layers ...
  landuse ... Done.
  railways ... Done.
Done.
```

(continues on next page)

(continued from previous page)

```
>>> len(dirs_of_layers) == 2
True
>>> os.path.relpath(os.path.commonpath(dirs_of_layers))
'tests\osm_data\birmingham\Birmingham-osm-shp'
>>> set(map(os.path.basename, dirs_of_layers))
{'landuse', 'railways'}

>>> # Remove the subdirectories
>>> delete_dir(dirs_of_layers, confirmation_required=False)

>>> # To extract all (without specifying `layer_names`
>>> bham_shp_dir = SHP.unzip_shp_zip(
...     path_to_shp_zip, verbose=True, ret_extract_dir=True)
Extracting "./tests/osm_data/birmingham/Birmingham.osm.shp.zip"
  to "./tests/osm_data/birmingham/Birmingham-osm-shp/" ... Done.

>>> # Check the directory
>>> os.path.relpath(bham_shp_dir)
'tests\osm_data\birmingham\Birmingham-osm-shp'
>>> list_of_files = os.listdir(cd(bham_shp_dir, "Birmingham-shp/shape"))
>>> len(list_of_files)
40
>>> # Get the names of all available layers
>>> set(filter(None, map(SHP.get_layer_name, list_of_files)))
{'buildings',
 'landuse',
 'natural',
 'places',
 'points',
 'railways',
 'roads',
 'waterways'}

>>> # Delete the download/data directory
>>> delete_dir(bbd.download_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

SHP.validate_layer_names

classmethod SHP.**validate_layer_names**(*layer_names*)

Validate the input of layer name(s) for reading shapefiles.

Parameters

layer_names (*str | list | None*) – name of a shapefile layer, e.g. 'railways', or names of multiple layers; if `None` (default), returns an empty list; if `layer_names='all'`, the function returns a list of all available layers

Returns

valid layer names to be input

Return type

list

Examples:

```
>>> from pydriosm.reader._shp import SHP

>>> SHP.validate_layer_names(None)
[]

>>> SHP.validate_layer_names('point')
['points']

>>> SHP.validate_layer_names(['point', 'land'])
['points', 'landuse']

>>> SHP.validate_layer_names('all')
['buildings',
 'landuse',
 'natural',
 'places',
 'pofw',
 'points',
 'pois',
 'railways',
 'roads',
 'traffic',
 'transport',
 'water',
 'waterways']
```

SHP.write_to_shapefile

classmethod SHP.**write_to_shapefile**(*data*, *write_to*, *shp_filename=None*, *decimal_precision=5*, *ret_shp_pathname=False*, *verbose=False*, *raise_error=False*)

Save .shp data as a shapefile by PyShp.

Parameters

- **data** (*pandas.DataFrame*) – data of a shapefile
- **write_to** (*str*) – pathname of a directory where the shapefile data is to be saved
- **shp_filename** (*str | os.PahtLike[str] | None*) – filename (or pathname) of the target .shp file, defaults to `None`; when `shp_filename=None`, it is by default the basename of `write_to`
- **decimal_precision** (*int*) – decimal precision for writing float records, defaults to `5`
- **ret_shp_pathname** (*bool*) – whether to return the pathname of the output .shp file, defaults to `False`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Examples:

```
>>> from pydriosm.reader._shp import SHP
>>> from pydriosm.downloader import BBBikeDownloader
>>> from pyhelpers.dirs import cd, delete_dir
>>> import os
>>> import glob

>>> # Download the shapefile data of London as an example
>>> subrgn_name = 'Birmingham'
>>> file_format = ".shp"
>>> dwnld_dir = "tests/osm_data"

>>> bbd = BBBikeDownloader()

>>> bbd.download_data(subrgn_name, file_format, dwnld_dir, verbose=True)
Proceed to download data in the format '.shp.zip' for the following geographic (sub...
    "Birmingham"
  to "./tests/osm_data/birmingham/"
? [No]|Yes: yes
Downloading "Birmingham.osm.shp.zip" 100%|■■■■■■■■■■| 79.1M/79.1M | 18.1MB/s ...
  Saving "Birmingham.osm.shp.zip" to "./tests/osm_data/birmingham/" ... Done.

>>> bham_shp_zip = bbd.data_paths[0]
>>> os.path.relpath(bham_shp_zip)
'tests\osm_data\birmingham\Birmingham.osm.shp.zip'

>>> # Extract the 'railways' layer of the downloaded .shp.zip file
>>> lyr_name = 'railways'

>>> railways_shp_dir = SHP.unzip_shp_zip(
...     bham_shp_zip, layer_names=lyr_name, verbose=True, ret_extract_dir=True)
Extracting the following layer(s):
  'railways'
  from: "./tests/osm_data/birmingham/Birmingham.osm.shp.zip" ...
    to: "./tests/osm_data/birmingham/Birmingham-osm-shp/" ... Done.
>>> # Check out the output directory
>>> os.path.relpath(railways_shp_dir)
'tests\osm_data\birmingham\Birmingham-osm-shp'

>>> # Get the pathname of the .shp data of 'railways'
>>> path_to_railways_shp = glob.glob(
...     cd(railways_shp_dir, "Birmingham-shp", "shape", f"*{lyr_name}*.shp"))[0]
>>> os.path.relpath(path_to_railways_shp)  # Check the pathname of the .shp file
'tests\osm_data\birmingham\Birmingham-osm-shp\Birmingham-shp\shape\railways.shp'

>>> # Read the .shp file
>>> bham_railways_shp = SHP.read_shp(path_to_railways_shp)

>>> # Create a new directory for saving the 'railways' data
>>> railways_subdir = cd(os.path.dirname(railways_shp_dir), lyr_name)
>>> os.path.relpath(railways_subdir)
'tests\osm_data\birmingham\railways'

>>> # Save the data of 'railways' to the new directory
>>> path_to_railways_shp_ = SHP.write_to_shapefile(
...     bham_railways_shp, railways_subdir, ret_shp_pathname=True, verbose=True)
Writing data to "tests/osm_data/birmingham/railways.*" ... Done.
```

(continues on next page)

(continued from previous page)

```
>>> os.path.basename(path_to_railways_shp_)
'railways.shp'

>>> # If `shp_filename` is specified
>>> path_to_railways_shp_ = SHP.write_to_shapefile(
...     bham_railways_shp, railways_subdir, shp_filename="rail_data",
...     ret_shp_pathname=True, verbose=True)
Writing data to "tests/osm_data/birmingham/rail_data.*" ... Done.
>>> os.path.basename(path_to_railways_shp_)
'rail_data.shp'

>>> # Retrieve the saved the .shp file
>>> bham_railways_shp_ = SHP.read_shp(path_to_railways_shp_)
>>> # Check if the retrieved .shp data is equal to the original one
>>> bham_railways_shp_.equals(bham_railways_shp)
True

>>> # Delete the download/data directory
>>> delete_dir(bbd.download_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

PBF

class pydriosm.reader._pbf.**PBF**

Read/parse PBF data.

Examples:

```
>>> from pydriosm.reader._pbf import PBF

>>> PBF.LAYER_GEOM
{'points': shapely.geometry.point.Point,
 'lines': shapely.geometry.linestring.LineString,
 'multilinestrings': shapely.geometry.multilinestring.MultiLineString,
 'multipolygons': shapely.geometry.multipolygon.MultiPolygon,
 'other_relations': shapely.geometry.collection.GeometryCollection}
```

Attributes

1anti-flashwhitewhite

LAYER_GEOM	Layer names of an OSM PBF file and their corresponding geometric objects defined in Shapely.

PBF.LAYER_GEOM

PBF.**LAYER_GEOM: dict = {'lines': <class 'shapely.geometry.linestring.LineString'>, 'multilinestrings': <class 'shapely.geometry.multilinestring.MultiLineString'>, 'multipolygons': <class 'shapely.geometry.multipolygon.MultiPolygon'>, 'other_relations': <class 'shapely.geometry.collection.GeometryCollection'>, 'points': <class 'shapely.geometry.point.Point'>}**

Layer names of an OSM PBF file and their corresponding geometric objects defined in Shapely.

Methods

1anti-flashwhitewhite

get_layer_geom_types([shape_name])	A dictionary cross-referencing the names of PBF layers and their corresponding geometric objects defined in Shapely, or names.
get_layer_names(path_to_file[, verbose, ...])	Get names (and indices) of all available layers in a PBF data file.
read_layer(layer[, readable, expand, ...])	Parse a layer of a PBF data file.
read_pbf(path_to_file[, readable, expand, ...])	Parse a PBF data file (by GDAL).
transform_pbf_layer_field(layer_data, layer_name)	Parse data of a layer of PBF data.

PBF.get_layer_geom_types

classmethod PBF.**get_layer_geom_types**(*shape_name=False*)

A dictionary cross-referencing the names of PBF layers and their corresponding geometric objects defined in Shapely, or names.

Parameters

shape_name (*bool*) – whether to return the names of geometry shapes, defaults to `False`

Returns

a dictionary with keys and values being, respectively, PBF layers and their corresponding geometric objects defined in Shapely

Return type

dict

Examples:

```
>>> from pydriosm.reader._pbf import PBF

>>> PBF.get_layer_geom_types()
{'points': shapely.geometry.point.Point,
```

(continues on next page)

(continued from previous page)

```
 'lines': shapely.geometry.linestring.LineString,
 'multilinestrings': shapely.geometry.multilinestring.MultiLineString,
 'multipolygons': shapely.geometry.multipolygon.MultiPolygon,
 'other_relations': shapely.geometry.collection.GeometryCollection}

>>> PBF.get_layer_geom_types(shape_name=True)
{'points': 'Point',
 'lines': 'LineString',
 'multilinestrings': 'MultiLineString',
 'multipolygons': 'MultiPolygon',
 'other_relations': 'GeometryCollection'}
```

PBF.get_layer_names

classmethod PBF.**get_layer_names**(*path_to_file*, *verbose=False*, *raise_error=False*)

Get names (and indices) of all available layers in a PBF data file.

Parameters

- **path_to_file** (*str | os.PathLike[str]*) – path to a PBF data file
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

indices and names of each layer of the PBF data file

Return type

dict

Examples:

```
>>> from pydriosm.reader._pbf import PBF
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pyhelpers.dirs import delete_dir
>>> import os

>>> # Download the PBF data file of London as an example
>>> subrgn_name = 'london'
>>> file_format = ".pbf"
>>> dwnld_dir = "tests/osm_data"

>>> gfd = GeofabrikDownloader()

>>> gfd.download_data(subrgn_name, file_format, dwnld_dir, verbose=True)
To download .osm.pbf data of the following geographic (sub)region(s):
    Greater London
? [No]|Yes: yes
Downloading "greater-london-latest.osm.pbf"
    to "tests\osm_data\greater-london\" ... Done.

>>> london_pbf_pathname = gfd.data_paths[0]
```

(continues on next page)

(continued from previous page)

```
>>> os.path.relpath(london_pbf_pathname)
'tests\osm_data\greater-london\greater-london-latest.osm.pbf'

>>> # Get indices and names of all layers in the downloaded PBF data file
>>> pbf_layer_idx_names = PBF.get_layer_names(london_pbf_pathname)
>>> type(pbf_layer_idx_names)
dict
>>> pbf_layer_idx_names
{0: 'points',
 1: 'lines',
 2: 'multilinestrings',
 3: 'multipolygons',
 4: 'other_relations'}

>>> # Delete the download directory (and the downloaded PBF data file)
>>> delete_dir(gfd.download_dir, verbose=True)
To delete the directory "tests\osm_data\" (Not empty)
? [No]|Yes: yes
Deleting "tests\osm_data\" ... Done.
```

PBF.read_layer

classmethod PBF.**read_layer**(*layer*, *readable=True*, *expand=False*, *parse_geometry=False*, *parse_properties=False*, *parse_other_tags=False*, *number_of_chunks=None*)

Parse a layer of a PBF data file.

Parameters

- **layer** (*osgeo.ogr.Layer*) – a layer of a PBF data file, loaded by GDAL/OGR
- **readable** (*bool*) – whether to parse each feature in the raw data, defaults to `False`
- **expand** (*bool*) – whether to expand dict-like data into separate columns, defaults to `False`
- **parse_geometry** (*bool*) – whether to represent the `'geometry'` field in a shapely.geometry format, defaults to `False`
- **parse_properties** (*bool*) – whether to represent the `'properties'` field in a tabular format, defaults to `False`
- **parse_other_tags** (*bool*) – whether to represent a `'other_tags'` (of `'properties'`) in a dict format, defaults to `False`
- **number_of_chunks** (*int | None*) – number of chunks, defaults to `None`

Returns

parsed data of the given OSM PBF layer

Return type

dict

See also

- Examples for *PBF.read_pbf()*.

PBF.read_pbf

classmethod PBF.**read_pbf**(*path_to_file*, *readable=True*, *expand=False*, *parse_geometry=False*, *parse_properties=False*, *parse_other_tags=False*, *number_of_chunks=None*, *max_tmpfile_size=5000*, ***kwargs*)

Parse a PBF data file (by GDAL).

Parameters

- **path_to_file** (*str*) – pathname of a PBF data file
- **readable** (*bool*) – whether to parse each feature in the raw data, defaults to `False`
- **expand** (*bool*) – whether to expand dict-like data into separate columns, defaults to `False`
- **parse_geometry** (*bool*) – whether to represent the `'geometry'` field in a shapely.geometry format, defaults to `False`
- **parse_properties** (*bool*) – whether to represent the `'properties'` field in a tabular format, defaults to `False`
- **parse_other_tags** (*bool*) – whether to represent a `'other_tags'` (of `'properties'`) in a dict format, defaults to `False`
- **number_of_chunks** (*int* / *None*) – number of chunks, defaults to `None`
- **max_tmpfile_size** (*int* / *None*) – maximum size of the temporary file, defaults to `None`; when `max_tmpfile_size=None`, it defaults to `5000`
- **kwargs** – [optional] parameters of the function pyhelpers.settings.gdal_configurations()

Returns

parsed OSM PBF data

Return type

dict

Note

The GDAL/OGR drivers categorizes the features of OSM PBF data into five layers:

- **0: 'points'** - "node" features having significant tags attached
- **1: 'lines'** - "way" features being recognized as non-area
- **2: 'multilinestrings'** - "relation" features forming a multilinestring (type='multilinestring' / type='route')

- **3: 'multipolygons'** - "relation" features forming a multipolygon (type='multipolygon' / type='boundary'), and "way" features being recognized as area
- **4: 'other_relations'** - "relation" features not belonging to the above 2 layers

For more information, please refer to OpenStreetMap XML and PBF.

Warning

- **Parsing large PBF data files (e.g. > 50MB) can be time-consuming!**
- The function `read_osm_pbf()` may require fairly high amount of physical memory to parse large files, in which case it would be recommended that `number_of_chunks` is set to be a reasonable value.

Examples:

```
>>> from pydriosm.reader._pbf import PBF
>>> from pydriosm.downloader import Downloader
>>> from pyhelpers.dirs import delete_dir
>>> import os

>>> # Download the PBF data file of 'Rutland' as an example
>>> subregion_name = 'rutland'
>>> osm_file_format = ".pbf"
>>> download_dir = "tests/osm_data"

>>> dl = Downloader()

>>> dl.download_data(subregion_name, osm_file_format, download_dir, verbose=True)
To download data in the format '.osm.pbf' for the following geographic (sub)region(s):
    "Rutland"
  to "./tests/osm_data/rutland/"
? [No]|Yes: >? yes
Downloading "rutland-latest.osm.pbf" 100%|■■■■■■■■■■■■| 1.83M/1.83M | 5.74MB/s ...
    Saving "rutland-latest.osm.pbf" to "./tests/osm_data/rutland/" ... Done.

>>> path_to_file = dl.data_paths[0]
>>> os.path.relpath(path_to_file)
'tests\osm_data\rutland\rutland-latest.osm.pbf'

>>> # Read the downloaded PBF data
>>> rutland_pbf = PBF.read_pbf(path_to_file)
>>> type(rutland_pbf)
dict
>>> list(rutland_pbf.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> rutland_pbf_points = rutland_pbf['points']
>>> rutland_pbf_points.head()
0    {'type': 'Feature', 'geometry': {'type': 'Poin...
1    {'type': 'Feature', 'geometry': {'type': 'Poin...
```

(continues on next page)

(continued from previous page)

```
2    {'type': 'Feature', 'geometry': {'type': 'Poin...
3    {'type': 'Feature', 'geometry': {'type': 'Poin...
4    {'type': 'Feature', 'geometry': {'type': 'Poin...
Name: points, dtype: object

>>> # Set `expand` to be `True`
>>> pbf_0 = PBF.read_pbf(path_to_file, expand=True)
>>> type(pbf_0)
dict
>>> list(pbf_0.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']
>>> pbf_0_points = pbf_0['points']
>>> pbf_0_points.head()
         id  ...                                         properties
0    488432  ...  {'osm_id': '488432', 'name': None, 'barrier': ...
1    488658  ...  {'osm_id': '488658', 'name': 'Tickencote Inter...
2  13883868  ...  {'osm_id': '13883868', 'name': None, 'barrier'...
3  14049101  ...  {'osm_id': '14049101', 'name': None, 'barrier'...
4  14558402  ...  {'osm_id': '14558402', 'name': None, 'barrier'...
[5 rows x 3 columns]

>>> pbf_0_points['geometry'].head()
0    {'type': 'Point', 'coordinates': [-0.5134241, ...
1    {'type': 'Point', 'coordinates': [-0.5313354, ...
2    {'type': 'Point', 'coordinates': [-0.7229332, ...
3    {'type': 'Point', 'coordinates': [-0.7249816, ...
4    {'type': 'Point', 'coordinates': [-0.7266581, ...
Name: geometry, dtype: object

>>> # Set both `expand` and `parse_geometry` to be `True`
>>> pbf_1 = PBF.read_pbf(path_to_file, expand=True, parse_geometry=True)
>>> pbf_1_points = pbf_1['points']
>>> # Check the difference in 'geometry' column, compared to `pbf_0_points`
>>> pbf_1_points['geometry'].head()
0    POINT (-0.5134241 52.6555853)
1    POINT (-0.5313354 52.6737716)
2    POINT (-0.7229332 52.5889864)
3    POINT (-0.7249816 52.6748426)
4     POINT (-0.7266543 52.669517)
Name: geometry, dtype: object

>>> # Set both `expand` and `parse_properties` to be `True`
>>> pbf_2 = PBF.read_pbf(path_to_file, expand=True, parse_properties=True)
>>> pbf_2_points = pbf_2['points']
>>> pbf_2_points['other_tags'].head()
0                "odbl"=>"clean"
1                           None
2                           None
3    "traffic_calming"=>"cushion"
4        "direction"=>"clockwise"
Name: other_tags, dtype: object

>>> # Set both `expand` and `parse_other_tags` to be `True`
>>> pbf_3 = PBF.read_pbf(path_to_file, expand=True, parse_properties=True,
...     parse_other_tags=True)
>>> pbf_3_points = pbf_3['points']
>>> # Check the difference in 'other_tags', compared to ``pbf_2_points``
```

(continues on next page)

(continued from previous page)

```
>>> pbf_3_points['other_tags'].head()
0                {'odbl': 'clean'}
1                             None
2                             None
3    {'traffic_calming': 'cushion'}
4        {'direction': 'clockwise'}
Name: other_tags, dtype: object

>>> # Delete the downloaded PBF data file
>>> delete_dir(dl.download_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

See also

- Examples for the methods: *GeofabrikReader.read_pbf()* and *BBBikeReader.read_pbf()*.

PBF.transform_pbf_layer_field

classmethod PBF.**transform_pbf_layer_field**(*layer_data*, *layer_name*, *parse_geometry=False*, *parse_properties=False*, *parse_other_tags=False*)

Parse data of a layer of PBF data.

Parameters

- **layer_data** (*pandas.DataFrame | pandas.Series*) – dataframe of a specific layer of PBF data
- **layer_name** (*str*) – name (geometric type) of the PBF layer
- **parse_geometry** (*bool*) – whether to represent the `'geometry'` field in a shapely.geometry format, defaults to `False`
- **parse_properties** (*bool*) – whether to represent the `'properties'` field in a tabular format, defaults to `False`
- **parse_other_tags** (*bool*) – whether to represent a `'other_tags'` (of `'properties'`) in a dict format, defaults to `False`

Returns

readable data of the given PBF layer

Return type

pandas.DataFrame | pandas.Series

See examples for *PBF.read_pbf()*.

VAR

class pydriosm.reader._var.**VAR**

Read/parse OSM data of various formats (other than PBF and Shapefile).

Methods

1anti-flashwhitewhite

read_csv_xz(path_to_file[, col_names])	Read/parse a compressed CSV (.csv.xz) data file.
read_geojson_xz(path_to_file[, engine, ...])	Read/parse a compressed Osmium GeoJSON (.geojson.xz) data file.

VAR.read_csv_xz

classmethod VAR.**read_csv_xz**(*path_to_file*, *col_names=None*)

Read/parse a compressed CSV (.csv.xz) data file.

Parameters

- **path_to_file** (*str*) – path to a .csv.xz data file
- **col_names** (*list | None*) – column names of .csv.xz data, defaults to None

Returns

tabular data of the CSV file

Return type

pandas.DataFrame

See examples for the method *BBBikeReader.read_csv_xz()*.

VAR.read_geojson_xz

classmethod VAR.**read_geojson_xz**(*path_to_file*, *engine=None*, *parse_geometry=False*)

Read/parse a compressed Osmium GeoJSON (.geojson.xz) data file.

Parameters

- **path_to_file** (*str*) – path to a .geojson.xz data file
- **engine** (*str | None*) – an open-source Python package for JSON serialization, defaults to None; when engine=None, it refers to the built-in json module; otherwise options include: 'ujson' (for UltraJSON), 'orjson' (for orjson) and 'rapidjson' (for python-rapidjson)
- **parse_geometry** (*bool*) – whether to reformat coordinates into a geometric object, defaults to False

Returns

tabular data of the Osmium GeoJSON file

Return type
pandas.DataFrame

See also

- Examples for the method *BBBikeReader.read_geojson_xz()*.

4.2.2 Read OSM data

GeofabrikReader([data_dir, max_tmpfile_size])	Read Geofabrik OpenStreetMap data extracts.
BBBikeReader([data_dir, max_tmpfile_size])	Read BBBike exports of OpenStreetMap data.

GeofabrikReader

class pydriosm.reader.**GeofabrikReader**(*data_dir=None*, *max_tmpfile_size=None*)

Read Geofabrik OpenStreetMap data extracts.

Parameters

- **max_tmpfile_size** (*int* | *None*) – defaults to None, see also the function pyhelpers.settings.gdal_configurations()
- **data_dir** (*str* | *None*) – (a path or a name of) a directory where a data file is, defaults to None; when data_dir=None, it refers to a folder named osm_geofabrik under the current working directory

Variables

- **downloader** (GeofabrikDownloader) – instance of the class *GeofabrikDownloader*
- **name** (*str*) – name of the data resource
- **url** (*str*) – url of the homepage to the Geofabrik free download server

Examples:

```
>>> from pydriosm.reader import GeofabrikReader
>>> gfr = GeofabrikReader()
>>> gfr.NAME
'Geofabrik'
```

Attributes

DEFAULT_DATA_DIR	Default download directory.

continues on next page

Table 14 – continued from previous page

FILE_FORMATS	Valid file formats.
LONG_NAME	Full name of the data source.
NAME	Name of the data source.
data_dir	Name or pathname of a data directory.
data_paths	Pathnames of all data files.

GeofabrikReader.DEFAULT_DATA_DIR

GeofabrikReader.**DEFAULT_DATA_DIR: str = 'osm_data/geofabrik'**

Default download directory.

Type
str

GeofabrikReader.FILE_FORMATS

GeofabrikReader.**FILE_FORMATS: set = {'.gpkg.zip', '.osm.bz2', '.osm.pbf', '.shp.zip'}**

Valid file formats.

Type
set

GeofabrikReader.LONG_NAME

GeofabrikReader.**LONG_NAME: str = 'Geofabrik OpenStreetMap data extracts'**

Full name of the data source.

GeofabrikReader.NAME

GeofabrikReader.**NAME: str = 'Geofabrik'**

Name of the data source.

GeofabrikReader.data_dir

property GeofabrikReader.**data_dir**

Name or pathname of a data directory.

Returns
name or pathname of a directory for saving downloaded data files

Return type
str | None

Tests:

```
>>> from pydriosm.reader._base import BaseReader
>>> from pydriosm.downloader import GeofabrikDownloader, BBBikeDownloader
```

(continues on next page)

(continued from previous page)

```
>>> import os
>>> brd = BaseReader()
>>> os.path.relpath(brd.data_dir)
'osm_data'
>>> brd = BaseReader(downloader=GeofabrikDownloader)
>>> os.path.relpath(brd.data_dir)
'osm_data'
>>> brd = BaseReader(downloader=BBBikeDownloader)
>>> os.path.relpath(brd.data_dir)
'osm_data'
```

GeofabrikReader.data_paths

property GeofabrikReader.**data_paths**

Pathnames of all data files.

Returns
pathnames of all data files

Return type
list

Tests:

```
>>> from pydriosm.reader._base import BaseReader
>>> brd = BaseReader()
>>> brd.data_paths
[]
```

Methods

1anti-flashwhitewhite

cdd(*sub_dir[, mkdir])	Change directory to default data directory and its subdirectories or a specific file.
get_file_path(subregion_name, osm_file_format)	Get the local path to an OSM data file of a geographic (sub)region.
get_pbf_layer_names(subregion_name[, data_dir])	Get indices and names of all layers in the PBF data file of a given (sub)region.
get_shp_pathname(subregion_name[, ...])	Get path(s) to .shp file(s) for a geographic (sub)region (by searching a local data directory).
make_shp_pkl_pathname(shp_zip_filename, ...)	Make a pathname of a pickle file for saving shapefile data.
merge_shp_layers(subregion_names, layer_name)	Merge shapefiles for a specific layer of two or multiple geographic regions.
read_gpkg(subregion_name[, layer_names, ...])	Reads GeoPackage (.gpkg.zip) data for a specific subregion.

continues on next page

Table 15 – continued from previous page

read_pbf(subregion_name[, data_dir, ...])	Read a PBF (.osm.pbf) data file of a geographic (sub)region.
read_shp(subregion_name[, layer_names, ...])	Read a .shp.zip data file of a geographic (sub)region.
read_var_osm(meth, subregion_name, ...[, ...])	Read data file of various formats (other than PBF and shapefile) for a geographic (sub)region.
remove_extracts(path_to_extract_dir, verbose)	Remove data extracts.
validate_dtype(x)	Validate the data type of the input variable.
validate_file_path(path_to_file[, ...])	Validate the pathname of an OSM data file.
validate_shp_layer_names(layer_names_, ...)	Validate the input of layer name(s) for reading shapefiles.

GeofabrikReader.cdd

classmethod GeofabrikReader.**cdd**(**sub_dir*, *mkdir=False*, ***kwargs*)

Change directory to default data directory and its subdirectories or a specific file.

Parameters

- **sub_dir** (*str | os.PathLike[str]*) – name of directory; names of directories (and/or a filename)
- **mkdir** (*bool*) – whether to create a directory, defaults to False
- **kwargs** – [optional] parameters of the function pyhelpers.dir.cd()

Returns

an absolute pathname to a directory (or a file)

Return type

str | os.PathLike[str]

Tests:

```
>>> from pydriosm.reader._base import BaseReader
>>> import os
>>> os.path.relpath(BaseReader.cdd())
'osm_data'
>>> os.path.exists(BaseReader.cdd())
False
```

GeofabrikReader.get_file_path

GeofabrikReader.**get_file_path**(*subregion_name*, *osm_file_format*, *data_dir=None*)

Get the local path to an OSM data file of a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on Geofabrik free download server

- **osm_file_format** (*str*) – file format of the OSM data available on the free download server
- **data_dir** (*str | None*) – directory where the data file of the `subregion_name` is located/saved; if `None` (default), the default local directory

Returns
path to PBF (.osm.pbf) file

Return type
str | None

Examples:

```
>>> from pydriosm.reader import GeofabrikReader
>>> from pyhelpers.dirs import delete_dir
>>> import os

>>> gfr = GeofabrikReader()

>>> subrgn_name = 'rutland'
>>> file_format = ".pbf"
>>> dat_dir = "tests\osm_data"
>>> path_to_rutland_pbf = gfr.get_file_path(subrgn_name, file_format, data_dir=dat_dir)
>>> # When "rutland-latest.osm.pbf" is unavailable at the package data directory
>>> os.path.isfile(path_to_rutland_pbf)
False

>>> # Download the PBF data file of Rutland to "tests\osm_data\"
>>> gfr.downloader.download_data(subrgn_name, file_format, dat_dir, verbose=True)
To download data in the format '.osm.pbf' for the following geographic (sub)region(s):
  "Rutland"
  to "./tests/osm_data/rutland/"
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" 100%|■■■■■■■■■■| 1.89M/1.89M | 730kB/s |...
  Saving "rutland-latest.osm.pbf" to "./tests/osm_data/rutland/" ... Done.

>>> # Check again
>>> path_to_rutland_pbf = gfr.get_file_path(subrgn_name, file_format, data_dir=dat_dir)
>>> os.path.relpath(path_to_rutland_pbf)
'tests\osm_data\rutland\rutland-latest.osm.pbf'
>>> os.path.isfile(path_to_rutland_pbf)
True

>>> # Delete the test data directory
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

GeofabrikReader.get_pbf_layer_names

`GeofabrikReader.`**`get_pbf_layer_names`**(*subregion_name*, *data_dir=None*)

Get indices and names of all layers in the PBF data file of a given (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on Geofabrik free download server
- **data_dir**

Returns
indices and names of each layer of the PBF data file

Return type
dict

Examples:

```
>>> from pydriosm.reader import GeofabrikReader
>>> from pyhelpers.dirs import delete_dir
>>> import os
>>> gfr = GeofabrikReader()
>>> # Download the PBF data of Rutland as an example
>>> subregion_name = 'rutland'
>>> osm_file_format = ".pbf"
>>> data_dir = "tests/osm_data"
>>> # Download the PBF data of Rutland to "./tests/osm_data/"
>>> gfr.downloader.download_data(
...     subregion_name, osm_file_format, data_dir, verbose=True)
To download data in the format '.osm.pbf' for the following geographic (sub)region(s):
    "Rutland"
  to "./tests/osm_data/rutland/"
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" 100%|■■■■■■■■■■■| 1.83M/1.83M | 1.00MB/s ...
    Saving "rutland-latest.osm.pbf" to "./tests/osm_data/rutland/" ... Done.
>>> path_to_pbf = gfr.data_paths[0]
>>> os.path.relpath(path_to_pbf)
'tests\osm_data\rutland\rutland-latest.osm.pbf'
>>> lyr_idx_names = gfr.get_pbf_layer_names(subregion_name)
>>> lyr_idx_names
{0: 'points',
 1: 'lines',
 2: 'multilinestrings',
 3: 'multipolygons',
 4: 'other_relations'}
>>> # Delete the example data and the test data directory
>>> delete_dir(data_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

GeofabrikReader.get_shp_pathname

GeofabrikReader.**get_shp_pathname**(*subregion_name*, *layer_name=None*, *feature_name=None*, *data_dir=None*)

Get path(s) to .shp file(s) for a geographic (sub)region (by searching a local data directory).

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on Geofabrik free download server

- **layer_name** (*str | None*) – name of a .shp layer (e.g. `'railways'`), defaults to `None`
- **feature_name** (*str | None*) – name of a feature (e.g. `'rail'`); if `None` (default), all available features included
- **data_dir** (*str | None*) – directory where the search is conducted; if `None` (default), the default directory

Returns
path(s) to .shp file(s)

Return type
list

Examples:

```
>>> from pydriosm.reader import GeofabrikReader
>>> from pyhelpers.dirs import delete_dir
>>> import os

>>> gfr = GeofabrikReader()

>>> subrgn_name = 'london'
>>> file_format = ".shp"
>>> dat_dir = "tests\osm_data"

>>> # Try to get the shapefiles' pathnames
>>> london_shp_path = gfr.get_shp_pathname(subrgn_name, data_dir=dat_dir)
>>> london_shp_path  # An empty list if no data is available
[]

>>> # Download the shapefiles of London
>>> path_to_london_shp_zip = gfr.downloader.download_data(
...     subrgn_name, file_format, dat_dir, verbose=True, ret_download_path=True)
To download data in the format '.shp.zip' for the following geographic (sub)region(s):
  "Greater London"
  to "./tests/osm_data/greater-london/"
? [No]|Yes: yes
Downloading "greater-london-latest-free.shp.zip" 100%|■■■■■■■■■■| 196M/196M |...
  Saving "greater-london-latest-free.shp.zip" ...
      to "./tests/osm_data/greater-london/" ... Done.

>>> type(path_to_london_shp_zip)
list
>>> len(path_to_london_shp_zip)
1

>>> # Extract the downloaded .zip file
>>> gfr.SHP.unzip_shp_zip(path_to_london_shp_zip[0], verbose=True)
Extracting "./tests/osm_data/greater-london/greater-london-latest-free.shp.zip"
    to "./tests/osm_data/greater-london/greater-london-latest-free-shp/" ... Done.

>>> # Try again to get the shapefiles' pathnames
>>> london_shp_path = gfr.get_shp_pathname(subrgn_name, data_dir=dat_dir)
>>> len(london_shp_path) > 1
True
```

(continues on next page)

(continued from previous page)

```
>>> # Get the file path of 'railways' shapefile
>>> lyr_name = 'railways'
>>> railways_shp_path = gfr.get_shp_pathname(subrgn_name, lyr_name, data_dir=dat_dir)
>>> len(railways_shp_path)
1
>>> railways_shp_path = railways_shp_path[0]
>>> os.path.relpath(railways_shp_path)
'tests\osm_data\greater-london\greater-london-latest-free-shp\gis_osm_railways_...

>>> # Get/save shapefile data of features labelled 'rail' only
>>> feat_name = 'rail'
>>> railways_shp = gfr.SHP.read_layer_shps(
...     railways_shp_path, feature_names=feat_name, save_feat_shp=True)
>>> railways_shp.head()
    osm_id  code  ...                                        coordinates shape_type
0    30804  6101  ...  [(0.0048644, 51.6279262), (0.0061979, 51.62926...          3
3   101511  6101  ...  [(-0.2119027, 51.5241906), (-0.2108059, 51.523...          3
5   361978  6101  ...  [(-0.0298545, 51.6619398), (-0.0302322, 51.659...          3
6  2370155  6101  ...  [(-0.3379005, 51.5937776), (-0.3367807, 51.593...          3
7  2526598  6101  ...  [(-0.1886021, 51.3602632), (-0.1884216, 51.360...          3
[5 rows x 9 columns]

>>> # Get the file path to the data of 'rail'
>>> rail_shp_path = gfr.get_shp_pathname(subrgn_name, lyr_name, feat_name, dat_dir)
>>> len(rail_shp_path)
1
>>> rail_shp_path = rail_shp_path[0]
>>> os.path.relpath(rail_shp_path)
'tests\osm_data\greater-london\greater-london-latest-free-shp\gis_osm_railways_...

>>> # Retrieve the data of 'rail' feature
>>> railways_rail_shp = gfr.SHP.read_layer_shps(rail_shp_path)
>>> railways_rail_shp.head()
    osm_id  code  ...                                        coordinates shape_type
0    30804  6101  ...  [(0.0048644, 51.6279262), (0.0061979, 51.62926...          3
1   101511  6101  ...  [(-0.2119027, 51.5241906), (-0.2108059, 51.523...          3
2   361978  6101  ...  [(-0.0298545, 51.6619398), (-0.0302322, 51.659...          3
3  2370155  6101  ...  [(-0.3379005, 51.5937776), (-0.3367807, 51.593...          3
4  2526598  6101  ...  [(-0.1886021, 51.3602632), (-0.1884216, 51.360...          3
[5 rows x 9 columns]

>>> # Delete the example data and the test data directory
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

GeofabrikReader.make_shp_pkl_pathname

classmethod GeofabrikReader.**make_shp_pkl_pathname**(*shp_zip_filename*, *extract_dir*, *layer_names_*, *feature_names_*)

Make a pathname of a pickle file for saving shapefile data.

Parameters

- **shp_zip_filename** (*str*) – filename of a .shp.zip file

- **extract_dir** (*str*) – pathname of a directory to which the .shp.zip file is extracted
- **layer_names** (*list*) – names of shapefile layers
- **feature_names** (*list*) – names of shapefile features

Returns
pathname of a pickle file for saving data of the specified shapefile

Return type
str

See examples for the methods *GeofabrikReader.read_shp()* and *BBBikeReader.read_shp()*.

GeofabrikReader.merge_shp_layers

GeofabrikReader.**merge_shp_layers**(*subregion_names*, *layer_name*, *data_dir=None*, *engine='pyshp'*, *update=False*, *download=False*, *rm_zip_extracts=True*, *merged_shp_dir=None*, *rm_shp_temp=True*, *verbose=False*, *ret_merged_shp_path=False*)

Merge shapefiles for a specific layer of two or multiple geographic regions.

Parameters

- **subregion_names** (*list*) – names of geographic region (case-insensitive) that is available on Geofabrik free download server
- **layer_name** (*str*) – name of a layer (e.g. 'railways')
- **engine** (*str*) – the method used to merge/save shapefiles; options include: `'pyshp'` (default) and `'geopandas'` (or `'gpd'`) if `engine='geopandas'`, this function relies on geopandas.GeoDataFrame.to_file(); otherwise, it by default uses shapefile.Writer()
- **update** (*bool*) – whether to update the source .shp.zip files, defaults to `False`
- **download** (*bool*) – whether to ask for confirmation before starting to download a file, defaults to `True`
- **data_dir** (*str | None*) – directory where the .shp.zip data files are located/saved; if `None` (default), the default directory
- **rm_zip_extracts** (*bool*) – whether to delete the extracted files, defaults to `False`
- **rm_shp_temp** (*bool*) – whether to delete temporary layer files, defaults to `False`
- **merged_shp_dir** (*str | None*) – if `None` (default), use the layer name as the name of the folder where the merged .shp files will be saved

- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **ret_merged_shp_path** (*bool*) – whether to return the path to the merged .shp file, defaults to `False`

Returns
the path to the merged file when `ret_merged_shp_path=True`

Return type
list | str

Examples:

```
>>> from pydriosm.reader import GeofabrikReader
>>> from pyhelpers.dirs import cd, delete_dir
>>> import os

>>> gfr = GeofabrikReader()
```

Example 1:

```
>>> # To merge 'railways' of Greater Manchester and West Yorkshire
>>> subrgn_name = ['Manchester', 'West Yorkshire']
>>> lyr_name = 'railways'
>>> dat_dir = "tests\osm_data"
>>> path_to_merged_shp_file = gfr.merge_shp_layers(
...     subrgn_name, lyr_name, dat_dir, verbose=True, ret_merged_shp_path=True)
To download data in the format '.shp.zip' for the following geographic (sub)region(s):
  "West Yorkshire"
  "Greater Manchester"
  to "./tests/osm_data/"
? [No]|Yes: >? yes
Downloading "greater-manchester-latest-free.shp.zip" 100%|██████████| 87.5M/8...
  Saving "greater-manchester-latest-free.shp.zip" ...
      to "./tests/osm_data/greater-manchester/" ... Done.
Downloading "west-yorkshire-latest-free.shp.zip" 100%|██████████| 87.3M/87.3M...
  Saving "west-yorkshire-latest-free.shp.zip" ...
      to "./tests/osm_data/west-yorkshire/" ... Done.
Merging the following shapefiles:
    "greater-manchester_gis_osm_railways_free_1.shp"
    "west-yorkshire_gis_osm_railways_free_1.shp"
        In progress ... Done.
        Find the merged shapefile at "tests/osm_data/gre_man-wes_yor-railways".

>>> path_to_merged_shp_file = path_to_merged_shp_file[0]
>>> os.path.relpath(path_to_merged_shp_file)
'tests\osm_data\gre_man-wes_yor-railways\linestring.shp'

>>> # Read the merged data
>>> manchester_yorkshire_railways_shp = gfr.SHP.read_shp(path_to_merged_shp_file)
>>> manchester_yorkshire_railways_shp.head()
    osm_id  code  ...                                        coordinates shape_type
0   928999  6101  ...  [(-2.2844621, 53.4802635), (-2.2949851, 53.481...          3
1   929904  6101  ...  [(-2.2917977, 53.4619559), (-2.2924877, 53.461...          3
2   929905  6102  ...  [(-2.2794048, 53.4605819), (-2.2799722, 53.460...          3
3  3663332  6102  ...  [(-2.2382139, 53.4817985), (-2.2381708, 53.481...          3
```

(continues on next page)

(continued from previous page)

```
4  3996086  6101  ...  [(-2.6003053, 53.4604346), (-2.6005261, 53.460...          3
[5 rows x 9 columns]

>>> # Delete the merged files
>>> delete_dir(os.path.dirname(path_to_merged_shp_file), verbose=True)
To delete the directory "./tests/osm_data/gre_man-wes_yor-railways/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/gre_man-wes_yor-railways/" ... Done.

>>> # Delete the downloaded .shp.zip data files
>>> delete_dir(list(map(os.path.dirname, gfr.downloader.data_paths)), verbose=True)
To delete the following directories:
  "./tests/osm_data/greater-manchester/" (Not empty)
  "./tests/osm_data/west-yorkshire/" (Not empty)
? [No]|Yes: yes
Deleting:
  "./tests/osm_data/greater-manchester/" ... Done.
  "./tests/osm_data/west-yorkshire/" ... Done.
```

Example 2:

```
>>> # To merge 'transport' of Greater London, Kent and Surrey

>>> subrgn_name = ['London', 'Kent', 'Surrey']
>>> lyr_name = 'transport'
>>> path_to_merged_shp_file = gfr.merge_shp_layers(
...     subrgn_name, lyr_name, dat_dir, verbose=True, ret_merged_shp_path=True)
To download data in the format '.shp.zip' for the following geographic (sub)region(s):
  "Kent"
  "Surrey"
  "Greater London"
  to "./tests/osm_data/"
? [No]|Yes: >? yes
Downloading "greater-london-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 196M/196M |...
  Saving "greater-london-latest-free.shp.zip" ...
      to "./tests/osm_data/greater-london/" ... Done.
Downloading "kent-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 89.1M/89.1M | 912kB/s...
  Saving "kent-latest-free.shp.zip" to "./tests/osm_data/kent/" ... Done.
Downloading "surrey-latest-free.shp.zip" 100%|■■■■■■■■■■■■| 73.7M/73.7M | 1.00M...
  Saving "surrey-latest-free.shp.zip" to "./tests/osm_data/surrey/" ... Done.
Merging the following shapefiles:
    "greater-london_gis_osm_transport_a_free_1.shp"
    "greater-london_gis_osm_transport_free_1.shp"
    "kent_gis_osm_transport_a_free_1.shp"
    "kent_gis_osm_transport_free_1.shp"
    "surrey_gis_osm_transport_a_free_1.shp"
    "surrey_gis_osm_transport_free_1.shp"
        In progress ... Done.
        Find the merged shapefile at "tests/osm_data/gre_lon-ken-sur-transport".

>>> type(path_to_merged_shp_file)
list
>>> len(path_to_merged_shp_file)
2
>>> os.path.relpath(path_to_merged_shp_file[0])
'tests\osm_data\gre_lon-ken-sur-transport\point.shp'
```

(continues on next page)

(continued from previous page)

```
>>> os.path.relpath(path_to_merged_shp_file[1])
'tests\osm_data\gre-lon_ken_sur_transport\polygon.shp'

>>> # Read the merged shapefile
>>> merged_transport_shp_1 = gfr.SHP.read_shp(path_to_merged_shp_file[1])
>>> merged_transport_shp_1.head()
     osm_id  ...  shape_type
0   5077928  ...           5
1   8610280  ...           5
2  15705264  ...           5
3  23077379  ...           5
4  24016945  ...           5
[5 rows x 6 columns]

>>> # Delete the merged files
>>> delete_dir(os.path.commonpath(path_to_merged_shp_file), verbose=True)
To delete the directory "./tests/osm_data/gre_lon-ken-sur-transport/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/gre_lon-ken-sur-transport/" ... Done.

>>> # Delete the example data and the test data directory
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

GeofabrikReader.read_gpkg

GeofabrikReader.**read_gpkg**(*subregion_name*, *layer_names=None*, *feature_names=None*, *data_dir=None*, *update=False*, *download=False*, *verbose=False*, *raise_error=True*, ***kwargs*)

Reads GeoPackage (.gpkg.zip) data for a specific subregion.

Parameters

- **subregion_name** (*str*) – Name of the subregion (e.g. 'West Midlands' or 'london').
- **layer_names** (*str | list | None*) – Specific OSM layer(s) to load (e.g., 'points', 'roads'); defaults to None (loads all available layers).
- **feature_names** (*str | list | None*) – Specific feature type(s) to extract from the loaded layers (e.g., 'residential', 'motorway'); defaults to None.
- **data_dir** (*str | os.PathLike | None*) – Directory where the data file is stored or will be downloaded to; defaults to None.
- **update** (*bool*) – Whether to update the local file by re-downloading it; defaults to False.
- **download** (*bool*) – Whether to download the file if it is missing locally; defaults to False.
- **verbose** (*bool*) – Whether to print progress messages to the console; defaults to False.

- **raise_error** (*bool*) – Whether to raise an exception if an error occurs during parsing; defaults to `True`.
- **kwargs** (*Any*) – Additional optional arguments for `pyhelpers.store.load_geopackage()`.

Returns

A GeoDataFrame (if a single layer is resolved) or a dictionary of GeoDataFrames (keyed by layer names).

Return type

geopandas.GeoDataFrame | dict | None

Examples:

```
>>> from pydriosm.reader import GeofabrikReader
>>> from pyhelpers.dirs import cd, delete_dir
>>> gbr = GeofabrikReader()
>>> subregion_name = 'West midlands'
>>> data_dir = "tests/osm_data"
>>> wm_gpkg = gbr.read_gpkg(subregion_name, data_dir=data_dir, verbose=True)
Traceback (most recent call last):
    ...
FileNotFoundError: The shapefile "west-midlands-latest-free.gpkg.zip" is not availa...
  Set `download=True` to download it.
>>> wm_gpkg = gbr.read_gpkg(
...     subregion_name, data_dir=data_dir, verbose=True, download=True)
Downloading "west-midlands-latest-free.gpkg.zip" 100%|■■■■■■■■■■■■| 103M/103M |...
  Saving "west-midlands-latest-free.gpkg.zip" to "./tests/osm_data/west-midlands/" ...
Parsing the data ... Done.
>>> type(wm_gpkg)
dict
>>> list(wm_gpkg.keys())
['traffic',
 'pois',
 'transport',
 'places',
 'pofw',
 'natural',
 'roads',
 'railways',
 'waterways',
 'landuse',
 'water',
 'protected_areas',
 'buildings',
 'adminareas']
>>> delete_dir(data_dir, verbose=True)  # Delete the downloaded .csv.xz data file
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

GeofabrikReader.read_pbf

GeofabrikReader.**read_pbf**(*subregion_name*, *data_dir=None*, *readable=False*, *expand=False*, *parse_geometry=False*, *parse_properties=False*, *parse_other_tags=False*, *update=False*, *download=True*, *pickle_it=False*, *ret_pickle_path=False*, *rm_pbf_file=False*, *chunk_size_limit=50*, *verbose=False*, ***kwargs*)

Read a PBF (.osm.pbf) data file of a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on Geofabrik free download server
- **data_dir** (*str | None*) – directory where the .osm.pbf data file is located/saved; if None, the default local directory
- **readable** (*bool*) – whether to parse each feature in the raw data, defaults to False
- **expand** (*bool*) – whether to expand dict-like data into separate columns, defaults to False
- **parse_geometry** (*bool*) – whether to represent the 'geometry' field in a shapely.geometry format, defaults to False
- **parse_properties** (*bool*) – whether to represent the 'properties' field in a tabular format, defaults to False
- **parse_other_tags** (*bool*) – whether to represent a 'other_tags' (of 'properties') in a dict format, defaults to False
- **download** (*bool*) – whether to download/update the PBF data file of the given subregion, if it is not available at the specified path, defaults to True
- **update** (*bool*) – whether to check to update pickle backup (if available), defaults to False
- **pickle_it** (*bool*) – whether to save the .pbf data as a pickle file, defaults to False
- **ret_pickle_path** (*bool*) – (when pickle_it=True) whether to return a path to the saved pickle file
- **rm_pbf_file** (*bool*) – whether to delete the downloaded .osm.pbf file, defaults to False
- **chunk_size_limit** (*int | None*) – threshold (in MB) that triggers the use of chunk parser, defaults to 50; if the size of the .osm.pbf file (in MB) is greater than chunk_size_limit, it will be parsed in a chunk-wise way
- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to False
- **kwargs** – [optional] parameters of *PBF.read_pbf()*.

Returns

dictionary of the .osm.pbf data; when `pickle_it=True`, return a tuple of the dictionary and a path to the pickle file

Return type

dict | tuple | None

Examples:

```
>>> from pydriosm.reader import GeofabrikReader
>>> from pyhelpers.dirs import delete_dir
>>> gfr = GeofabrikReader()
>>> subregion_name = 'rutland'
>>> data_dir = "tests/osm_data"
>>> # If the PBF data of Rutland is not available at the specified data directory,
>>> # the function can download the latest data by setting `download=True` (default)
>>> pbf_raw = gfr.read_pbf(subregion_name, data_dir=data_dir, verbose=True)
Downloading "rutland-latest.osm.pbf" 100%|■■■■■■■■■■■■| 1.83M/1.83M | 5.63MB/s ...
    Saving "rutland-latest.osm.pbf" to "./tests/osm_data/rutland/" ... Done.
Reading "./tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
>>> type(pbf_raw)
dict
>>> list(pbf_raw.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']
>>> pbf_raw_points = pbf_raw['points']
>>> type(pbf_raw_points)
list
>>> type(pbf_raw_points[0])
osgeo.ogr.Feature
>>> # Set `readable=True`
>>> pbf_parsed = gfr.read_pbf(subregion_name, data_dir, readable=True, verbose=True)
Parsing "./tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
>>> pbf_parsed_points = pbf_parsed['points']
>>> pbf_parsed_points.head()
0    {'type': 'Feature', 'geometry': {'type': 'Poin...
1    {'type': 'Feature', 'geometry': {'type': 'Poin...
2    {'type': 'Feature', 'geometry': {'type': 'Poin...
3    {'type': 'Feature', 'geometry': {'type': 'Poin...
4    {'type': 'Feature', 'geometry': {'type': 'Poin...
Name: points, dtype: object
>>> # Set `expand=True`, which would force `readable=True`
>>> pbf_parsed_ = gfr.read_pbf(subregion_name, data_dir, expand=True, verbose=True)
Parsing "./tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
>>> pbf_parsed_points_ = pbf_parsed_['points']
>>> pbf_parsed_points_.head()
         id  ...                                         properties
0    488658  ...  {'osm_id': '488658', 'name': 'Tickencote Inter...
1  13883868  ...  {'osm_id': '13883868', 'name': None, 'barrier'...
2  14049101  ...  {'osm_id': '14049101', 'name': None, 'barrier'...
3  14558402  ...  {'osm_id': '14558402', 'name': None, 'barrier'...
4  14558409  ...  {'osm_id': '14558409', 'name': None, 'barrier'...
[5 rows x 3 columns]
>>> # Set `readable` and `parse_geometry` to be `True`
>>> pbf_parsed_1 = gfr.read_pbf(
...     subregion_name, data_dir, readable=True, parse_geometry=True)
>>> pbf_parsed_1_point = pbf_parsed_1['points'][0]
>>> pbf_parsed_1_point['geometry']
'POINT (-0.5313354 52.6737716)'
```

(continues on next page)

(continued from previous page)

```
>>> pbf_parsed_1_point['properties']['highway']
'motorway_junction'

>>> # Set `readable` and `parse_other_tags` to be `True`
>>> pbf_parsed_2 = gfr.read_pbf(
...     subregion_name, data_dir, readable=True, parse_other_tags=True)
>>> pbf_parsed_2_point = pbf_parsed_2['points'][0]
>>> pbf_parsed_2_point['geometry']
{'type': 'Point', 'coordinates': [-0.5313354, 52.6737716]}
>>> pbf_parsed_2_point['properties']['highway']
'motorway_junction'
>>> # Set `readable`, `parse_geometry` and `parse_other_tags` to be `True`
>>> pbf_parsed_3 = gfr.read_pbf(
...     subregion_name, data_dir, readable=True, parse_geometry=True,
...     parse_other_tags=True)
>>> pbf_parsed_3_point = pbf_parsed_3['points'][0]
>>> pbf_parsed_3_point['geometry']
'POINT (-0.5313354 52.6737716)'
>>> pbf_parsed_3_point['properties']['highway']
'motorway_junction'
>>> # Delete the example data and the test data directory
>>> delete_dir(data_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

GeofabrikReader.read_shp

GeofabrikReader.**read_shp**(*subregion_name*, *layer_names=None*, *feature_names=None*, *data_dir=None*, *update=False*, *download=False*, *pickle_it=False*, *ret_pickle_path=False*, *rm_extracts=False*, *rm_shp_zip=False*, *verbose=False*, ***kwargs*)

Read a .shp.zip data file of a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on Geofabrik free download server
- **layer_names** (*str | list | None*) – name of a .shp layer, e.g. 'railways', or names of multiple layers; if None (default), all available layers
- **feature_names** (*str | list | None*) – name of a feature, e.g. 'rail', or names of multiple features; if None (default), all available features
- **data_dir** (*str | None*) – directory where the .shp.zip data file is located/saved; if None, the default directory
- **update** (*bool*) – whether to check to update pickle backup (if available), defaults to False
- **download** (*bool*) – whether to ask for confirmation before starting to download a file, defaults to True

- **pickle_it** (*bool*) – whether to save the .shp data as a pickle file, defaults to `False`
- **ret_pickle_path** (*bool*) – (when `pickle_it=True`) whether to return a path to the saved pickle file
- **rm_extracts** (*bool*) – whether to delete extracted files from the .shp.zip file, defaults to `False`
- **rm_shp_zip** (*bool*) – whether to delete the downloaded .shp.zip file, defaults to `False`
- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns

dictionary of the shapefile data, with keys and values being layer names and tabular data (in the format of geopandas.GeoDataFrame), respectively

Return type

dict | None

Examples:

```
>>> from pydriosm.reader import GeofabrikReader
>>> from pyhelpers.dirs import delete_dir

>>> gfr = GeofabrikReader()

>>> subrgn_name = 'London'
>>> dat_dir = "tests/osm_data"
>>> london_shp_data = gfr.read_shp(
...     subregion_name=subrgn_name, data_dir=dat_dir, download=False, verbose=True)
The .shp.zip file for "Greater London" is not found.

>>> # Set `download=True`
>>> london_shp_data = gfr.read_shp(
...     subregion_name=subrgn_name, data_dir=dat_dir, download=True, verbose=True)
Downloading "greater-london-latest-free.shp.zip" 100%|■■■■■■■■■■| 196M/196M |...
  Saving "greater-london-latest-free.shp.zip" ...
      to "./tests/osm_data/greater-london/" ... Done.
Extracting "./tests/osm_data/greater-london/greater-london-latest-free.shp.zip"
    to "./tests/osm_data/greater-london/greater-london-latest-free-shp/" ... Done.
Reading the shapefile(s) at "./tests/osm_data/greater-london/greater-london-latest-...
>>> type(london_shp_data)
dict
>>> list(london_shp_data.keys())
['buildings',
 'landuse',
 'natural',
 'places',
 'pofw',
 'pois',
 'railways',
 'roads',
 'traffic',
 'transport',
```

(continues on next page)

(continued from previous page)

```
 'water',
 'waterways']

>>> # Data of the 'railways' layer
>>> london_shp_railways = london_shp_data['railways']
>>> london_shp_railways.head()
   osm_id  code  ...                                        coordinates shape_type
0   30804  6101  ...  [(0.0048644, 51.6279262), (0.0061979, 51.62926...          3
1  101298  6103  ...  [(-0.2249906, 51.493682), (-0.2251678, 51.4945...          3
2  101486  6103  ...  [(-0.2055497, 51.5195429), (-0.2051377, 51.519...          3
3  101511  6101  ...  [(-0.2119027, 51.5241906), (-0.2108059, 51.523...          3
4  282898  6103  ...  [(-0.1862586, 51.6159083), (-0.1868721, 51.613...          3
[5 rows x 9 columns]

>>> # Read data of the 'transport' layer only from the original .shp.zip file
>>> # (and delete any extracts)
>>> subrgn_layer = 'transport'

>>> # Set `rm_extracts=True` to remove the extracts
>>> london_shp_transport = gfr.read_shp(
...     subregion_name=subrgn_name, layer_names=subrgn_layer, data_dir=dat_dir,
...     rm_extracts=True, verbose=True)
Reading the shapefile(s) at "./tests/osm_data/greater-london/greater-london-latest-...
Deleting the extracts "./tests/osm_data/greater-london/greater-london-latest-free-s...
>>> type(london_shp_transport)
dict
>>> list(london_shp_transport.keys())
['transport']
>>> london_shp_transport_ = london_shp_transport['transport']
>>> london_shp_transport_.head()
     osm_id  ...  shape_type
0   5077928  ...           5
1   8610280  ...           5
2  15705264  ...           5
3  23077379  ...           5
4  24016945  ...           5
[5 rows x 6 columns]

>>> # Read data of only the 'bus_stop' feature (in the 'transport' layer)
>>> # from the original .shp.zip file (and delete any extracts)
>>> feat_name = 'bus_stop'
>>> london_bus_stop = gfr.read_shp(
...     subregion_name=subrgn_name, layer_names=subrgn_layer, feature_names=feat_name,
...     data_dir=dat_dir, rm_extracts=True, verbose=True)
Extracting the following layer(s):
    'transport'
        from "./tests/osm_data/greater-london/greater-london-latest-free.shp.zip" ...
            to "./tests/osm_data/greater-london/greater-london-latest-free-shp/" .....
Reading the shapefile(s) at "./tests/osm_data/greater-london/greater-london-latest-...
Deleting the extracts "./tests/osm_data/greater-london/greater-london-latest-free-s...
>>> type(london_bus_stop)
dict
>>> list(london_bus_stop.keys())
['transport']

>>> fclass = london_bus_stop['transport'].fclass.unique()
>>> fclass
```

(continues on next page)

(continued from previous page)

```
<StringArray>
['bus_stop']
Length: 1, dtype: str

>>> # Read multiple features of multiple layers
>>> # (and delete both the original .shp.zip file and extracts)
>>> subrgn_layers = ['traffic', 'roads']
>>> feat_names = ['parking', 'trunk']
>>> london_shp_tra_roa_par_tru = gfr.read_shp(
...     subregion_name=subrgn_name, layer_names=subrgn_layers, feature_names=feat_names,
...     data_dir=dat_dir, rm_extracts=True, rm_shp_zip=True, verbose=True)
Extracting the following layer(s):
    'traffic'
    'roads'
        from "./tests/osm_data/greater-london/greater-london-latest-free.shp.zip" ...
            to "./tests/osm_data/greater-london/greater-london-latest-free-shp/" .....
Reading the shapefile(s) at "./tests/osm_data/greater-london/greater-london-latest-...
Deleting the extracts "./tests/osm_data/greater-london/greater-london-latest-free-s...
Deleting "tests/osm_data/greater-london/greater-london-latest-free.shp.zip" ... Done.
>>> type(london_shp_tra_roa_par_tru)
dict
>>> list(london_shp_tra_roa_par_tru.keys())
['traffic', 'roads']

>>> # Data of the 'traffic' layer
>>> london_shp_tra_roa_par_tru['traffic'].head()
    osm_id  code  ...                                        coordinates shape_type
0  2956081  5260  ...  [(-0.0218269, 51.4369515), (-0.020097, 51.4372...          5
1  2956183  5260  ...  [(-0.0224697, 51.4452646), (-0.0223272, 51.445...          5
2  2956184  5260  ...  [(-0.0186703, 51.444221), (-0.0185442, 51.4447...          5
3  2956185  5260  ...  [(-0.0189846, 51.4481958), (-0.0189417, 51.448...          5
4  2956473  5260  ...  [(-0.0059602, 51.4579088), (-0.0058695, 51.457...          5
[5 rows x 6 columns]

>>> # Data of the 'roads' layer
>>> london_shp_tra_roa_par_tru['roads'].head()
   osm_id  code  ...                                        coordinates shape_type
7    1200  5112  ...  [(-0.2916285, 51.5160418), (-0.2915517, 51.516...          3
8    1201  5112  ...  [(-0.2925582, 51.5300857), (-0.2925916, 51.529...          3
9    1202  5112  ...  [(-0.2230893, 51.5735075), (-0.2228416, 51.573...          3
10   1203  5112  ...  [(-0.139105, 51.6101568), (-0.1395372, 51.6100...          3
11   1208  5112  ...  [(-0.1176027, 51.6124616), (-0.1169584, 51.612...          3
[5 rows x 12 columns]

>>> # Delete the example data and the test data directory
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

GeofabrikReader.read_var_osm

GeofabrikReader.**read_var_osm**(*meth*, *subregion_name*, *osm_file_format*, *data_dir=None*, *download=False*, *verbose=False*, *raise_error=True*, ***kwargs*)

Read data file of various formats (other than PBF and shapefile) for a geographic

(sub)region.

Parameters

- **meth** (*Callable*) – name of a class method for getting (auxiliary) prepacked data
- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on a free download server
- **osm_file_format** (*str*) – format (file extension) of OSM data
- **data_dir** (*str | None*) – directory where the data file is located/saved, defaults to `None`; when `data_dir=None`, it refers to the directory specified by the corresponding downloader
- **download** (*bool*) – whether to download/update the PBF data file of the given subregion, if it is not available at the specified path, defaults to `False`
- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **kwargs** – [optional] parameters of the method specified by `meth`

Returns
data of the specified file format

Return type
pandas.DataFrame | None

See examples for the methods *BBBikeReader.read_csv_xz()* and *BBBikeReader.read_geojson_xz()*.

GeofabrikReader.remove_extracts

classmethod GeofabrikReader.**remove_extracts**(*path_to_extract_dir*, *verbose*)

Remove data extracts.

Parameters

- **path_to_extract_dir** (*str | os.PathLike[str]*) – pathname of the directory where data extracts are stored
- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to `False`

See examples for the methods *GeofabrikReader.read_shp()* and *BBBikeReader.read_shp()*.

GeofabrikReader.validate_dtype

classmethod GeofabrikReader.**validate_dtype**(*x*)

Validate the data type of the input variable.

Parameters
x (*str | list | None*) – a variable

Returns
validated input

Return type
list

Tests:

```
>>> from pydriosm.reader._base import BaseReader
>>> BaseReader.validate_dtype(x=None)
[]
>>> BaseReader.validate_dtype(x='str')
['str']
>>> BaseReader.validate_dtype(x=['str'])
['str']
```

GeofabrikReader.validate_file_path

classmethod GeofabrikReader.**validate_file_path**(*path_to_file*, *osm_filename=None*, *data_dir=None*)

Validate the pathname of an OSM data file.

Parameters

- **path_to_file** (*str | os.PathLike[str]*) – pathname of an OSM data file
- **osm_filename** (*str*) – filename of the OSM data file
- **data_dir** (*str | os.PathLike[str]*) – name or pathname of the data directory

Returns
validated pathname of the specified OSM data file

Return type
str

Tests:

```
>>> from pydriosm.reader._base import BaseReader
>>> import os

>>> file_path = BaseReader.validate_file_path("a\b\c.osm.pbf")
>>> file_path
'a\b\c.osm.pbf'
>>> file_path = BaseReader.validate_file_path("a\b\c.osm.pbf", "x.y.z", data_dir="a\b")
```

(continues on next page)

(continued from previous page)

```
>>> os.path.relpath(file_path)
'a\b\x.y.z'
```

GeofabrikReader.validate_shp_layer_names

GeofabrikReader.**validate_shp_layer_names**(*layer_names_*, *extract_dir*, *shp_zip_pathname*, *subregion_name*, *osm_file_format*, *data_dir*, *update*, *download*, *verbose*)

Validate the input of layer name(s) for reading shapefiles.

Parameters

- **layer_names** (*list*) – names of shapefile layers
- **extract_dir** (*str*) – pathname of a directory to which the .shp.zip file is extracted
- **shp_zip_pathname** (*str*) – pathname of a .shp.zip file
- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on Geofabrik free download server
- **osm_file_format** (*str*) – format (file extension) of OSM data
- **data_dir** (*str | os.PathLike[str]*) – name or pathname of the data directory
- **update** (*bool*) – whether to check to update pickle backup (if available), defaults to False
- **download** (*bool*) – whether to download/update the PBF data file of the given subregion, if it is not available at the specified path, defaults to True
- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to False

Returns

validated shapefile layer names

Return type

list

See examples for the methods *GeofabrikReader.read_shp()* and *BBBikeReader.read_shp()*.

BBBikeReader

class pydriosm.reader.**BBBikeReader**(*data_dir=None*, *max_tmpfile_size=None*)

Read BBBike exports of OpenStreetMap data.

Parameters

- **data_dir** (*str | None*) – (a path or a name of) a directory where a data file is; if None (default), a folder osm_bbbike under the current working directory

- **max_tmpfile_size** (*int | None*) – defaults to None, see also gdal_configurations

Variables

- **downloader** (BBBikeDownloader) – instance of the class *BBBikeDownloader*
- **name** (*str*) – name of the data resource
- **url** (*str*) – url of the homepage to the BBBike free download server

Examples:

```
>>> from pydriosm.reader import BBBikeReader
>>> bbr = BBBikeReader()
>>> bbr.NAME
'BBBike'
```

Attributes

1anti-flashwhitewhite

DEFAULT_DATA_DIR	Default download directory.
FILE_FORMATS	Valid file formats.
LONG_NAME	Full name of the data source.
NAME	Name of the data source.
data_dir	Name or pathname of a data directory.
data_paths	Pathnames of all data files.

BBBikeReader.DEFAULT_DATA_DIR

BBBikeReader.**DEFAULT_DATA_DIR: str = 'osm_data/bbbike'**

Default download directory.

BBBikeReader.FILE_FORMATS

BBBikeReader.**FILE_FORMATS: set = {'.csv.xz', '.garmin-onroad-latin1.zip', '.garmin-ontrail-latin1.zip', '.garmin-opentopo-latin1.zip', '.garmin-osm.zip', '.geojson.xz', '.gz', '.mapsforge-osm.zip', '.mbtiles-openmaptiles.zip', '.pbf', '.shp.zip'}**

Valid file formats.

BBBikeReader.LONG_NAME

BBBikeReader.**LONG_NAME: str = 'BBBike exports of OpenStreetMap data'**

Full name of the data source.

BBBikeReader.NAME

BBBikeReader.**NAME: str = 'BBBike'**

Name of the data source.

BBBikeReader.data_dir

property BBBikeReader.**data_dir**

Name or pathname of a data directory.

Returns
name or pathname of a directory for saving downloaded data files

Return type
str | None

Tests:

```
>>> from pydriosm.reader._base import BaseReader
>>> from pydriosm.downloader import GeofabrikDownloader, BBBikeDownloader
>>> import os
>>> brd = BaseReader()
>>> os.path.relpath(brd.data_dir)
'osm_data'
>>> brd = BaseReader(downloader=GeofabrikDownloader)
>>> os.path.relpath(brd.data_dir)
'osm_data'
>>> brd = BaseReader(downloader=BBBikeDownloader)
>>> os.path.relpath(brd.data_dir)
'osm_data'
```

BBBikeReader.data_paths

property BBBikeReader.**data_paths**

Pathnames of all data files.

Returns
pathnames of all data files

Return type
list

Tests:

```
>>> from pydriosm.reader._base import BaseReader
>>> brd = BaseReader()
>>> brd.data_paths
[]
```

Methods

1anti-flashwhitewhite

cdd(*sub_dir[, mkdir])	Change directory to default data directory and its subdirectories or a specific file.
get_file_path(subregion_name, osm_file_format)	Get the local path to an OSM data file of a geographic (sub)region.
get_pbf_layer_names(subregion_name[, data_dir])	Get indices and names of all layers in the PBF data file of a given (sub)region.
get_shp_pathname(subregion_name[, ...])	Get path(s) to shapefile(s) for a geographic (sub)region (by searching a local data directory).
make_shp_pkl_pathname(shp_zip_filename, ...)	Make a pathname of a pickle file for saving shapefile data.
merge_shp_layers(subregion_names, layer_name)	Merge shapefiles for a specific layer of two or multiple geographic regions.
read_csv_xz(subregion_name[, data_dir, ...])	Read a compressed CSV (.csv.xz) data file of a geographic (sub)region.
read_geojson_xz(subregion_name[, data_dir, ...])	Read a .geojson.xz data file of a geographic (sub)region.
read_gpkg(subregion_name[, layer_names, ...])	Reads GeoPackage (.gpkg.zip) data for a specific subregion.
read_pbf(subregion_name[, data_dir, ...])	Read a PBF (.osm.pbf) data file of a geographic (sub)region.
read_shp(subregion_name[, layer_names, ...])	Read a shapefile of a geographic (sub)region.
read_var_osm(meth, subregion_name, ...[, ...])	Read data file of various formats (other than PBF and shapefile) for a geographic (sub)region.
remove_extracts(path_to_extract_dir, verbose)	Remove data extracts.
validate_dtype(x)	Validate the data type of the input variable.
validate_file_path(path_to_file[, ...])	Validate the pathname of an OSM data file.
validate_shp_layer_names(layer_names_, ...)	Validate the input of layer name(s) for reading shapefiles.

BBBikeReader.cdd

classmethod BBBikeReader.**cdd**(**sub_dir*, *mkdir=False*, ***kwargs*)

Change directory to default data directory and its subdirectories or a specific file.

Parameters

- **sub_dir** (*str | os.PathLike[str]*) – name of directory; names of directories (and/or a filename)
- **mkdir** (*bool*) – whether to create a directory, defaults to False

- **kwargs** – [optional] parameters of the function pyhelpers.dir.cd()

Returns
an absolute pathname to a directory (or a file)

Return type
str | os.PathLike[str]

Tests:

```
>>> from pydriosm.reader._base import BaseReader
>>> import os
>>> os.path.relpath(BaseReader.cdd())
'osm_data'
>>> os.path.exists(BaseReader.cdd())
False
```

BBBikeReader.get_file_path

BBBikeReader.**get_file_path**(*subregion_name*, *osm_file_format*, *data_dir=None*)

Get the local path to an OSM data file of a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on Geofabrik free download server
- **osm_file_format** (*str*) – file format of the OSM data available on the free download server
- **data_dir** (*str | None*) – directory where the data file of the subregion_name is located/saved; if None (default), the default local directory

Returns
path to PBF (.osm.pbf) file

Return type
str | None

Examples:

```
>>> from pydriosm.reader import BBBikeReader
>>> from pyhelpers.dirs import delete_dir
>>> import os
>>> bbr = BBBikeReader()
>>> subregion_name = 'rutland'
>>> osm_file_format = ".pbf"
>>> data_dir = "tests/osm_data"
>>> path_to_pbf = bbr.get_file_path(subregion_name, osm_file_format, data_dir)
Traceback (most recent call last):
    ...
pydriosm.errors.InvalidSubregionNameError:
  `subregion_name='rutland'` -> The input of `subregion_name` is not recognizable.
  Check the `.data_source`, or try another one instead.
>>> subregion_name = 'birmingham'
```

(continues on next page)

(continued from previous page)

```
>>> path_to_pbf = bbr.get_file_path(subregion_name, osm_file_format, data_dir)
>>> # When "Birmingham.osm.pbf" is unavailable at the data directory
>>> os.path.isfile(path_to_pbf)
False
>>> # Download the PBF data file of Birmingham to "./tests/osm_data/"
>>> bbr.downloader.download_data(
...     subregion_name, osm_file_format, data_dir, verbose=True)
Proceed to download data in the format '.pbf' for the following geographic (sub)reg...
  "Birmingham"
  to "./tests/osm_data/birmingham/"
? [No]|Yes: yes
Downloading "Birmingham.osm.pbf" 100%|■■■■■■■■■■■| 55.8M/55.8M | 15.3MB/s | ET...
  Saving "Birmingham.osm.pbf" to "./tests/osm_data/birmingham/" ... Done.
>>> # Check again
>>> path_to_pbf = bbr.get_file_path(subregion_name, osm_file_format, data_dir)
>>> os.path.isfile(path_to_pbf)
True
>>> os.path.relpath(path_to_pbf)  # (on Windows)
'tests\osm_data\birmingham\Birmingham.osm.pbf'
>>> # Delete the test data directory
>>> delete_dir(data_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

BBBikeReader.get_pbf_layer_names

BBBikeReader.**get_pbf_layer_names**(*subregion_name*, *data_dir=None*)

Get indices and names of all layers in the PBF data file of a given (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on Geofabrik free download server
- **data_dir**

Returns

indices and names of each layer of the PBF data file

Return type

dict

Examples:

```
>>> from pydriosm.reader import BBBikeReader
>>> from pyhelpers.dirs import delete_dir
>>> import os
>>> bbr = BBBikeReader()
>>> # Download the PBF data of Birmingham as an example
>>> subregion_name = 'birmingham'
>>> osm_file_format = ".pbf"
>>> data_dir = "tests/osm_data"
>>> # Download the PBF data of Birmingham to "./tests/osm_data/"
>>> bbr.downloader.download_data(
```

(continues on next page)

(continued from previous page)

```
...     subregion_name, osm_file_format, data_dir, verbose=True)
Proceed to download data in the format '.pbf' for the following geographic (sub)reg...
  "Birmingham"
  to "./tests/osm_data/birmingham/"
? [No]|Yes: yes
Downloading "Birmingham.osm.pbf" 100%|■■■■■■■■■■| 55.8M/55.8M | 15.3MB/s | ET...
  Saving "Birmingham.osm.pbf" to "./tests/osm_data/birmingham/" ... Done.
>>> path_to_pbf = bbr.data_paths[0]
>>> os.path.relpath(path_to_pbf)
'tests\osm_data\birmingham\Birmingham.osm.pbf'
>>> lyr_idx_names = bbr.get_pbf_layer_names(subregion_name)
>>> lyr_idx_names
{0: 'points',
 1: 'lines',
 2: 'multilinestrings',
 3: 'multipolygons',
 4: 'other_relations'}
>>> # Delete the example data and the test data directory
>>> delete_dir(data_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

BBBikeReader.get_shp_pathname

BBBikeReader.**get_shp_pathname**(*subregion_name*, *layer_name=None*, *feature_name=None*, *data_dir=None*)

Get path(s) to shapefile(s) for a geographic (sub)region (by searching a local data directory).

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on Geofabrik free download server
- **layer_name** (*str | None*) – name of a .shp layer (e.g. `'railways'`), defaults to `None`
- **feature_name** (*str | None*) – name of a feature (e.g. `'rail'`); if `None` (default), all available features included
- **data_dir** (*str | None*) – directory where the search is conducted; if `None` (default), the default directory

Returns

path(s) to shapefile(s)

Return type

list

See examples for *pydriosm.reader.GeofabrikReader.get_shp_pathname()*.

BBBikeReader.make_shp_pkl_pathname

classmethod BBBikeReader.**make_shp_pkl_pathname**(*shp_zip_filename*, *extract_dir*, *layer_names_*, *feature_names_*)

Make a pathname of a pickle file for saving shapefile data.

Parameters

- **shp_zip_filename** (*str*) – filename of a .shp.zip file
- **extract_dir** (*str*) – pathname of a directory to which the .shp.zip file is extracted
- **layer_names** (*list*) – names of shapefile layers
- **feature_names** (*list*) – names of shapefile features

Returns
pathname of a pickle file for saving data of the specified shapefile

Return type
str

See examples for the methods *GeofabrikReader.read_shp()* and *BBBikeReader.read_shp()*.

BBBikeReader.merge_shp_layers

BBBikeReader.**merge_shp_layers**(*subregion_names*, *layer_name*, *data_dir=None*, *engine='pyshp'*, *update=False*, *download=False*, *rm_zip_extracts=True*, *merged_shp_dir=None*, *rm_shp_temp=True*, *verbose=False*, *ret_merged_shp_path=False*)

Merge shapefiles for a specific layer of two or multiple geographic regions.

Parameters

- **subregion_names** (*list*) – names of geographic region (case-insensitive) that is available on Geofabrik free download server
- **layer_name** (*str*) – name of a layer (e.g. 'railways')
- **engine** (*str*) – the method used to merge/save shapefiles; options include: `'pyshp'` (default) and `'geopandas'` (or `'gpd'`) if `engine='geopandas'`, this function relies on geopandas.GeoDataFrame.to_file(); otherwise, it by default uses shapefile.Writer()
- **update** (*bool*) – whether to update the source .shp.zip files, defaults to `False`
- **download** (*bool*) – whether to ask for confirmation before starting to download a file, defaults to `True`
- **data_dir** (*str | None*) – directory where the .shp.zip data files are located/saved; if `None` (default), the default directory

- **rm_zip_extracts** (*bool*) – whether to delete the extracted files, defaults to `False`
- **rm_shp_temp** (*bool*) – whether to delete temporary layer files, defaults to `False`
- **merged_shp_dir** (*str | None*) – if `None` (default), use the layer name as the name of the folder where the merged .shp files will be saved
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **ret_merged_shp_path** (*bool*) – whether to return the path to the merged .shp file, defaults to `False`

Returns

the path to the merged file when `ret_merged_shp_path=True`

Return type

list | str

Examples:

```
>>> from pydriosm.reader import BBBikeReader
>>> from pyhelpers.dirs import cd, delete_dir
>>> import os

>>> bbr = BBBikeReader()
```

Example 1:

```
>>> # To merge 'railways' of London and Birmingham
>>> subrgn_name = ['London', 'Birmingham']
>>> lyr_name = 'railways'
>>> dat_dir = "tests/osm_data"
>>> path_to_merged_shp_file = bbr.merge_shp_layers(
...     subrgn_name, lyr_name, dat_dir, verbose=True, ret_merged_shp_path=True)
Proceed to download data in the format '.shp.zip' for the following geographic (sub...
    "London"
    "Birmingham"
  to "./tests/osm_data/"
? [No]|Yes: yes
Downloading "London.osm.shp.zip" 100%|■■■■■■■■■■■| 248M/248M | 16.5MB/s | ETA:...
  Saving "London.osm.shp.zip" to "./tests/osm_data/london/" ... Done.
Downloading "Birmingham.osm.shp.zip" 100%|■■■■■■■■■■■| 79.1M/79.1M | 16.9MB/s ...
  Saving "Birmingham.osm.shp.zip" to "./tests/osm_data/birmingham/" ... Done.
Merging the following shapefiles:
  "london_railways.shp"
  "birmingham_railways.shp"
  In progress ... Done.
    Find the merged shapefile in "./tests/osm_data/lon-bir-railways/".

>>> path_to_merged_shp_file = path_to_merged_shp_file[0]
>>> os.path.relpath(path_to_merged_shp_file)
'tests\osm_data\lon-bir-railways\lon-bir-railways.shp'

>>> # Read the merged data
```

(continues on next page)

(continued from previous page)

```
>>> lon_bir_railways_shp = bbr.SHP.read_shp(path_to_merged_shp_file)
>>> lon_bir_railways_shp.head()
   osm_id  ...                                           geometry
0   30804  ...     LINESTRING (0.00486 51.62793, 0.0062 51.62927)
1  101298  ...  LINESTRING (-0.22499 51.4937, -0.22516 51.4945...
2  101486  ...  LINESTRING (-0.20555 51.51954, -0.20514 51.519...
3  101511  ...  LINESTRING (-0.2119 51.52419, -0.21081 51.5239...
4  282898  ...   LINESTRING (-0.1862 51.61592, -0.18687 51.61386)
[5 rows x 4 columns]

>>> # Delete the merged files
>>> delete_dir(os.path.dirname(path_to_merged_shp_file), verbose=True)
To delete the directory "./tests/osm_data/lon-bir-railways/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/lon-bir-railways/" ... Done.

>>> # Delete the downloaded .shp.zip data files
>>> delete_dir(list(map(os.path.dirname, bbr.downloader.data_paths)), verbose=True)
To delete the following directories:
  "./tests/osm_data/london/" (Not empty)
  "./tests/osm_data/birmingham/" (Not empty)
? [No]|Yes: yes
Deleting:
  "./tests/osm_data/london/" ... Done.
  "./tests/osm_data/birmingham/" ... Done.

>>> # Delete the example data and the test data directory
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

BBBikeReader.read_csv_xz

BBBikeReader.**read_csv_xz**(*subregion_name*, *data_dir=None*, *download=False*, *verbose=False*, ***kwargs*)

Read a compressed CSV (.csv.xz) data file of a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on BBBike free download server
- **data_dir** (*str | None*) – directory where the .csv.xz data file is located/saved; if None (default), the default directory
- **download** (*bool*) – whether to try to download the requisite data file if it does not exist, defaults to True
- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to False

Returns

tabular data of the .csv.xz file

Return type
pandas.DataFrame | None

Examples:

```
>>> from pydriosm.reader import BBBikeReader
>>> from pyhelpers.dirs import cd, delete_dir
>>> bbr = BBBikeReader()
>>> subregion_name = 'Leeds'
>>> data_dir = "tests/osm_data"
>>> leeds_csv_xz = bbr.read_csv_xz(subregion_name, data_dir, verbose=True)
The requisite data file "./tests/osm_data/leeds/Leeds.osm.csv.xz" does not exist.
>>> leeds_csv_xz = bbr.read_csv_xz(
...     subregion_name, data_dir=data_dir, verbose=True, download=True)
Downloading "Leeds.osm.csv.xz" 100%|■■■■■■■■■■■■| 4.08M/4.08M | 17.5MB/s | ETA:...
  Saving "Leeds.osm.csv.xz" to "./tests/osm_data/leeds/" ... Done.
Parsing the data ... Done.
>>> leeds_csv_xz.head()
   type      id feature  note
0  node  154915    None  None
1  node  154916    None  None
2  node  154919    None  None
3  node  154921    None  None
4  node  154922    None  None
>>> delete_dir(data_dir, verbose=True)  # Delete the downloaded .csv.xz data file
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

BBBikeReader.read_geojson_xz

BBBikeReader.**read_geojson_xz**(*subregion_name*, *data_dir=None*, *parse_geometry=False*, *download=False*, *verbose=False*, ***kwargs*)

Read a .geojson.xz data file of a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on BBBike free download server
- **data_dir** (*str | None*) – directory where the .geojson.xz data file is located/saved; if None (default), the default directory
- **parse_geometry** (*bool*) – whether to represent coordinates in a format of a geometric object, defaults to False
- **download** (*bool*) – whether to try to download the requisite data file if it does not exist, defaults to True
- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to False

Returns
tabular data of the .csv.xz file

Return type
pandas.DataFrame | None

Examples:

```
>>> from pydriosm.reader import BBBikeReader
>>> from pyhelpers.dirs import cd, delete_dir
>>> import os

>>> bbr = BBBikeReader()

>>> subrgn_name = 'Leeds'
>>> dat_dir = "tests\osm_data"

>>> leeds_geoj = bbr.read_geojson_xz(subrgn_name, dat_dir, verbose=True)
The requisite data file "./tests/osm_data/leeds/Leeds.osm.geojson.xz" does not exist.

>>> # Set `try_download=True`
>>> leeds_geoj = bbr.read_geojson_xz(subrgn_name, dat_dir, verbose=True, download=True)
Downloading "Leeds.osm.geojson.xz" 100%|■■■■■■■■■■■| 60.4M/60.4M | 20.7MB/s | ...
  Saving "Leeds.osm.geojson.xz" to "./tests/osm_data/leeds/" ... Done.
Parsing the data ... Done.
>>> leeds_geoj.head()
                                geometry                                         properties
0  {'type': 'Point', 'coordinates': [...  {'highway': 'motorway_junction', 'name': ...
1  {'type': 'Point', 'coordinates': [...  {'highway': 'motorway_junction', 'name': ...
2  {'type': 'Point', 'coordinates': [...  {'highway': 'motorway_junction', 'name': ...
3  {'type': 'Point', 'coordinates': [...  {'highway': 'motorway_junction', 'name': ...
4  {'type': 'Point', 'coordinates': [...  {'highway': 'motorway_junction', 'name': ...

>>> # Set `parse_geometry` to be True
>>> leeds_geoj_ = bbr.read_geojson_xz(
...     subrgn_name, dat_dir, parse_geometry=True, verbose=True)
Parsing "./tests/osm_data/leeds/Leeds.osm.geojson.xz" ... Done.
>>> leeds_geoj_['geometry'].head()
0    POINT (-1.5560511 53.6879848)
1       POINT (-1.34293 53.844618)
2     POINT (-1.517335 53.7499667)
3     POINT (-1.514175 53.7418444)
4     POINT (-1.516511 53.7256632)
Name: geometry, dtype: object

>>> # Delete the download directory
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

BBBikeReader.read_gpkg

BBBikeReader.**read_gpkg**(*subregion_name*, *layer_names=None*, *feature_names=None*, *data_dir=None*, *update=False*, *download=False*, *verbose=False*, *raise_error=True*, ***kwargs*)

Reads GeoPackage (.gpkg.zip) data for a specific subregion.

This method retrieves, downloads (optional), and parses OSM data in GeoPackage format. It supports selective layer loading and feature extraction, with automatic concatenation of related layers (e.g. merging multiple layers into a single key).

Parameters

- **subregion_name** (*str*) – Name of the subregion (e.g. 'West Midlands' or 'london').
- **layer_names** (*str | list | None*) – Specific OSM layer(s) to load (e.g., 'points', 'roads'); defaults to `None` (loads all available layers).
- **feature_names** (*str | list | None*) – Specific feature type(s) to extract from the loaded layers (e.g., 'residential', 'motorway'); defaults to `None`.
- **data_dir** (*str | os.PathLike | None*) – Directory where the data file is stored or will be downloaded to; defaults to `None`.
- **update** (*bool*) – Whether to update the local file by re-downloading it; defaults to `False`.
- **download** (*bool*) – Whether to download the file if it is missing locally; defaults to `False`.
- **verbose** (*bool*) – Whether to print progress messages to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise an exception if an error occurs during parsing; defaults to `True`.
- **kwargs** (*Any*) – Additional optional arguments for `pyhelpers.store.load_geopackage()`.

Returns

A GeoDataFrame (if a single layer is resolved) or a dictionary of GeoDataFrames (keyed by layer names).

Return type

geopandas.GeoDataFrame | dict | None

Examples:

```
>>> from pydriosm.reader import GeofabrikReader
>>> from pyhelpers.dirs import delete_dir
>>> reader = GeofabrikReader()
>>> # Read 'railways' layer for Rutland
>>> data_dir = "tests/osm_data"
>>> rutland_railways = reader.read_gpkg(
...     subregion_name='Rutland',
...     layer_names='railways',
...     data_dir=data_dir,
...     download=True,
...     verbose=True)
Downloading "rutland-latest-free.gpkg.zip" 100%|■■■■■■■■■■| 3.30M/3.30M | 4.5...
  Saving "rutland-latest-free.gpkg.zip" to "./tests/osm_data/rutland/" ... Done.
Parsing the data ... Done.
>>> type(rutland_railways)
dict
>>> list(rutland_railways.keys())
['railways']
>>> delete_dir(data_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

BBBikeReader.read_pbf

BBBikeReader.**read_pbf**(*subregion_name*, *data_dir=None*, *readable=False*, *expand=False*, *parse_geometry=False*, *parse_properties=False*, *parse_other_tags=False*, *update=False*, *download=False*, *pickle_it=False*, *ret_pickle_path=False*, *rm_pbf_file=False*, *chunk_size_limit=50*, *verbose=False*, ***kwargs*)

Read a PBF (.osm.pbf) data file of a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on Geofabrik free download server
- **data_dir** (*str | None*) – directory where the .osm.pbf data file is located/saved; if None, the default local directory
- **readable** (*bool*) – whether to parse each feature in the raw data, defaults to False
- **expand** (*bool*) – whether to expand dict-like data into separate columns, defaults to False
- **parse_geometry** (*bool*) – whether to represent the 'geometry' field in a shapely.geometry format, defaults to False
- **parse_properties** (*bool*) – whether to represent the 'properties' field in a tabular format, defaults to False
- **parse_other_tags** (*bool*) – whether to represent a 'other_tags' (of 'properties') in a dict format, defaults to False
- **download** (*bool*) – whether to download/update the PBF data file of the given subregion, if it is not available at the specified path, defaults to True
- **update** (*bool*) – whether to check to update pickle backup (if available), defaults to False
- **pickle_it** (*bool*) – whether to save the .pbf data as a pickle file, defaults to False
- **ret_pickle_path** (*bool*) – (when pickle_it=True) whether to return a path to the saved pickle file
- **rm_pbf_file** (*bool*) – whether to delete the downloaded .osm.pbf file, defaults to False
- **chunk_size_limit** (*int | None*) – threshold (in MB) that triggers the use of chunk parser, defaults to 50; if the size of the .osm.pbf file (in MB) is greater than chunk_size_limit, it will be parsed in a chunk-wise way
- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to False
- **kwargs** – [optional] parameters of the method BaseReader.read_pbf()

Returns

dictionary of the .osm.pbf data; when `pickle_it=True`, return a tuple of the dictionary and a path to the pickle file

Return type

dict | tuple | None

Examples:

```
>>> from pydriosm.reader import BBBikeReader
>>> from pyhelpers.dirs import delete_dir
>>> bbr = BBBikeReader()
>>> subregion_name = 'Leeds'
>>> data_dir = "tests/osm_data"
>>> leeds_pbf_raw = bbr.read_pbf(subregion_name, data_dir=data_dir, verbose=True)
The .osm.pbf file for "Leeds" is not found.
>>> leeds_pbf_raw is None
True
>>> # Set `download=True`
>>> leeds_pbf_raw = bbr.read_pbf(
...     subregion_name, data_dir=data_dir, download=True, verbose=True)
Downloading "Leeds.osm.pbf" 100%|■■■■■■■■■■| 38.1M/38.1M | 18.9MB/s | ETA: 00:00
  Saving "Leeds.osm.pbf" to "./tests/osm_data/leeds/" ... Done.
Reading "./tests/osm_data/leeds/Leeds.osm.pbf" ... Done.
>>> type(leeds_pbf_raw)
dict
>>> list(leeds_pbf_raw.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']
>>> pbf_raw_points = leeds_pbf_raw['points']
>>> type(pbf_raw_points)
list
>>> type(pbf_raw_points[0])
osgeo.ogr.Feature
>>> # (Parsing the data in this example might take up to a few minutes.)
>>> leeds_pbf_parsed = bbr.read_pbf(
...     subregion_name, data_dir=data_dir, readable=True, expand=True,
...     parse_geometry=True, parse_other_tags=True, parse_properties=True,
...     verbose=True)
Parsing "./tests/osm_data/leeds/Leeds.osm.pbf" ... Done.
>>> list(leeds_pbf_parsed.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']
>>> # Data of the 'multipolygons' layer
>>> leeds_pbf_parsed_multipolygons = leeds_pbf_parsed['multipolygons']
>>> leeds_pbf_parsed_multipolygons.shape
(481516, 26)
>>> leeds_pbf_parsed_multipolygons.head()
      id  ...                      other_tags
0  10595  ...                            None
1  10600  ...                            None
2  10601  ...                            None
3  10612  ...  {'ref:GB:uprn': '10025043089'}
4  10776  ...                            None
[5 rows x 26 columns]
>>> # Delete the example data and the test data directory
>>> delete_dir(data_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

See also

- Examples for the method `GeofabrikReader.read_pbf()`.

BBBikeReader.read_shp

BBBikeReader.**read_shp**(*subregion_name*, *layer_names=None*, *feature_names=None*, *data_dir=None*, *update=False*, *download=False*, *pickle_it=False*, *ret_pickle_path=False*, *rm_extracts=False*, *rm_shp_zip=False*, *verbose=False*, ***kwargs*)

Read a shapefile of a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on BBBike free download server
- **layer_names** (*str | list | None*) – name of a .shp layer, e.g. 'railways', or names of multiple layers; if `None` (default), all available layers
- **feature_names** (*str | list | None*) – name of a feature, e.g. 'rail', or names of multiple features; if `None` (default), all available features
- **data_dir** (*str | None*) – directory where the .shp.zip data file is located/saved; if `None`, the default directory
- **update** (*bool*) – whether to check to update pickle backup (if available), defaults to `False`
- **download** (*bool*) – whether to ask for confirmation before starting to download a file, defaults to `True`
- **pickle_it** (*bool*) – whether to save the .shp data as a pickle file, defaults to `False`
- **ret_pickle_path** (*bool*) – (when `pickle_it=True`) whether to return a path to the saved pickle file
- **rm_extracts** (*bool*) – whether to delete extracted files from the .shp.zip file, defaults to `False`
- **rm_shp_zip** (*bool*) – whether to delete the downloaded .shp.zip file, defaults to `False`
- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns

dictionary of the shapefile data, with keys and values being layer names and tabular data (in the format of geopandas.GeoDataFrame), respectively; when `pickle_it=True`, return a tuple of the dictionary and a path to the pickle file

Return type

dict | collections.OrderedDict | tuple | None

Examples:

```
>>> from pydriosm.reader import BBBikeReader
>>> from pyhelpers.dirs import delete_dir
>>> import os

>>> bbr = BBBikeReader()

>>> subrgn_name = 'Birmingham'
>>> dat_dir = "tests/osm_data"

>>> bham_shp = bbr.read_shp(
...     subregion_name=subrgn_name, data_dir=dat_dir, download=False, verbose=True)
The .shp.zip file for "Birmingham" is not found.

>>> # Set `download=True`
>>> bham_shp = bbr.read_shp(
...     subregion_name=subrgn_name, data_dir=dat_dir, download=True, verbose=True)
Downloading "Birmingham.osm.shp.zip" 100%|■■■■■■■■■■| 79.0M/79.0M | 28.5MB/s ...
  Saving "Birmingham.osm.shp.zip" to "./tests/osm_data/birmingham/" ... Done.
Extracting "./tests/osm_data/birmingham/Birmingham.osm.shp.zip"
  to "./tests/osm_data/birmingham/" ... Done.
Reading the shapefile(s) at "./tests/osm_data/birmingham/Birmingham-shp/shape/" ......
>>> type(bham_shp)
dict
>>> list(bham_shp.keys())
['buildings',
 'landuse',
 'natural',
 'places',
 'points',
 'railways',
 'roads',
 'waterways']

>>> # Data of 'railways' layer
>>> bham_railways_shp = bham_shp['railways']
>>> bham_railways_shp.shape
(3994, 5)
>>> bham_railways_shp.head()
    osm_id  ... shape_type
0      740  ...          3
1     2148  ...          3
2  2950000  ...          3
3  3491845  ...          3
4  3981454  ...          3
[5 rows x 5 columns]

>>> # Read data of 'road' layer only from the original .shp.zip file
>>> # (and delete all extracts)
>>> lyr_name = 'roads'
>>> bham_roads_shp = bbr.read_shp(
...     subregion_name=subrgn_name, layer_names=lyr_name, data_dir=dat_dir,
...     rm_extracts=True, verbose=True)
Reading "./tests/osm_data/birmingham/Birmingham-shp/shape/roads.shp" ... Done.
Deleting the extracts "./tests/osm_data/birmingham/Birmingham-shp/" ... Done.
>>> type(bham_roads_shp)
dict
```

(continues on next page)

(continued from previous page)

```
>>> list(bham_roads_shp.keys())
['roads']
>>> bham_roads_shp[lyr_name].shape
(170370, 9)
>>> bham_roads_shp[lyr_name].head()
   osm_id  ... shape_type
0      37  ...          3
1      38  ...          3
2      41  ...          3
3      42  ...          3
4      45  ...          3
[5 rows x 9 columns]

>>> # Read data of multiple layers and features from the original .shp.zip file
>>> # (and delete all extracts)
>>> lyr_names = ['railways', 'waterways']
>>> feat_names = ['rail', 'canal']
>>> bham_rw_rc_shp = bbr.read_shp(
...     subregion_name=subrgn_name, layer_names=lyr_names, feature_names=feat_names,
...     data_dir=dat_dir, rm_extracts=True, rm_shp_zip=True, verbose=True)
Extracting the following layer(s):
  'railways'
  'waterways'
    from "./tests/osm_data/birmingham/Birmingham.osm.shp.zip" ...
      to "./tests/osm_data/birmingham/" ... Done.
Reading the shapefile(s) at "./tests/osm_data/birmingham/Birmingham-shp/shape/" ......
Deleting the extracts "./tests/osm_data/birmingham/Birmingham-shp/" ... Done.
Deleting "tests/osm_data/birmingham/Birmingham.osm.shp.zip" ... Done.
>>> type(bham_rw_rc_shp)
dict
>>> list(bham_rw_rc_shp.keys())
['railways', 'waterways']

>>> # Data of the 'railways' layer
>>> bham_rw_rc_shp_railways = bham_rw_rc_shp['railways']
>>> bham_rw_rc_shp_railways[['type', 'name']].head()
   type                                          name
0  rail                                Cross-City Line
1  rail                                Cross-City Line
2  rail  Derby to Birmingham (Proof House Junction) Line
3  rail                 Birmingham to Peterborough Line
4  rail          Water Orton to Park Lane Junction Curve

>>> # Data of the 'waterways' layer
>>> bham_rw_rc_shp_waterways = bham_rw_rc_shp['waterways']
>>> bham_rw_rc_shp_waterways[['type', 'name']].head()
     type                          name
2   canal  Birmingham and Fazeley Canal
9   canal  Birmingham and Fazeley Canal
10  canal      Icknield Port Loop Canal
11  canal     Oozells Street Loop Canal
12  canal  Worcester & Birmingham Canal

>>> # Delete the example data and the test data directory
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
```

(continues on next page)

(continued from previous page)

```
Deleting "./tests/osm_data/" ... Done.
```

BBBikeReader.read_var_osm

BBBikeReader.**read_var_osm**(*meth*, *subregion_name*, *osm_file_format*, *data_dir=None*, *download=False*, *verbose=False*, *raise_error=True*, ***kwargs*)

Read data file of various formats (other than PBF and shapefile) for a geographic (sub)region.

Parameters

- **meth** (*Callable*) – name of a class method for getting (auxiliary) prepacked data
- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on a free download server
- **osm_file_format** (*str*) – format (file extension) of OSM data
- **data_dir** (*str | None*) – directory where the data file is located/saved, defaults to `None`; when `data_dir=None`, it refers to the directory specified by the corresponding downloader
- **download** (*bool*) – whether to download/update the PBF data file of the given subregion, if it is not available at the specified path, defaults to `False`
- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **kwargs** – [optional] parameters of the method specified by `meth`

Returns

data of the specified file format

Return type

pandas.DataFrame | None

See examples for the methods *BBBikeReader.read_csv_xz()* and *BBBikeReader.read_geojson_xz()*.

BBBikeReader.remove_extracts

classmethod BBBikeReader.**remove_extracts**(*path_to_extract_dir*, *verbose*)

Remove data extracts.

Parameters

- **path_to_extract_dir** (*str | os.PathLike[str]*) – pathname of the directory where data extracts are stored

- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to `False`

See examples for the methods *GeofabrikReader.read_shp()* and *BBBikeReader.read_shp()*.

BBBikeReader.validate_dtype

classmethod BBBikeReader.**validate_dtype**(*x*)

Validate the data type of the input variable.

Parameters
x (*str | list | None*) – a variable

Returns
validated input

Return type
list

Tests:

```
>>> from pydriosm.reader._base import BaseReader
>>> BaseReader.validate_dtype(x=None)
[]
>>> BaseReader.validate_dtype(x='str')
['str']
>>> BaseReader.validate_dtype(x=['str'])
['str']
```

BBBikeReader.validate_file_path

classmethod BBBikeReader.**validate_file_path**(*path_to_file*, *osm_filename=None*, *data_dir=None*)

Validate the pathname of an OSM data file.

Parameters

- **path_to_file** (*str | os.PathLike[str]*) – pathname of an OSM data file
- **osm_filename** (*str*) – filename of the OSM data file
- **data_dir** (*str | os.PathLike[str]*) – name or pathname of the data directory

Returns
validated pathname of the specified OSM data file

Return type
str

Tests:

```
>>> from pydriosm.reader._base import BaseReader
>>> import os

>>> file_path = BaseReader.validate_file_path("a\b\c.osm.pbf")
>>> file_path
'a\b\c.osm.pbf'
>>> file_path = BaseReader.validate_file_path("a\b\c.osm.pbf", "x.y.z", data_dir="a\b")
>>> os.path.relpath(file_path)
'a\b\x.y.z'
```

BBBikeReader.validate_shp_layer_names

BBBikeReader.**validate_shp_layer_names**(*layer_names_*, *extract_dir*, *shp_zip_pathname*, *subregion_name*, *osm_file_format*, *data_dir*, *update*, *download*, *verbose*)

Validate the input of layer name(s) for reading shapefiles.

Parameters

- **layer_names** (*list*) – names of shapefile layers
- **extract_dir** (*str*) – pathname of a directory to which the .shp.zip file is extracted
- **shp_zip_pathname** (*str*) – pathname of a .shp.zip file
- **subregion_name** (*str*) – name of a geographic (sub)region (case-insensitive) that is available on Geofabrik free download server
- **osm_file_format** (*str*) – format (file extension) of OSM data
- **data_dir** (*str | os.PathLike[str]*) – name or pathname of the data directory
- **update** (*bool*) – whether to check to update pickle backup (if available), defaults to False
- **download** (*bool*) – whether to download/update the PBF data file of the given subregion, if it is not available at the specified path, defaults to True
- **verbose** (*bool | int*) – whether to print relevant information in console as the function runs, defaults to False

Returns

validated shapefile layer names

Return type

list

See examples for the methods *GeofabrikReader.read_shp()* and *BBBikeReader.read_shp()*.

4.3 ios

Implement storage I/O of (parsed) OSM data extracts with PostgreSQL.

4.3.1 Manage I/O storage of parsed OSM data

1anti-flashwhitewhite

PostgresOSM([host, port, username, ...])	Implement storage I/O of OpenStreetMap data with PostgreSQL.

PostgresOSM

class pydriosm.ios.**PostgresOSM**(*host=None*, *port=None*, *username=None*, *password=None*, *database_name=None*, *data_source='Geofabrik'*, *max_tmpfile_size=None*, *data_dir=None*, ***kwargs*)

Implement storage I/O of OpenStreetMap data with PostgreSQL.

Parameters

- **host** (*str | None*) – host name/address of a PostgreSQL server, e.g. `'localhost'` or `'127.0.0.1'` (default by installation of PostgreSQL); when `host=None` (default), it is initialized as `'localhost'`
- **port** (*int | None*) – listening port used by PostgreSQL; when `port=None` (default), it is initialized as `5432` (default by installation of PostgreSQL)
- **username** (*str | None*) – username of a PostgreSQL server; when `username=None` (default), it is initialized as `'postgres'` (default by installation of PostgreSQL)
- **password** (*str | int | None*) – user password; when `password=None` (default), it is required to mannually type in the correct password to connect the PostgreSQL server
- **database_name** (*str | None*) – name of a database; when `database=None` (default), it is initialized as `'postgres'` (default by installation of PostgreSQL)
- **confirm_db_creation** – whether to prompt a confirmation before creating a new database (if the specified database does not exist), defaults to `False`
- **data_source** (*str*) – name of data source, defaults to `'Geofabrik'`; options include `{'Geofabrik', 'BBBike'}`
- **max_tmpfile_size** (*int | None*) – defaults to `None`, see also the function pyhelpers.settings.gdal_configurations()
- **data_dir** (*str | None*) – directory where the data file is located/saved, defaults to `None`; when `data_dir=None`, it should be the same as the directory specified by the corresponding *downloader*/*reader*
- **kwargs** – [optional] parameters of the class pyhelpers.sql.PostgreSQL

Variables

data_source (*str*) – name of data sources, options include {'Geofabrik', 'BBBike'}

Examples:

```
>>> from pydriosm.ios import PostgresOSM

>>> osmdb = PostgresOSM(database_name='osmdb_test', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> osmdb.data_source
'Geofabrik'
>>> type(osmdb.downloader)
pydriosm.downloader._wrapper.Downloader
>>> type(osmdb.reader)
pydriosm.reader._wrapper.Reader

>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> type(osmdb.downloader)
pydriosm.downloader._wrapper.Downloader
>>> type(osmdb.reader)
pydriosm.reader._wrapper.Reader

>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

Attributes

1anti-flashwhitewhite

BUILTIN_SCHEMAS	Built-in schemas of PostgreSQL.
DATA_SOURCES	Names of the data sources.
DATA_TYPES	Specify a data-type dictionary for data or columns corresponding to Pandas.
DEFAULT_DATABASE	Default database name (usually created during PostgreSQL installation).
DEFAULT_DIALECT	Default dialect used by SQLAlchemy to communicate with PostgreSQL; see also [DBMS-PS-1].
DEFAULT_DRIVER	Default name of the database driver.
DEFAULT_HOST	Default host name/address (typically *localhost*).
DEFAULT_PORT	Default listening port used by PostgreSQL.
DEFAULT_SCHEMA	Default schema name created during PostgreSQL installation.

continues on next page

Table 19 – continued from previous page

DEFAULT_USERNAME	Default username.
LONG_NAME	Name of the current property *downloader*.
NAME	Name of the current property *downloader*.
URL	Homepage URL of data resource for current property *downloader*.
downloader	Instance of either the class *GeofabrikDownloader* or *BBBikeDownloader*, depending on the specified `data_source` for creating an instance of the class *PostgresOSM*.
reader	Instance of either *GeofabrikReader* or *BBBikeReader*, depending on the specified `data_source` for creating an instance of the calss *PostgresOSM*.

PostgresOSM.BUILTIN_SCHEMAS

PostgresOSM.**BUILTIN_SCHEMAS: set = {'information_schema'}**

Built-in schemas of PostgreSQL.

PostgresOSM.DATA_SOURCES

PostgresOSM.**DATA_SOURCES: list = ['Geofabrik', 'BBBike']**

Names of the data sources.

PostgresOSM.DATA_TYPES

PostgresOSM.**DATA_TYPES: dict = {'bigint': <class 'numpy.int64'>, 'json': <class 'str'>, 'text': <class 'str'>}**

Specify a data-type dictionary for data or columns corresponding to Pandas.

PostgresOSM.DEFAULT_DATABASE

PostgresOSM.**DEFAULT_DATABASE: str = 'postgres'**

Default database name (usually created during PostgreSQL installation).

PostgresOSM.DEFAULT_DIALECT

PostgresOSM.**DEFAULT_DIALECT: str = 'postgresql'**

Default dialect used by SQLAlchemy to communicate with PostgreSQL; see also [DBMS-PS-1].

PostgresOSM.DEFAULT_DRIVER

PostgresOSM.**DEFAULT_DRIVER: str = 'psycopg2'**

Default name of the database driver.

PostgresOSM.DEFAULT_HOST

PostgresOSM.**DEFAULT_HOST: str = 'localhost'**

Default host name/address (typically *localhost*).

PostgresOSM.DEFAULT_PORT

PostgresOSM.**DEFAULT_PORT: str = 5432**

Default listening port used by PostgreSQL.

PostgresOSM.DEFAULT_SCHEMA

PostgresOSM.**DEFAULT_SCHEMA: str = 'public'**

Default schema name created during PostgreSQL installation.

PostgresOSM.DEFAULT_USERNAME

PostgresOSM.**DEFAULT_USERNAME: str = 'postgres'**

Default username.

PostgresOSM.LONG_NAME

property PostgresOSM.**LONG_NAME**

Name of the current property *downloader*.

Examples:

```
>>> from pydriosm.ios._base import BaseIOS

>>> osmdb = BaseIOS(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> osmdb.data_source
'Geofabrik'
>>> osmdb.LONG_NAME
'Geofabrik OpenStreetMap data extracts'

>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> osmdb.LONG_NAME
'BBBike exports of OpenStreetMap data'

>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
```

(continues on next page)

(continued from previous page)

```
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

PostgresOSM.NAME

property PostgresOSM.**NAME**

Name of the current property *downloader*.

Examples:

```
>>> from pydriosm.ios._base import BaseIOS

>>> osmdb = BaseIOS(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> osmdb.data_source
'Geofabrik'
>>> osmdb.NAME
'Geofabrik OpenStreetMap data extracts'

>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> osmdb.NAME
'BBBike exports of OpenStreetMap data'

>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

PostgresOSM.URL

property PostgresOSM.**URL**

Homepage URL of data resource for current property *downloader*.

Examples:

```
>>> from pydriosm.ios._base import BaseIOS

>>> osmdb = BaseIOS(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> osmdb.URL
'https://download.geofabrik.de/'

>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> osmdb.URL
```

(continues on next page)

(continued from previous page)

```
'https://download.bbbike.org/osm/bbbike/'

>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

PostgresOSM.downloader

property PostgresOSM.**downloader**

Instance of either the class *GeofabrikDownloader* or *BBBikeDownloader*, depending on the specified `data_source` for creating an instance of the class *PostgresOSM*.

Examples:

```
>>> from pydriosm.ios._base import BaseIOS

>>> osmdb = BaseIOS(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> osmdb.data_source
'Geofabrik'
>>> type(osmdb.downloader)
pydriosm.downloader.GeofabrikDownloader

>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> type(osmdb.downloader)
pydriosm.downloader.BBBikeDownloader

>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

PostgresOSM.reader

property PostgresOSM.**reader**

Instance of either *GeofabrikReader* or *BBBikeReader*, depending on the specified `data_source` for creating an instance of the calss *PostgresOSM*.

Examples:

```
>>> from pydriosm.ios._base import BaseIOS

>>> osmdb = BaseIOS(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.
```

(continues on next page)

(continued from previous page)

```
>>> type(osmdb.reader)
pydriosm.reader.GeofabrikReader

>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> type(osmdb.reader)
pydriosm.reader.BBBikeReader

>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

Methods

1anti-flashwhitewhite

add_primary_keys(primary_keys, table_name[, ...])	Adds a primary key or multiple primary keys to a table.
alter_table_schema(table_name, schema_name, ...)	Moves a table from one schema to another within the currently-connected database.
connect_database([database_name, verbose, ...])	Establishes a connection to a database.
create_database(database_name[, verbose])	Creates a database.
create_schema(schema_name[, verbose, ...])	Creates a schema.
create_table(table_name, column_specs[, ...])	Creates a table.
database_exists([database_name])	Checks if a specified database exists.
decode_pbf_layer(layer_dat[, decode_geojson])	Process raw data of a PBF layer retrieved from database.
disconnect_all_others()	Terminates connections to all databases except the currently-connected one.
disconnect_database([database_name, ...])	Disconnects from a database.
drop_database([database_name, ...])	Deletes/drops a database.
drop_schema(schema_names[, ...])	Deletes/drops one or multiple schemas.
drop_subregion_tables(subregion_names[, ...])	Delete all or specific schemas/layers of subregion data from the database being connected.
drop_table(table_name[, schema_name, ...])	Deletes/drops a specified table.
fetch_data(subregion_name[, layer_names, ...])	Fetch OSM data (of one or multiple layers) of a geographic (sub)region.
get_column_dtype(table_name[, column_names, ...])	Retrieves information about data types of all or specific columns of a table.
get_column_info(table_name[, schema_name, ...])	Retrieves information about columns of a table.

continues on next page

Table 20 – continued from previous page

get_column_names(table_name[, schema_name])	Retrieves column names of a table.
get_database_names([names_only])	Retrieves the names of all existing databases.
get_database_size([database_name])	Retrieves the size of a database.
get_primary_keys(table_name[, schema_name, ...])	Retrieves the primary keys of a table.
get_schema_info([names_only, include_all, ...])	Retrieves information about existing schemas.
get_table_column_info(subregion_name, layer_name)	Get information about columns of a specific schema and table data for a geographic (sub)region.
get_table_name(subregion_name[, ...])	Get the default table name for a specific geographic (sub)region.
get_table_names([schema_name, verbose])	Retrieves the names of all tables in a schema.
import_data(data, table_name[, schema_name, ...])	Imports tabular data into the database.
import_osm_data(osm_data, table_name[, ...])	Import OSM data into a database.
import_osm_layer(layer_data, table_name, ...)	Import one layer of OSM data into a table.
import_osm_pbf(subregion_names[, data_dir, ...])	Import data of geographic (sub)region(s) that do not have (sub-)subregions into a database.
null_text_to_empty_string(table_name[, ...])	Converts null values (in text columns) to empty strings.
postprocess_pdf_layer(layer_dat[, ...])	Post-process the data of a specific layer.
psql_insert_copy(sql_table, sql_db_engine, ...)	Callable function using *PostgreSQL* `COPY` clause for executing data insertion.
read_sql_query(sql_query[, method, ...])	Reads table data by executing a SQL query (recommended for large tables).
read_table(table_name[, schema_name, ...])	Reads data from a specified table.
schema_exists(schema_name)	Checks if a schema exists.
subregion_table_exists(subregion_name, ...)	Check if a table (for a geographic (sub)region) exists.
table_exists(table_name[, schema_name])	Checks if a table exists.
validate_column_names(table_name[, ...])	Validates column names for a query statement.

PostgresOSM.add_primary_keys

PostgresOSM.**add_primary_keys**(*primary_keys*, *table_name*, *schema_name=None*)

Adds a primary key or multiple primary keys to a table.

Parameters

- **primary_keys** (*str | list | None*) – (List of) primary key(s) to be added.
- **table_name** (*str*) – Name of the table.

- **schema_name** (*str | None*) – Name of the schema; defaults to None.

See also

- Examples for the method get_primary_keys().

PostgresOSM.alter_table_schema

PostgresOSM.**alter_table_schema**(*table_name*, *schema_name*, *new_schema_name*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Moves a table from one schema to another within the currently-connected database.

Parameters

- **table_name** (*str*) – Name of the table.
- **schema_name** (*str*) – Name of the current schema containing the table.
- **new_schema_name** (*str*) – Name of the new schema to move the table to.
- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to True.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> # Create a new table named "test_table" in the schema "testdb"
>>> new_tbl_name = 'test_table'
>>> col_spec = 'col_name_1 INT, col_name_2 TEXT'
>>> testdb.create_table(table_name=new_tbl_name, column_specs=col_spec, verbose=True)
Creating a table: "public"."test_table" ... Done.
>>> # Create a new schema "test_schema"
>>> testdb.create_schema(schema_name='test_schema', verbose=True)
Creating a schema: "test_schema" ... Done.
>>> # Move the table "public"."test_table" to the schema "test_schema"
>>> testdb.alter_table_schema(
...     table_name='test_table', schema_name='public', new_schema_name='test_schema',
...     verbose=True)
To move the table "test_table" from the schema "public" to "test_schema"
? [No]|Yes: yes
Moving "public"."test_table" to "test_schema" ... Done.
>>> tbl_names = testdb.get_table_names(schema_name='test_schema')
>>> tbl_names
{'test_schema': ['test_table']}
>>> # Delete the database "testdb"
```

(continues on next page)

(continued from previous page)

```
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.connect_database

PostgresOSM.**connect_database**(*database_name=None*, *verbose=False*, *raise_error=False*)

Establishes a connection to a database.

Parameters

- **database_name** (*str | None*) – Name of the database. If database_name=None (default), the database name must be input manually.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.connect_database(verbose=True)
Being connected with postgres:***@localhost:5432/testdb.
>>> testdb.connect_database(database_name='postgres', verbose=True)
Connecting postgres:***@localhost:5432/postgres ... Successfully.
>>> testdb.database_name
'postgres'
>>> testdb.connect_database(database_name='testdb', verbose=True)
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
>>> testdb.database_name
'postgres'
```

PostgresOSM.create_database

PostgresOSM.**create_database**(*database_name*, *verbose=False*)

Creates a database.

Parameters

- **database_name** (*str*) – Name of the database to be created.

- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.create_database(database_name='testdb1', verbose=True)
Creating a database: "testdb1" ... Done.
>>> testdb.database_name
'testdb1'
>>> # Delete the database "testdb1"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb1" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb1" ... Done.
>>> testdb.database_name
'postgres'
>>> # Delete the database "testdb"
>>> testdb.drop_database(database_name='testdb', verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
>>> testdb.database_name
'postgres'
```

PostgresOSM.create_schema

`PostgresOSM.`**`create_schema`**(*schema_name*, *verbose=False*, *raise_error=False*)

Creates a schema.

Parameters

- **schema_name** (*str*) – Name of the schema to be created.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> test_schema_name = 'test_schema'
>>> testdb.create_schema(schema_name=test_schema_name, verbose=True)
Creating a schema: "test_schema" ... Done.
>>> testdb.schema_exists(schema_name=test_schema_name)
```

(continues on next page)

(continued from previous page)

```
True
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.create_table

PostgresOSM.**create_table**(*table_name*, *column_specs*, *schema_name=None*, *verbose=False*, *raise_error=False*)

Creates a table.

Parameters

- **table_name** (*str*) – Name of the table to be created.
- **column_specs** (*str*) – Specifications for each column of the table.
- **schema_name** (*str | None*) – Name of the schema; if `schema_name=None` (default), it defaults to `DEFAULT_SCHEMA` (i.e. `'public'`).
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> # Create a new table named 'test_table'
>>> tbl_name = 'test_table'
>>> col_spec = 'col_name_1 INT, col_name_2 TEXT'
>>> testdb.create_table(table_name=tbl_name, column_specs=col_spec, verbose=True)
Creating a table "public"."test_table" ... Done.
>>> testdb.table_exists(table_name=tbl_name)
True
>>> # Get information about all columns of the table "public"."test_table"
>>> test_tbl_col_info = testdb.get_column_info(table_name=tbl_name, as_dict=False)
>>> test_tbl_col_info.head()
                    column_0    column_1
table_catalog         testdb      testdb
table_schema          public      public
table_name        test_table  test_table
column_name       col_name_1  col_name_2
ordinal_position           1           2
>>> # Get column names of the table "public"."test_table"
>>> test_tbl_col_names = testdb.get_column_names(table_name=tbl_name)
>>> test_tbl_col_names
['col_name_1', 'col_name_2']
>>> # Get data types of all columns of the table
```

(continues on next page)

(continued from previous page)

```
>>> test_tbl_dtypes = testdb.get_column_dtype(table_name=tbl_name)
>>> test_tbl_dtypes
{'col_name_1': 'integer', 'col_name_2': 'text'}
>>> testdb.validate_column_names(table_name=tbl_name)
'"col_name_1", "col_name_2"'
>>> # Drop the table "public"."test_table"
>>> testdb.drop_table(table_name=tbl_name, verbose=True)
To drop the table "public"."test_table" from postgres:***@localhost:5432/testdb
? [No]|Yes: yes
Dropping "public"."test_table" ... Done.
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.database_exists

PostgresOSM.**database_exists**(*database_name=None*)

Checks if a specified database exists.

Parameters

database_name (*str | None*) – Name of the database to check; defaults to None.

Returns

True if the database exists, otherwise False.

Return type

bool

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> # Check whether the database "testdb" exists now
>>> testdb.database_exists(database_name='testdb')  # testdb.database_exists()
True
>>> testdb.database_name
'testdb'
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
>>> # Check again whether the database "testdb" still exists now
>>> testdb.database_exists(database_name='testdb')
False
>>> testdb.database_name
'postgres'
```

PostgresOSM.decode_pbf_layer

PostgresOSM.**decode_pbf_layer**(*layer_dat*, *decode_geojson=True*)

Process raw data of a PBF layer retrieved from database.

> **See also**
>
> - Examples of the method *fetch_data()*.

PostgresOSM.disconnect_all_others

PostgresOSM.**disconnect_all_others**()

Terminates connections to all databases except the currently-connected one.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.disconnect_all_others()
>>> testdb.database_name
'testdb'
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.disconnect_database

PostgresOSM.**disconnect_database**(*database_name=None*, *verbose=False*, *raise_error=False*)

Disconnects from a database.

See also [DBMS-PS-DD-1].

Parameters

- **database_name** (*str | None*) – Name of the database to disconnect from. If `database_name=None` (default), disconnect from the current database.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Creating a database: "testdb" ... Done.
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.disconnect_database()
>>> testdb.database_name
'postgres'
>>> # Delete the database "testdb"
>>> testdb.drop_database(database_name='testdb', verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.drop_database

PostgresOSM.**drop_database**(*database_name=None*, *confirmation_required=True*, *verbose=False*)

Deletes/drops a database.

Parameters

- **database_name** (*str | None*) – Name of the database to be dropped. If None (default), drop the currently connected database.
- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to True.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
>>> testdb.database_exists(database_name='testdb')
False
>>> testdb.drop_database(database_name='testdb', verbose=True)
The database "testdb" does not exist.
>>> testdb.database_name
'postgres'
```

PostgresOSM.drop_schema

PostgresOSM.**drop_schema**(*schema_names*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Deletes/drops one or multiple schemas.

Parameters

- **schema_names** (*str | Iterable[str]*) – Name of one schema or names of multiple schemas to be dropped.
- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to True.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> new_schema_names = ['points', 'lines', 'polygons']
>>> for new_schema in new_schema_names:
...     testdb.create_schema(new_schema, verbose=True)
Creating a schema: "points" ... Done.
Creating a schema: "lines" ... Done.
Creating a schema: "polygons" ... Done.
>>> new_schema_names_ = ['test_schema']
>>> testdb.drop_schema(new_schema_names + new_schema_names_, verbose=True)
To drop the following schemas from postgres:***@localhost:5432/testdb:
  "points"
  "lines"
  "polygons"
  "test_schema"
? [No]|Yes: yes
Dropping ...
  "points" ... Done.
  "lines" ... Done.
  "polygons" ... Done.
  "test_schema" (does not exist.)
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.drop_subregion_tables

PostgresOSM.**drop_subregion_tables**(*subregion_names*, *schema_names=None*, *table_named_as_subregion=False*, *schema_named_as_layer=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Delete all or specific schemas/layers of subregion data from the database being connected.

Parameters

- **subregion_names** (*str | list*) – name of table for a subregion (or name of a subregion)
- **schema_names** (*str | list | None*) – names of schemas for each layer of the PBF data, if None (default), the default layer names as schema names
- **table_named_as_subregion** (*bool*) – whether to use subregion name as a table name, defaults to False
- **schema_named_as_layer** (*bool*) – whether a schema is named as a layer name, defaults to False
- **confirmation_required** (*bool*) – whether to ask for confirmation to proceed, defaults to True
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to False
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Examples:

```
>>> from pydriosm.ios import PostgresOSM
>>> from pyhelpers.dirs import delete_dir

>>> osmdb = PostgresOSM(database_name='osmdb_test', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.
```

Import example data into the database:

```
>>> dat_dir = "tests/osm_data"  # Specify a temporary data directory

>>> # Import PBF data of 'Rutland' and 'Isle of Wight'
>>> subrgn_name_1 = ['Rutland', 'Isle of Wight']
>>> osmdb.import_osm_pbf(
...     subrgn_name_1, data_dir=dat_dir, expand=True, parse_geometry=True,
...     parse_properties=True, parse_other_tags=True, verbose=True)
Proceed to import .osm.pbf data of the following geographic (sub)region(s):
    "Rutland"
    "Isle of Wight"
  into postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" 100%|■■■■■■■■■■■■| 1.89M/1.89M | 6.46MB/s ...
  Saving "rutland-latest.osm.pbf" to "./tests/osm_data/rutland/" ... Done.
Reading "tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
Importing the data into table "Rutland" ...
  "points" ... Done. (<total of rows> features)
  "lines" ... Done. (<total of rows> features)
  "multilinestrings" ... Done. (<total of rows> features)
  "multipolygons" ... Done. (<total of rows> features)
  "other_relations" ... Done. (<total of rows> features)
Downloading "isle-of-wight-latest.osm.pbf" 100%|■■■■■■■■■■■■| 8.83M/8.83M | 16....
```

(continues on next page)

(continued from previous page)

```
  Saving "isle-of-wight-latest.osm.pbf" to "./tests/osm_data/isle-of-wight/" ... Done.
Reading "tests/osm_data/isle-of-wight/isle-of-wight-latest.osm.pbf" ... Done.
Importing the data into table "Isle of Wight" ...
  "points" ... Done. (<total of rows> features)
  "lines" ... Done. (<total of rows> features)
  "multilinestrings" ... Done. (<total of rows> features)
  "multipolygons" ... Done. (<total of rows> features)
  "other_relations" ... Done. (<total of rows> features)

>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> subrgn_name_2 = 'London'

>>> # An alternative way to import the shapefile data of 'London'
>>> london_shp = osmdb.reader.read_shp(
...     subrgn_name_2, data_dir=dat_dir, rm_extracts=True, download=True, verbose=True)
Downloading "London.osm.shp.zip" 100%|■■■■■■■■■■■| 245M/245M | 6.41MB/s | ETA:...
  Saving "London.osm.shp.zip" to "./tests/osm_data/london/" ... Done.
Extracting "./tests/osm_data/london/London.osm.shp.zip"
  to "./tests/osm_data/london/" ... Done.
Reading the shapefile(s) at "./tests/osm_data/london/London-shp/shape/" ... Done.
Deleting the extracts "./tests/osm_data/london/London-shp/" ... Done.
>>> osmdb.import_osm_data(london_shp, table_name=subrgn_name_2, verbose=True)
Proceed to import data into table "London" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
  "buildings" ... Done. (<total of rows> features)
  "landuse" ... Done. (<total of rows> features)
  "natural" ... Done. (<total of rows> features)
  "places" ... Done. (<total of rows> features)
  "points" ... Done. (<total of rows> features)
  "railways" ... Done. (<total of rows> features)
  "roads" ... Done. (<total of rows> features)
  "waterways" ... Done. (<total of rows> features)
```

Delete data of 'Rutland':

```
>>> subrgn_name = 'Rutland'

>>> # Delete data of Rutland under the schemas 'buildings' and 'landuse'
>>> lyr_name = ['buildings', 'landuse']
>>> osmdb.drop_subregion_tables(subrgn_name, lyr_name, verbose=True)
None of the data exists.

>>> # Delete 'points' layer data of Rutland
>>> lyr_name = 'points'
>>> osmdb.drop_subregion_tables(subrgn_name, lyr_name, verbose=True)
Proceed to drop table "points"."Rutland"
  from postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Dropping the table ...
  "points"."Rutland" ... Done.

>>> # Delete all available tables of Rutland
>>> osmdb.drop_subregion_tables(subrgn_name, verbose=True)
Proceed to drop table from postgres:***@localhost:5432/osmdb_test: "Rutland"
```

(continues on next page)

(continued from previous page)

```
  under the schemas:
    "lines"
    "multilinestrings"
    "multipolygons"
    "other_relations"
? [No]|Yes: yes
Dropping the tables ...
  "lines"."Rutland" ... Done.
  "multilinestrings"."Rutland" ... Done.
  "multipolygons"."Rutland" ... Done.
  "other_relations"."Rutland" ... Done.
```

Delete 'buildings' and 'points' data of London and Isle of Wight:

```
>>> # Delete 'buildings' and 'points' layers of London and Isle of Wight
>>> subrgn_names = ['London', 'Isle of Wight']
>>> lyr_names = ['buildings', 'points']
>>> osmdb.drop_subregion_tables(subrgn_names, schema_names=lyr_names, verbose=True)
Proceed to drop tables from postgres:***@localhost:5432/osmdb_test:
    "Isle of Wight"
    "London"
  under the schemas:
    "points"
    "buildings"
? [No]|Yes: yes
Dropping the tables ...
  "points"."Isle of Wight" ... Done.
  "points"."London" ... Done.
  "buildings"."London" ... Done.

>>> # Delete the rest of the data of London and Isle of Wight
>>> osmdb.drop_subregion_tables(subrgn_names, verbose=True)
Proceed to drop tables from postgres:***@localhost:5432/osmdb_test:
    "Isle of Wight"
    "London"
  under the schemas:
    "railways"
    "landuse"
    "other_relations"
    "lines"
    "multilinestrings"
    "waterways"
    "roads"
    "multipolygons"
    "natural"
    "places"
? [No]|Yes: yes
Dropping the tables ...
  "railways"."London" ... Done.
  "landuse"."London" ... Done.
  "other_relations"."Isle of Wight" ... Done.
  "lines"."Isle of Wight" ... Done.
  "multilinestrings"."Isle of Wight" ... Done.
  "waterways"."London" ... Done.
  "roads"."London" ... Done.
  "multipolygons"."Isle of Wight" ... Done.
```

(continues on next page)

(continued from previous page)

```
  "natural"."London" ... Done.
  "places"."London" ... Done.
```

Delete the test database and downloaded data files:

```
>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
Procced to drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.

>>> # Delete the downloaded data files
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

PostgresOSM.drop_table

PostgresOSM.**drop_table**(*table_name*, *schema_name=None*, *confirmation_required=True*, *verbose=False*, *indent=0*, *raise_error=False*)

Deletes/drops a specified table.

Parameters

- **table_name** (*str*) – Name of the table to be deleted.
- **schema_name** (*str | None*) – Name of the schema; if schema_name=None, it defaults to DEFAULT_SCHEMA (i.e., 'public').
- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to True.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **indent** (*int | str*) – Indentation level; if an integer, represents the number of spaces; If a string, used as the indentation character (e.g. '\t'); defaults to 0 (no space).
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

See also

- Examples for the method `create_table()`.

PostgresOSM.fetch_data

`PostgresOSM.`**`fetch_data`**(*subregion_name*, *layer_names=None*, *chunk_size=None*, *method='tempfile'*, *max_size_spooled=1*, *decode_geojson=True*, *sort_by='id'*, *table_named_as_subregion=False*, *schema_named_as_layer=False*, *verbose=False*, *raise_error=False*, ***kwargs*)

Fetch OSM data (of one or multiple layers) of a geographic (sub)region.

See also [ROP-1].

Parameters

- **`subregion_name`** (*`str`*) – name of a geographic (sub)region (or the corresponding table)
- **`layer_names`** (*`list | None`*) – names of schemas for each layer of the PBF data, if `None` (default), the default layer names as schema names
- **`chunk_size`** (*`int | None`*) – the number of rows in each batch to be written at a time, defaults to `None`
- **`method`** (*`str | None`*) – method to be used for buffering temporary data, defaults to `'tempfile'`
- **`max_size_spooled`** (*`int , float`*) – see pyhelpers.sql.PostgreSQL.read_sql_query(), defaults to `1` (in GB)
- **`decode_geojson`** (*`bool`*) – whether to decode string GeoJSON data, defaults to `True`
- **`sort_by`** (*`str | list`*) – column name(s) by which the data (fetched from PostgreSQL) is sorted, defaults to `'id'`
- **`table_named_as_subregion`** (*`bool`*) – whether to use subregion name as a table name, defaults to `False`
- **`schema_named_as_layer`** (*`bool`*) – whether a schema is named as a layer name, defaults to `False`
- **`verbose`** (*`bool | int`*) – whether to print relevant information in console, defaults to `False`
- **`raise_error`** (*`bool`*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

PBF (.osm.pbf) data

Return type
dict

Examples:

```
>>> from pydriosm.ios import PostgresOSM
>>> from pyhelpers.dirs import delete_dir

>>> osmdb = PostgresOSM(database_name='osmdb_test', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> subrgn_name = 'Rutland'  # name of a subregion
>>> dat_dir = "tests/osm_data"  # name of a data directory where the subregion data is

>>> # Import PBF data of Rutland
>>> osmdb.import_osm_pbf(subrgn_name, data_dir=dat_dir, verbose=True)
Proceed to import .osm.pbf data of the following geographic (sub)region(s):
    "Rutland"
  into postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" 100%|■■■■■■■■■■■| 1.89M/1.89M | 6.46MB/s ...
  Saving "rutland-latest.osm.pbf" to "./tests/osm_data/rutland/" ... Done.
Reading "tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
Importing the data into the table "Rutland" ...
  "points" ... Done. (6473 features)
  "lines" ... Done. (11057 features)
  "multilinestrings" ... Done. (71 features)
  "multipolygons" ... Done. (9423 features)
  "other_relations" ... Done. (33 features)

>>> # Import shapefile data of Rutland
>>> rutland_shp = osmdb.reader.read_shp(
...     subrgn_name, data_dir=dat_dir, rm_extracts=True, verbose=True)
>>> rutland_shp is None
True

>>> # Import geopackage data of Rutland
>>> rutland_gpkg = osmdb.reader.read_gpkg(
...     subrgn_name, data_dir=dat_dir, download=True, verbose=True)
Downloading "rutland-latest-free.gpkg.zip" 100%|■■■■■■■■■■■| 3.30M/3.30M | 7.5...
  Saving "rutland-latest-free.gpkg.zip" to "./tests/osm_data/rutland/" ... Done.
Parsing the data ... Done.
>>> osmdb.import_osm_data(rutland_gpkg, table_name=subrgn_name, verbose=True)
Proceed to import data into the table "Rutland" at postgres:***@localhost:5432/osmd...
? [No]|Yes: yes
Importing the data ...
  "traffic" ... Done. (557 features)
  "places" ... Done. (301 features)
  "pois" ... Done. (1081 features)
  "transport" ... Done. (65 features)
  "pofw" ... Done. (65 features)
  "natural" ... Done. (665 features)
  "railways" ... Done. (141 features)
  "roads" ... Done. (7315 features)
  "waterways" ... Done. (379 features)
  "protected_areas" ... Done. (20 features)
```

(continues on next page)

(continued from previous page)

```
  "water" ... Done. (233 features)
  "landuse" ... Done. (2461 features)
  "buildings" ... Done. (5706 features)
  "adminareas" ... Done. (58 features)

>>> # Retrieve the data of specific layers
>>> lyr_names = ['points', 'multipolygons']
>>> rutland_data_ = osmdb.fetch_data(subrgn_name, lyr_names, verbose=True)
Fetching the data of "Rutland" ...
  "points" ... Done.
  "multipolygons" ... Done.
>>> type(rutland_data_)
dict
>>> list(rutland_data_.keys())
['points', 'multipolygons']

>>> # Data of the 'points' layer
>>> rutland_points = rutland_data_['points']
>>> rutland_points.head()
                                              points
0  {'type': 'Feature', 'geometry': {'type': 'Poin...
1  {'type': 'Feature', 'geometry': {'type': 'Poin...
2  {'type': 'Feature', 'geometry': {'type': 'Poin...
3  {'type': 'Feature', 'geometry': {'type': 'Poin...
4  {'type': 'Feature', 'geometry': {'type': 'Poin...

>>> # Retrieve the data of all the layers from the database
>>> rutland_data = osmdb.fetch_data(subrgn_name, layer_names=None, verbose=True)
Fetching the data of "Rutland" ...
  "points" ... Done.
  "lines" ... Done.
  "multilinestrings" ... Done.
  "multipolygons" ... Done.
  "other_relations" ... Done.
  "adminareas" ... Done.
  "buildings" ... Done.
  "landuse" ... Done.
  "natural" ... Done.
  "places" ... Done.
  "pofw" ... Done.
  "pois" ... Done.
  "protected_areas" ... Done.
  "railways" ... Done.
  "roads" ... Done.
  "traffic" ... Done.
  "transport" ... Done.
  "water" ... Done.
  "waterways" ... Done.
>>> type(rutland_data)
dict
>>> list(rutland_data.keys())
['points',
 'lines',
 'multilinestrings',
 'multipolygons',
 'other_relations',
 'adminareas',
```

(continues on next page)

(continued from previous page)

```
 'buildings',
 'landuse',
 'natural',
 'places',
 'pofw',
 'pois',
 'protected_areas',
 'railways',
 'roads',
 'traffic',
 'transport',
 'water',
 'waterways']

>>> # Data of the 'waterways' layer
>>> rutland_waterways = rutland_data['waterways']
>>> rutland_waterways.head()
    osm_id  ...                                           geometry
0  3701346  ...  LINESTRING (-0.7536654 52.6495358, -0.7536236 ...
1  3701347  ...  LINESTRING (-0.7948821 52.6569468, -0.7946128 ...
2  3707149  ...  LINESTRING (-0.7262381 52.6790459, -0.7258244 ...
3  3707303  ...  LINESTRING (-0.7213277 52.6765954, -0.7206778 ...
4  4470795  ...  LINESTRING (-0.4995349 52.6418825, -0.4984075 ...
[5 rows x 6 columns]
```

Delete the test database and downloaded data files:

```
>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
Drop the database "osmdb_test" from postgres:***@localhost:5432?
 [No]|Yes: yes
Dropping "osmdb_test" ... Done.

>>> # Delete the downloaded data files
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

See also

- More details of the above data can be found in the examples for *import_osm_data()*.
- Similar examples about *fetching data from the database* are available in *Quick start*.

PostgresOSM.get_column_dtype

PostgresOSM.**get_column_dtype**(*table_name*, *column_names=None*, *schema_name=None*)

Retrieves information about data types of all or specific columns of a table.

Parameters

- **table_name** (*str*) – Name of the table.
- **column_names** (*str | list | None*) – (List of) column name(s); if `column_names=None` (default), data types of all available columns are retrieved.
- **schema_name** – Name of the schema; if `schema_name=None` (default), it defaults to `DEFAULT_SCHEMA` (i.e. `'public'`).

Returns
Data types of all or specific columns of a table.

Return type
dict | None

See also

- Examples for the method `create_table()`.

PostgresOSM.get_column_info

`PostgresOSM.`**`get_column_info`**(*table_name*, *schema_name=None*, *as_dict=True*)

Retrieves information about columns of a table.

Parameters

- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema; if `schema_name=None` (default), it defaults to `DEFAULT_SCHEMA` (i.e. `'public'`).
- **as_dict** (*bool*) – Whether to return the column information as a dictionary; defaults to `True`.

Returns
Information about all columns of the given table.

Return type
pandas.DataFrame | dict

See also

- Examples for the method `create_table()`.

PostgresOSM.get_column_names

`PostgresOSM.`**`get_column_names`**(*table_name*, *schema_name=None*)

Retrieves column names of a table.

Parameters

- **table_name** (*str*) – Name of the table to retrieve column names from.

- **schema_name** (*str | None*) – Name of the schema where the table is located; defaults to None.

Returns
List of column names in the specified table in the currently-connected database.

Return type
list

See also

- Examples for the method create_table().

PostgresOSM.get_database_names

PostgresOSM.**get_database_names**(*names_only=True*)

Retrieves the names of all existing databases.

Parameters
names_only (*bool*) – Whether to return only the names of the databases; defaults to True.

Returns
A list of database names if names_only is True; otherwise, a dataframe containing detailed information.

Return type
list | pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> # Connect the default database 'postgres'
>>> postgres = PostgreSQL(verbose=True)
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/postgres ... Successfully.
>>> isinstance(postgres.get_database_names(), list)
True
>>> 'postgres' in postgres.get_database_names()
True
```

PostgresOSM.get_database_size

PostgresOSM.**get_database_size**(*database_name=None*)

Retrieves the size of a database.

Parameters
database_name (*str | None*) – Name of the database. If database_name=None (default), it retrieves the size of the currently-connected database.

Returns
Size of the database.

Return type
str | None

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.get_database_size()
'7900 kB'
>>> testdb.DEFAULT_DATABASE
'postgres'
>>> testdb.get_database_size(database_name=testdb.DEFAULT_DATABASE)
'7828 kB'
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.get_primary_keys

PostgresOSM.**get_primary_keys**(*table_name*, *schema_name=None*, *names_only=True*)

Retrieves the primary keys of a table.

Parameters

- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema; defaults to None.
- **names_only** (*bool*) – Whether to return only the names of the primary keys; defaults to True.

Returns
Primary key(s) of the given table.

Return type
list | pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> from pyhelpers._cache import example_dataframe
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> dat = example_dataframe()
>>> dat
           Longitude   Latitude
City
London      -0.127647  51.507322
```

(continues on next page)

(continued from previous page)

```
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> tbl_name = 'test_table'
>>> testdb.import_data(data=dat, table_name=tbl_name, index=True, verbose=True)
To import data into "public"."test_table" at postgres:***@localhost:5432/postgres
? [No]|Yes: yes
>>> pri_keys = testdb.get_primary_keys(table_name=tbl_name)
>>> pri_keys
[]
>>> testdb.add_primary_keys(primary_keys='City', table_name=tbl_name)
```

The "*test_table*" is illustrated in Figure 10 below:

	City [PK] text	Longitude double precision	Latitude double precision
1	Birmingham	-1.9026911	52.4796992
2	Leeds	-1.5437941	53.7974185
3	London	-0.1276474	51.5073219
4	Manchester	-2.2451148	53.4794892

Figure 10: The table "*test_table*" (with a primary key) in the database.

```
>>> pri_keys = testdb.get_primary_keys(table_name=tbl_name)
>>> pri_keys
['City']
>>> pri_keys = testdb.get_primary_keys(table_name=tbl_name, names_only=False)
>>> pri_keys
  key_column data_type
0       City      text
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.get_schema_info

PostgresOSM.**get_schema_info**(*names_only=True*, *include_all=False*, *column_names=None*, *verbose=False*)

Retrieves information about existing schemas.

Parameters

- **names_only** (*bool*) – Whether to return only the names of the schemas; defaults to True.
- **include_all** (*bool*) – Whether to list all available schemas; defaults to False.

- **column_names** (*list | None*) – Column names for the returned dataframe if `names_only=False`; defaults to `['schema_name', 'schema_id', 'role']` if `column_names=None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Returns
Names of schemas or a dataframe with schema information if requested.

Return type
list | pandas.DataFrame | None

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.get_schema_info()
['public']
>>> testdb.get_schema_info(names_only=False, include_all=True)
          schema_name       schema_owner    oid  nspowner
0              public  pg_database_owner   2200      6171
1            pg_toast           postgres     99        10
2          pg_catalog           postgres     11        10
3  information_schema           postgres  13183        10
>>> testdb.create_schema(schema_name='test_schema', verbose=True)
Creating a schema: "test_schema" ... Done.
>>> testdb.get_schema_info()
['public', 'test_schema']
>>> testdb.drop_schema(schema_names=['public', 'test_schema'], verbose=True)
To drop the following schemas from postgres:***@localhost:5432/testdb:
  "public"
  "test_schema"
? [No]|Yes: yes
Dropping ...
  "public" ... Done.
  "test_schema" ... Done.
>>> testdb.get_schema_info() is None
True
>>> testdb.get_schema_info(verbose=True)
No schema exists in the currently-connected database "testdb".
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.get_table_column_info

PostgresOSM.**get_table_column_info**(*subregion_name*, *layer_name*, *as_dict=False*, *table_named_as_subregion=False*, *schema_named_as_layer=False*)

Get information about columns of a specific schema and table data for a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region, which acts as a table name.
- **layer_name** (*str*) – name of an OSM layer (e.g. 'points', 'railways', ...), which acts as a schema name.
- **as_dict** (*bool*) – whether to return the column information as a dictionary. Defaults to `True`.
- **table_named_as_subregion** (*bool*) – whether to use subregion name as table name, defaults to `False`
- **schema_named_as_layer** (*bool*) – whether a schema is named as a layer name. Defaults to `False`.

Returns

information about each column of the given table

Return type

pandas.DataFrame | dict

Examples:

```
>>> from pydriosm.ios._base import BaseIOS

>>> osmdb = BaseIOS(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> subrgn_name = 'London'
>>> lyr_name = 'points'

>>> # Take for example a table named "points"."London"
>>> tbl_col_info = osmdb.get_table_column_info(subrgn_name, lyr_name)
>>> type(tbl_col_info)
pandas.core.frame.DataFrame
>>> tbl_col_info.index.to_list()[:5]
['table_catalog',
 'table_schema',
 'table_name',
 'column_name',
 'ordinal_position']

>>> # Another example of a table named "points"."Greater London"
>>> tbl_col_info_dict = osmdb.get_table_column_info(
...     subrgn_name, lyr_name, as_dict=True, table_named_as_subregion=True,
...     schema_named_as_layer=True)
>>> type(tbl_col_info_dict)
dict
>>> list(tbl_col_info_dict.keys())[:5]
['table_catalog',
 'table_schema',
 'table_name',
 'column_name',
 'ordinal_position']
```

(continues on next page)

(continued from previous page)

```
>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

PostgresOSM.get_table_name

PostgresOSM.**get_table_name**(*subregion_name*, *table_named_as_subregion=False*)

Get the default table name for a specific geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region, which acts as a table name
- **table_named_as_subregion** (*bool*) – whether to use subregion name as table name, defaults to False

Returns

default table name for storing the subregion data into the database

Return type

str

Examples:

```
>>> from pydriosm.ios._base import BaseIOS

>>> osmdb = BaseIOS(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> subrgn_name = 'london'

>>> tbl_name = osmdb.get_table_name(subrgn_name)
>>> tbl_name
'london'

>>> tbl_name = osmdb.get_table_name(subrgn_name, table_named_as_subregion=True)
>>> tbl_name
'Greater London'

>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> tbl_name = osmdb.get_table_name(subrgn_name, table_named_as_subregion=True)
>>> tbl_name
'London'

>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

> **Note**
>
> In the examples above, the default data source is 'Geofabrik'. Changing it to 'BBBike', the function may produce a different output for the same input, as a geographic (sub)region that is included in one data source may not always be available from the other.

PostgresOSM.get_table_names

PostgresOSM.**get_table_names**(*schema_name=None*, *verbose=False*)

Retrieves the names of all tables in a schema.

Parameters

- **schema_name** (*str | list | None*) – Name of the schema; if schema_name=None (default), it defaults to DEFAULT_SCHEMA (i.e. 'public').
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

Table names of the specified schema(s) schema_name.

Return type

dict

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> tbl_names = testdb.get_table_names()
>>> tbl_names
{'public': []}
>>> tbl_names = testdb.get_table_names(schema_name='testdb', verbose=True)
The schema "testdb" does not exist.
>>> tbl_names is None
True
>>> # Create a new table named "test_table" in the schema "testdb"
>>> new_tbl_name = 'test_table'
>>> col_spec = 'col_name_1 INT, col_name_2 TEXT'
>>> testdb.create_table(table_name=new_tbl_name, column_specs=col_spec, verbose=True)
Creating a table: "public"."test_table" ... Done.
>>> tbl_names = testdb.get_table_names(schema_name='public')
>>> tbl_names
{'public': ['test_table']}
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.import_data

PostgresOSM.**import_data**(*data*, *table_name*, *schema_name=None*, *if_exists='fail'*, *force_replace=False*, *chunk_size=None*, *dtype=None*, *method='multi'*, *index=False*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Imports tabular data into the database.

See also [DBMS-PS-ID-1] and [DBMS-PS-ID-2].

Parameters

- **data** (*pandas.DataFrame | pandas.io.parsers.TextFileReader | list | tuple*) – Tabular data to be imported into the database.
- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema; if `schema_name=None` (default), it defaults to `DEFAULT_SCHEMA` (i.e. `'public'`).
- **if_exists** (*str*) – Action to take if the table already exists. Options are `'replace'`, `'append'` or `'fail'` (default).
- **force_replace** (*bool*) – Whether to force replace an existing table; defaults to `False`.
- **chunk_size** (*int | None*) – Number of rows in each batch to be written at a time; defaults to `None`.
- **dtype** (*dict | None*) – Data types for columns; defaults to `None`.
- **method** (*str | None | Callable*) – Method for SQL insertion clause; defaults to `'multi'`.
  - `None`: Uses standard SQL `INSERT` clause (one per row).
  - `'multi'`: Passes multiple values in a single `INSERT` clause.
  - Callable (e.g. `PostgreSQL.psql_insert_copy`) with signature `(pd_table, conn, keys, data_iter)`.
- **index** (*bool*) – Whether to include the index as a column in the table.
- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **kwargs** – [Optional] Additional parameters for the method pandas.DataFrame.to_sql().

See also

- Examples for the method `read_sql_query()`.

PostgresOSM.import_osm_data

PostgresOSM.**import_osm_data**(*osm_data*, *table_name*, *schema_names=None*, *table_named_as_subregion=False*, *schema_named_as_layer=False*, *if_exists='fail'*, *force_replace=False*, *chunk_size=None*, *confirmation_required=True*, *verbose=False*, *raise_error=False*, ***kwargs*)

Import OSM data into a database.

Parameters

- **osm_data** (*dict*) – OSM data of a geographic (sub)region
- **table_name** (*str*) – name of a table
- **schema_names** (*list | dict | None*) – names of schemas for each layer of the PBF data. Defaults to `None`; when `schema_names=None`, the default layer names as schema names
- **table_named_as_subregion** (*bool*) – whether to use subregion name as a table name, defaults to `False`
- **schema_named_as_layer** (*bool*) – whether a schema is named as a layer name, defaults to `False`
- **if_exists** (*str*) – if the table already exists. Defaults to `'fail'`; valid options include `{'replace', 'append', 'fail'}`
- **force_replace** (*bool*) – whether to force to replace existing table. Defaults to `False`
- **chunk_size** (*int | None*) – the number of rows in each batch to be written at a time, defaults to `None`
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`
- **raise_error** (*bool*) – Whether to raise the provided exception. If `raise_error=False` (default), the error will not be raised.
- **kwargs** – [optional] parameters of the method *import_osm_layer()*

Examples:

```
>>> from pydriosm.ios._base import BaseIOS
>>> from pyhelpers.dirs import delete_dir

>>> osmdb = BaseIOS(database_name='osmdb_test', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> subrgn_name = 'Rutland'  # name of a subregion
>>> dat_dir = "tests/osm_data"  # name of a data directory where the subregion data is
```

Example 1 - Import data of a PBF file:

```
>>> # First, read the PBF data of Rutland
>>> # (If the data file is not available, it'll be downloaded by confirmation)
>>> raw_rutland_pbf = osmdb.reader.read_pbf(
...     subrgn_name, dat_dir, download=True, verbose=True)
Downloading "rutland-latest.osm.pbf" 100%|■■■■■■■■■■■| 1.92M/1.92M | 4.88MB/s ...
  Saving "rutland-latest.osm.pbf" to "./tests/osm_data/rutland/" ... Done.
Reading "./tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
>>> type(raw_rutland_pbf)
dict
>>> list(raw_rutland_pbf.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> # Import all layers of the raw PBF data of Rutland
>>> osmdb.import_osm_data(raw_rutland_pbf, table_name=subrgn_name, verbose=True)
Proceed to import data into table "Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
  "points" ... Done. (6473 features)
  "lines" ... Done. (11057 features)
  "multilinestrings" ... Done. (71 features)
  "multipolygons" ... Done. (9423 features)
  "other_relations" ... Done. (33 features)

>>> # Get parsed PBF data
>>> parsed_rutland_pbf = osmdb.reader.read_pbf(
...     subregion_name=subrgn_name, data_dir=dat_dir, expand=True,
...     parse_geometry=True, parse_other_tags=True, verbose=True)
Parsing "./tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
>>> type(parsed_rutland_pbf)
dict
>>> list(parsed_rutland_pbf.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> # Import data of selected layers into specific schemas
>>> schemas = {
...     "schema_0": 'lines',
...     "schema_1": 'points',
...     "schema_2": 'multipolygons',
... }
>>> osmdb.import_osm_data(parsed_rutland_pbf, subrgn_name, schemas, verbose=True)
Proceed to import data into table "Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
  "schema_0" ... Done. (11057 features)
  "schema_1" ... Done. (6473 features)
  "schema_2" ... Done. (9423 features)

>>> # To drop the schemas "schema_0", "schema_1" and "schema_2"
>>> osmdb.drop_schema(schemas.keys(), confirmation_required=False, verbose=True)
Dropping the following schemas from postgres:***@localhost:5432/osmdb_test:
  "schema_0" ... Done.
  "schema_1" ... Done.
  "schema_2" ... Done.
```

Example 2 - Import OSM GeoPackage data:

```
>>> # Read GeoPackage data of Rutland
>>> rutland_gpkg = osmdb.reader.read_gpkg(
...     subregion_name=subrgn_name, data_dir=dat_dir, download=True, verbose=True)
Downloading "rutland-latest-free.gpkg.zip" 100%|■■■■■■■■■■| 3.30M/3.30M | 5.0...
  Saving "rutland-latest-free.gpkg.zip" to "./tests/osm_data/rutland/" ... Done.
Parsing the data ... Done.
>>> type(rutland_gpkg)
dict
>>> list(rutland_gpkg.keys())
['traffic',
 'places',
 'pois',
 'transport',
 'pofw',
 'natural',
 'railways',
 'roads',
 'waterways',
 'protected_areas',
 'water',
 'landuse',
 'buildings',
 'adminareas']

>>> # Import all layers of the shapefile data of Rutland
>>> osmdb.import_osm_data(osm_data=rutland_gpkg, table_name=subrgn_name, verbose=True)
Proceed to import data into table "Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
  "traffic" ... Done. (557 features)
  "places" ... Done. (301 features)
  "pois" ... Done. (1081 features)
  "transport" ... Done. (65 features)
  "pofw" ... Done. (65 features)
  "natural" ... Done. (665 features)
  "railways" ... Done. (141 features)
  "roads" ... Done. (7315 features)
  "waterways" ... Done. (379 features)
  "protected_areas" ... Done. (20 features)
  "water" ... Done. (233 features)
  "landuse" ... Done. (2461 features)
  "buildings" ... Done. (5706 features)
  "adminareas" ... Done. (58 features)
```

Example 3 - Import BBBike shapefile data file of Leeds:

```
>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> subrgn_name = 'Leeds'

>>> # Read shapefile data of Leeds
>>> leeds_shp = osmdb.reader.read_shp(
...     subregion_name=subrgn_name, data_dir=dat_dir, download=True, rm_extracts=True,
...     verbose=True)
Downloading "Leeds.osm.shp.zip" 100%|■■■■■■■■■■| 57.7M/57.7M | 20.6MB/s | ETA...
  Saving "Leeds.osm.shp.zip" to "./tests/osm_data/leeds/" ... Done.
Extracting "./tests/osm_data/leeds/Leeds.osm.shp.zip"
```

(continues on next page)

(continued from previous page)

```
  to "./tests/osm_data/leeds/" ... Done.
Reading the shapefile(s) at "./tests/osm_data/leeds/Leeds-shp/shape/" ... Done.
Deleting the extracts "./tests/osm_data/leeds/Leeds-shp/" ... Done.
>>> type(leeds_shp)
dict
>>> list(leeds_shp.keys())
['buildings',
 'landuse',
 'natural',
 'places',
 'points',
 'railways',
 'roads',
 'waterways']

>>> # Import all layers of the shapefile data of Leeds
>>> osmdb.import_osm_data(osm_data=leeds_shp, table_name=subrgn_name, verbose=True)
Proceed to import data into table "Leeds" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
  "buildings" ... Done. (432701 features)
  "landuse" ... Done. (22180 features)
  "natural" ... Done. (7759 features)
  "places" ... Done. (892 features)
  "points" ... Done. (47332 features)
  "railways" ... Done. (2897 features)
  "roads" ... Done. (152155 features)
  "waterways" ... Done. (3520 features)
```

Delete the test database and downloaded data files:

```
>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
Drop the database "osmdb_test" from postgres:***@localhost:5432?
 [No]|Yes: yes
Dropping "osmdb_test" ... Done.

>>> # Delete the downloaded data files
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

PostgresOSM.import_osm_layer

PostgresOSM.**import_osm_layer**(*layer_data*, *table_name*, *schema_name*, *table_named_as_subregion=False*, *schema_named_as_layer=False*, *if_exists='fail'*, *chunk_size=None*, *confirmation_required=True*, *verbose=False*, *raise_error=True*, ***kwargs*)

Import one layer of OSM data into a table.

Parameters

- **layer_data** (*pandas.DataFrame | geopandas.GeoDataFrame*) – one

layer of OSM data

- **schema_name** (*str*) – name of a schema (or name of a PBF layer)
- **table_name** (*str*) – name of a table
- **table_named_as_subregion** (*bool*) – whether to use subregion name as a table name. Defaults to `False`.
- **schema_named_as_layer** (*bool*) – whether a schema is named as a layer name. Defaults to `False`.
- **if_exists** (*str*) – if the table already exists. Valid options include `{'replace', 'append', 'fail'}`. Defaults to `'fail'`.
- **chunk_size** (*int | None*) – the number of rows in each batch to be written at a time. Defaults to `None`.
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed. Defaults to `True`.
- **verbose** (*bool*) – whether to print relevant information in console as the function runs. Defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception. If `raise_error=False`, the error will be suppressed. Defaults to `True`.
- **kwargs** – Ooptional parameters of pyhelpers.dbms.PostgreSQL.import_data.

Examples:

```
>>> from pydriosm.ios._base import BaseIOS
>>> from pyhelpers.dirs import delete_dir

>>> osmdb = BaseIOS(database_name='osmdb_test', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> subrgn_name = 'Rutland'  # name of a subregion
>>> dat_dir = "tests/osm_data"  # name of a data directory where the subregion data is
```

Example 1 - Import data of the 'points' layer of a PBF file:

```
>>> # First, read the PBF data of Rutland (from Geofabrik free download server)
>>> # (If the data file is not available, it'll be downloaded by confirmation)
>>> raw_pbf = osmdb.reader.read_pbf(
...     subrgn_name, data_dir=dat_dir, download=True, verbose=True)
Downloading "rutland-latest.osm.pbf" 100%|■■■■■■■■■■■| 1.89M/1.89M | 5.76MB/s ...
  Saving "rutland-latest.osm.pbf" to "./tests/osm_data/rutland/" ... Done.
Reading "./tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
>>> type(raw_pbf)
dict
>>> list(raw_pbf.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> # Get the data of 'points' layer
>>> points_key = 'points'
```

(continues on next page)

(continued from previous page)

```
>>> raw_pbf_points = raw_pbf[points_key]
>>> type(raw_pbf_points)
list
>>> type(raw_pbf_points[0])
osgeo.ogr.Feature

>>> # Now import the data of 'points' into the PostgreSQL server
>>> osmdb.import_osm_layer(
...     layer_data=raw_pbf_points, table_name=subrgn_name, schema_name=points_key,
...     verbose=True)
To import data into the table "points"."Rutland" at postgres:***@localhost:5432/osm...
? [No]|Yes: yes
Creating a schema: "points" ... Done.
Importing the data into "points"."Rutland" ... Done.

>>> tbl_col_info = osmdb.get_table_column_info(subrgn_name, points_key)
>>> tbl_col_info.head()
                    column_0
table_catalog     osmdb_test
table_schema          points
table_name           Rutland
column_name           points
ordinal_position           1

>>> # Parse the 'geometry' of the PBF data of Rutland
>>> parsed_pbf = osmdb.reader.read_pbf(
...     subregion_name=subrgn_name, data_dir=dat_dir, expand=True,
...     parse_geometry=True, verbose=True)
>>> type(parsed_pbf)
dict
>>> list(parsed_pbf.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']
>>> # Get the parsed data of 'points' layer
>>> parsed_pbf_points = parsed_pbf[points_key]
>>> type(parsed_pbf_points)
pandas.DataFrame
>>> parsed_pbf_points.head()
         id  ...                                         properties
0    488658  ...  {'osm_id': '488658', 'name': 'Tickencote Inter...
1  13883868  ...  {'osm_id': '13883868', 'name': None, 'barrier'...
2  14049101  ...  {'osm_id': '14049101', 'name': None, 'barrier'...
3  14558402  ...  {'osm_id': '14558402', 'name': None, 'barrier'...
4  14558409  ...  {'osm_id': '14558409', 'name': None, 'barrier'...
[5 rows x 3 columns]

>>> # Import the parsed 'points' data into the PostgreSQL database
>>> osmdb.import_osm_layer(
...     layer_data=parsed_pbf_points, table_name=subrgn_name, schema_name=points_key,
...     if_exists='replace', verbose=True)
Import data into "points"."Rutland" at postgres:***@localhost:5432/osmdb_test?
 [No]|Yes: yes
Importing the data ... Done.

>>> # Get the information of the table "points"."Rutland"
>>> tbl_col_info = osmdb.get_table_column_info(subrgn_name, points_key)
>>> tbl_col_info.head()
                    column_0    column_1    column_2
```

(continues on next page)

(continued from previous page)

```
table_catalog     osmdb_test  osmdb_test  osmdb_test
table_schema          points      points      points
table_name           Rutland     Rutland     Rutland
column_name               id    geometry  properties
ordinal_position           1           2           3
```

Example 2 - Import data of the 'railways' layer of a GeoPackage*:

```
>>> # Read the data of 'railways' layer and delete the extracts
>>> lyr_name = 'railways'
>>> rutland_railways_gpkg = osmdb.reader.read_gpkg(
...     subregion_name=subrgn_name, layer_names=lyr_name, data_dir=dat_dir,
...     download=True, verbose=True)
Downloading "rutland-latest-free.gpkg.zip" 100%|■■■■■■■■■■■| 3.30M/3.30M | 3.6...
  Saving "rutland-latest-free.gpkg.zip" to "./tests/osm_data/rutland/" ... Done.
Parsing the data ... Done.
>>> type(rutland_railways_gpkg)
dict
>>> list(rutland_railways_gpkg.keys())
['railways']

>>> # Get the data of 'railways' layer
>>> rutland_railways_gpkg_ = rutland_railways_gpkg[lyr_name]
>>> rutland_railways_gpkg_.head()
    osm_id  code  ... tunnel                                           geometry
0  2162114  6101  ...      F  LINESTRING (-0.46449 52.71043, -0.46111 52.706...
1  3681043  6101  ...      F  LINESTRING (-0.65312 52.57308, -0.65318 52.572...
2  3693985  6101  ...      F  LINESTRING (-0.72203 52.69658, -0.72182 52.697...
3  3693986  6101  ...      F  LINESTRING (-0.61731 52.61323, -0.62419 52.614...
4  8044108  6101  ...      T  LINESTRING (-0.69782 52.62858, -0.70271 52.63395)
[5 rows x 8 columns]

>>> # Import the 'railways' data into the PostgreSQL database
>>> osmdb.import_osm_layer(
...     rutland_railways_gpkg_, table_name=subrgn_name, schema_name=lyr_name,
...     verbose=True)
Import data into "railways"."Rutland" at postgres:***@localhost:5432/osmdb_test?
 [No]|Yes: yes
Creating a schema: "railways" ... Done.
Importing the data ... Done.

>>> # Get the information of the table "railways"."Rutland"
>>> tbl_col_info = osmdb.get_table_column_info(subrgn_name, lyr_name)
>>> tbl_col_info.head()
                   column_0    column_1  ...    column_6    column_7
table_catalog     osmdb_test  osmdb_test  ...  osmdb_test  osmdb_test
table_schema        railways    railways  ...    railways    railways
table_name           Rutland     Rutland  ...     Rutland     Rutland
column_name           osm_id        code  ...      tunnel    geometry
ordinal_position           1           2  ...           7           8
[5 rows x 8 columns]
```

Delete the test database and downloaded data files:

```
>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
```

(continues on next page)

(continued from previous page)

```
Drop the database "osmdb_test" from postgres:***@localhost:5432?
 [No]|Yes: yes
Dropping "osmdb_test" ... Done.

>>> # Delete the downloaded data files
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

PostgresOSM.import_osm_pbf

PostgresOSM.**import_osm_pbf**(*subregion_names*, *data_dir=None*, *update_osm_pbf=False*, *if_exists='fail'*, *chunk_size_limit=50*, *expand=False*, *parse_geometry=False*, *parse_properties=False*, *parse_other_tags=False*, *pickle_pbf_file=False*, *rm_pbf_file=False*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Import data of geographic (sub)region(s) that do not have (sub-)subregions into a database.

Parameters

- **subregion_names** (*str | list | None*) – name(s) of geographic (sub)region(s)
- **data_dir** (*str | None*) – directory where the PBF data file is located/saved; if `None` (default), the default directory
- **update_osm_pbf** (*bool*) – whether to update .osm.pbf data file (if available), defaults to `False`
- **if_exists** (*str*) – if the table already exists, defaults to `'fail'`; valid options include `{'replace', 'append', 'fail'}`
- **chunk_size_limit** (*int*) – threshold (in MB) that triggers the use of chunk parser, defaults to `50`; if the size of the .osm.pbf file (in MB) is greater than `chunk_size_limit`, it will be parsed in a chunk-wise way
- **expand** (*bool*) – whether to expand dict-like data into separate columns, defaults to `False`
- **parse_geometry** (*bool*) – whether to represent the `'geometry'` field in a shapely.geometry format, defaults to `False`
- **parse_properties** (*bool*) – whether to represent the `'properties'` field in a tabular format, defaults to `False`
- **parse_other_tags** (*bool*) – whether to represent a `'other_tags'` (of `'properties'`) in a dict format, defaults to `False`
- **pickle_pbf_file** (*bool*) – whether to save the .pbf data as a .pickle file, defaults to `False`

- **rm_pbf_file** (*bool*) – whether to delete the downloaded .osm.pbf file, defaults to `False`
- **confirmation_required** (*bool*) – whether to ask for confirmation to proceed, defaults to `True`
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of the method `_import_pbf()` or `_import_pbf_chunk_wisely()`

Examples:

```
>>> from pydriosm.ios import PostgresOSM
>>> from pyhelpers.dirs import cd, delete_dir
>>> from pyhelpers.store import load_pickle
>>> osmdb = PostgresOSM(database_name='osmdb_test', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.
```

Example 1 - Import PBF data of Rutland:

```
>>> subrgn_name = 'Rutland'  # name of a subregion
>>> dat_dir = "tests/osm_data"  # name of a data directory where the subregion data is
>>> osmdb.import_osm_pbf(subrgn_name, data_dir=dat_dir, verbose=True)
Proceed to import .osm.pbf data of the following geographic (sub)region(s):
    "Rutland"
  into postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" 100%|■■■■■■■■■■■| 1.89M/1.89M | 6.46MB/s ...
  Saving "rutland-latest.osm.pbf" to "./tests/osm_data/rutland/" ... Done.
Reading "tests/osm_data/rutland/rutland-latest.osm.pbf" ... Done.
Importing the data into the table "Rutland" ...
    "points" ... Done. (6420 features)
    "lines" ... Done. (10958 features)
    "multilinestrings" ... Done. (71 features)
    "multipolygons" ... Done. (9194 features)
    "other_relations" ... Done. (33 features)
```

Example 2 - Import PBF data of Leeds and London:

```
>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> subrgn_names = ['Leeds', 'London']
>>> # Note this may take a few minutes (or longer)
>>> osmdb.import_osm_pbf(
...     subregion_names=subrgn_names, data_dir=dat_dir, expand=True,
...     parse_geometry=True, parse_properties=True, parse_other_tags=True,
...     pickle_pbf_file=True, rm_pbf_file=True, verbose=True)
Proceed to import .osm.pbf data of the following geographic (sub)region(s):
    "Leeds"
    "London"
  into postgres:***@localhost:5432/osmdb_test
? [No]|Yes: >? yes
Downloading "Leeds.osm.pbf" 100%|■■■■■■■■■■■| 38.1M/38.1M | 7.73MB/s | ETA: 00:00
```

(continues on next page)

(continued from previous page)

```
  Saving "Leeds.osm.pbf" to "./tests/osm_data/leeds/" ... Done.
Reading "tests/osm_data/leeds/Leeds.osm.pbf" ... Done.
Importing the data into the table "Leeds" ...
    "points" ... Done. (102238 features)
    "lines" ... Done. (183218 features)
    "multilinestrings" ... Done. (442 features)
    "multipolygons" ... Done. (481516 features)
    "other_relations" ... Done. (7185 features)
Saving "Leeds-pbf.pickle" to "./tests/osm_data/leeds/" ... Done.
Deleting "tests/osm_data/leeds/Leeds.osm.pbf" ... Done.
Downloading "London.osm.pbf" 100%|■■■■■■■■■■■■| 188M/188M | 18.4MB/s | ETA: 00:00
  Saving "London.osm.pbf" to "./tests/osm_data/london/" ... Done.
Importing the data of "London" chunk-wisely
  into postgres:***@localhost:5432/osmdb_test ...
    "points" ... Done. (972803 features)
    "lines" ... Done. (1053607 features)
    "multilinestrings" ... Done. (1017 features)
    "multipolygons" ... Done. (1976 features)
    "other_relations" ... Done. (22590 features)
Saving "London-pbf.pickle" to "./tests/osm_data/london/" ... Done.
Deleting "tests/osm_data/london/London.osm.pbf" ... Done.

>>> # As `pickle_pbf_file=True`, the parsed PBF data have been saved as pickle files

>>> # Data of Leeds
>>> leeds_pbf = load_pickle(cd(dat_dir, "leeds", "Leeds-pbf.pickle"))
>>> type(leeds_pbf)
dict
>>> list(leeds_pbf.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']
>>> # Data of the 'points' layer of Leeds
>>> leeds_pbf_points = leeds_pbf['points']
>>> leeds_pbf_points.head()
       id                        geometry  ... man_made             other_tags
0  154941  POINT (-1.5560511 53.6879848)  ...     None  {'name:signed': 'no'}
1  154962     POINT (-1.34293 53.844618)  ...     None  {'name:signed': 'no'}
2  155014   POINT (-1.517335 53.7499667)  ...     None  {'name:signed': 'no'}
3  155023   POINT (-1.514175 53.7418444)  ...     None  {'name:signed': 'no'}
4  155035   POINT (-1.516511 53.7256632)  ...     None  {'name:signed': 'no'}
[5 rows x 11 columns]

>>> # Data of London
>>> london_pbf = load_pickle(cd(dat_dir, "london", "London-pbf.pickle"))
>>> type(london_pbf)
dict
>>> list(london_pbf.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']
>>> # Data of the 'points' layer of London
>>> london_pbf_points = london_pbf['points']
>>> london_pbf_points.head()
      id  ...                                          other_tags
0  99878  ...  {'access': 'permissive', 'bicycle': 'no', 'mot...
1  99880  ...  {'crossing': 'unmarked', 'crossing:island': 'n...
2  99884  ...                       {'amenity': 'waste_basket'}
3  99918  ...                        {'emergency': 'life_ring'}
4  99939  ...          {'traffic_signals:direction': 'forward'}
[5 rows x 11 columns]
```

Delete the test database and downloaded data files:

```
>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
Proceed to drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.

>>> # Delete the downloaded data files
>>> delete_dir(dat_dir, verbose=True)
To delete the directory "./tests/osm_data/" (Not empty)
? [No]|Yes: yes
Deleting "./tests/osm_data/" ... Done.
```

PostgresOSM.null_text_to_empty_string

PostgresOSM.**null_text_to_empty_string**(*table_name*, *column_names=None*, *schema_name=None*)

Converts null values (in text columns) to empty strings.

Parameters

- **table_name** (*str*) – Name of the table.
- **column_names** (*str | list | None*) – (List of) column name(s) to convert null values to empty strings; if `column_names=None` (default), all available columns are included.
- **schema_name** (*str | None*) – Name of the schema; defaults to `None`.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> from pyhelpers._cache import example_dataframe
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> dat = example_dataframe()
>>> dat['Longitude'] = dat['Longitude'].astype(str)
>>> dat.loc['London', 'Longitude'] = None
>>> dat
            Longitude   Latitude
City
London            None  51.507322
Birmingham  -1.9026911  52.479699
Manchester  -2.2451148  53.479489
Leeds       -1.5437941  53.797418
>>> tbl_name = 'test_table'
>>> testdb.import_data(data=dat, table_name=tbl_name, index=True, verbose=True)
To import data into "public"."test_table" at postgres:***@localhost:5432/postgres
? [No]|Yes: yes
>>> testdb.table_exists(table_name=tbl_name)
True
>>> testdb.get_column_dtype(table_name=tbl_name)
{'City': 'text', 'Longitude': 'text', 'Latitude': 'double precision'}
```

(continues on next page)

(continued from previous page)

```
>>> dat_ = testdb.read_table(tbl_name)
>>> dat_.loc[0, 'Longitude'] is None
True
```

	City text	Longitude text	Latitude double precision
1	London	[null]	51.5073219
2	Birmingham	-1.9026911	52.4796992
3	Manchester	-2.2451148	53.4794892
4	Leeds	-1.5437941	53.7974185

Figure 11: The table "*test_table*" in the database "*testdb*".

```
>>> # Replace the 'null' value with an empty string
>>> testdb.null_text_to_empty_string(table_name=tbl_name)
>>> dat_ = testdb.read_table(tbl_name)
>>> dat_.loc[0, 'Longitude']
''
```

	City text	Longitude text	Latitude double precision
1	London		51.5073219
2	Birmingham	-1.9026911	52.4796992
3	Manchester	-2.2451148	53.4794892
4	Leeds	-1.5437941	53.7974185

Figure 12: The table "*test_table*" in the database "*testdb*" (after converting 'null' to empty string).

```
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.postprocess_pdf_layer

PostgresOSM.**postprocess_pdf_layer**(*layer_dat*, *decode_geojson=True*, *sort_by='id'*)

Post-process the data of a specific layer.

PostgresOSM.psql_insert_copy

static PostgresOSM.**psql_insert_copy**(*sql_table*, *sql_db_engine*, *column_names*, *data_iter*)

Callable function using *PostgreSQL* `COPY` clause for executing data insertion.

Parameters

- **sql_table** (*pandas.io.sql.SQLTable*) – Object that represents the table to insert into.
- **sql_db_engine** (*sqlalchemy.engine.Connection | sqlalchemy.engine.Engine*) – Object that represents the database engine or connection.
- **column_names** (*list[str]*) – List of column names to insert data into.
- **data_iter** (*Iterable*) – Iterable that iterates over the values to be inserted.

Note

This function is modified from the source code available at [DBMS-PS-PIC-1].

PostgresOSM.read_sql_query

PostgresOSM.**read_sql_query**(*sql_query*, *method='tempfile'*, *max_size_spooled=1*, *delimiter=','*, *tempfile_kwargs=None*, *stringio_kwargs=None*, ***kwargs*)

Reads table data by executing a SQL query (recommended for large tables).

See also [DBMS-PS-RSQ-1], [DBMS-PS-RSQ-2] and [DBMS-PS-RSQ-3].

Parameters

- **sql_query** (*str*) – SQL query to be executed.
- **method** (*str*) – Method to be used for buffering temporary data.
  - `'tempfile'` (default): Uses tempfile.TemporaryFile().
  - `'stringio'`: Uses io.StringIO().
  - `'spooled'`: Uses tempfile.SpooledTemporaryFile().
- **max_size_spooled** (*int | float*) – Maximum size of the file generated via tempfile.SpooledTemporaryFile(); defaults to `1` (in gigabyte).
- **delimiter** (*str*) – Delimiter used in data; defaults to `','`.
- **tempfile_kwargs** – [Optional] Additional parameters for tempfile.TemporaryFile() or tempfile.SpooledTemporaryFile(); defaults to `None`.
- **stringio_kwargs** – [Optional] Additional parameters for io.StringIO(), e.g. `initial_value`; defaults to `None`.
- **kwargs** – [Optional] Additional parameters for the function pandas.read_csv().

Returns

Data queried by the statement `sql_query`.

Return type

pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> from pyhelpers._cache import example_dataframe
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> # Create an example dataframe
>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> table = 'England'
>>> schema = 'points'
>>> # Import the data into a table named "points"."England"
>>> testdb.import_data(example_df, table, schema, index=True, verbose=2)
Import data into "points"."England" at postgres:***@localhost:5432/testdb?
 [No]|Yes: yes
Creating a schema: "points" ... Done.
Importing the data into "points"."England" ... Done.
```

The table *"points"."England"* is illustrated in Figure 10 below:

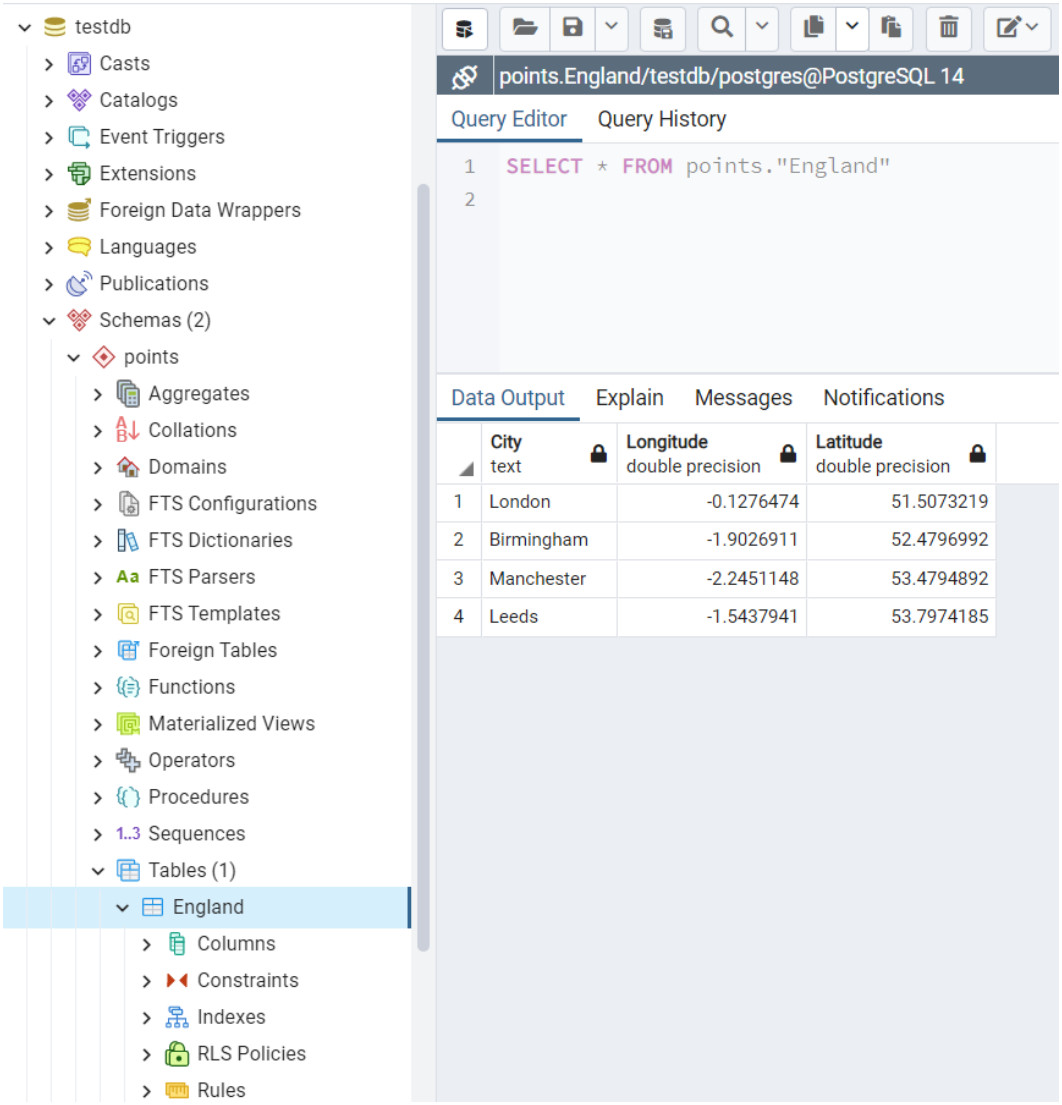

Figure 13: The table "*points*"."*England*" in the database "*testdb*".

```
>>> res = testdb.table_exists(table_name=table, schema_name=schema)
>>> print(f'The table "{schema}"."{table}" exists? {res}.')
The table "points"."England" exists? True.
>>> # Retrieve the data using the method .read_table()
>>> example_df_ret = testdb.read_table(table, schema_name=schema, index_col='City')
>>> example_df_ret
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> # Alternatively, read the data by a SQL query statement
>>> sql_qry = f'SELECT * FROM "{schema}"."{table}"'
>>> example_df_ret_alt = testdb.read_sql_query(sql_query=sql_qry, index_col='City')
>>> example_df_ret_alt
            Longitude   Latitude
City
```

(continues on next page)

(continued from previous page)

```
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> example_df_ret.equals(example_df_ret_alt)
True
>>> # Delete the table "points"."England"
>>> testdb.drop_table(table_name=table, schema_name=schema, verbose=True)
To drop the table "points"."England" from postgres:***@localhost:5432/testdb
? [No]|Yes: yes
Dropping "points"."England" ... Done.
>>> # Delete the schema "points"
>>> testdb.drop_schema(schema_names=schema, verbose=True)
To drop the schema "points" from postgres:***@localhost:5432/testdb
? [No]|Yes: yes
Dropping "points" ... Done.
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

Aside: a brief example of using the parameter `params` for pandas.read_sql

```
>>> import datetime
>>> import pandas as pd
>>> sql_qry = 'SELECT * FROM "table_name" '
...           'WHERE "timestamp_column_name" BETWEEN %(ts_start)s AND %(ts_end)s'
>>> params = {'d_start': datetime.datetime.today(), 'd_end': datetime.datetime.today()}
>>> data_frame = pd.read_sql(sql=sql_qry, con=testdb.engine, params=params)
```

PostgresOSM.read_table

PostgresOSM.**read_table**(*table_name*, *schema_name=None*, *conditions=None*, *chunk_size=None*, *sorted_by=None*, ***kwargs*)

Reads data from a specified table.

See also [DBMS-PS-RT-1].

Parameters

- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema; if `schema_name=None`, it defaults to `DEFAULT_SCHEMA` (i.e., `'public'`).
- **conditions** (*str | None*) – SQL conditions to filter rows; defaults to `None`.
- **chunk_size** (*int | None*) – Number of rows to fetch per iteration; defaults to `None`.
- **sorted_by** (*str | None*) – Name(s) of column(s) by which the retrieved data is sorted; defaults to `None`.

- **kwargs** – [Optional] Additional parameters for the method `read_sql_query()` or the function pandas.read_sql().

Returns
Data of the specified table.

Return type
pandas.DataFrame

See also

- Examples for the method `read_sql_query()`.

PostgresOSM.schema_exists

`PostgresOSM.`**`schema_exists`**(*schema_name*)

Checks if a schema exists.

Parameters
schema_name (*str*) – Name of the schema to check.

Returns
`True` if the schema exists, otherwise `False`.

Return type
bool

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.schema_exists('public')
True
>>> testdb.schema_exists('test_schema')  # (if the schema 'test_schema' does not exist)
False
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.subregion_table_exists

`PostgresOSM.`**`subregion_table_exists`**(*subregion_name*, *layer_name*, *table_named_as_subregion=False*, *schema_named_as_layer=False*)

Check if a table (for a geographic (sub)region) exists.

Parameters

- **subregion_name** (*str*) – name of a geographic (sub)region, which acts as a table name
- **layer_name** (*str*) – name of an OSM layer (e.g. 'points', 'railways', ...), which acts as a schema name
- **table_named_as_subregion** (*bool*) – whether to use subregion name as table name, defaults to `False`
- **schema_named_as_layer** (*bool*) – whether a schema is named as a layer name. Defaults to `False`.

Returns

`True` if the table exists, `False` otherwise

Return type

bool

Examples:

```
>>> from pydriosm.ios._base import BaseIOS

>>> osmdb = BaseIOS(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> subrgn_name = 'London'
>>> lyr_name = 'pt'

>>> # Check whether the table "pt"."london" is available
>>> osmdb.subregion_table_exists(subregion_name=subrgn_name, layer_name=lyr_name)
False

>>> # Check whether the table "points"."greater_london" is available
>>> osmdb.subregion_table_exists(
...     subregion_name=subrgn_name, layer_name=lyr_name, table_named_as_subregion=True,
...     schema_named_as_layer=True)
False

>>> # Delete the database 'osmdb_test'
>>> osmdb.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

PostgresOSM.table_exists

PostgresOSM.**table_exists**(*table_name*, *schema_name=None*)

Checks if a table exists.

Parameters

- **table_name** (*str*) – Name of the table to check.
- **schema_name** (*str | None*) – Name of the schema; if `schema_name=None` (default), it defaults to `DEFAULT_SCHEMA` (i.e. `'public'`).

Returns
`True` if the table exists in the currently-connected database, otherwise `False`.

Return type
bool

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> tbl_name = 'points'
>>> testdb.table_exists(table_name=tbl_name)  # (if 'public.points' does not exist)
False
>>> testdb.create_table(table_name=tbl_name, column_specs='column_0 INT', verbose=1)
Creating a table: "public"."points" ... Done.
>>> testdb.table_exists(table_name=tbl_name)
True
>>> testdb.drop_table(table_name=tbl_name, verbose=True)
To drop the table "public"."points" from postgres:***@localhost:5432/testdb
? [No]|Yes: yes
Dropping "public"."points" ... Done.
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.validate_column_names

PostgresOSM.**validate_column_names**(*table_name*, *schema_name=None*, *column_names=None*)

Validates column names for a query statement.

Parameters

- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema; defaults to `None`.
- **column_names** (*str | list | tuple | None*) – Column name(s) for a dataframe.

Returns
Validated column names formatted for a PostgreSQL query statement.

Return type
str

See also

- Examples for the method `create_table()`.

4.3.2 Customised alternatives (Optional)

1anti-flashwhitewhite

GeofabrikIOS(**kwargs)	Implement storage I/O of Geofabrik OpenStreetMap data extracts with PostgreSQL.
BBBikeIOS(**kwargs)	Implement storage I/O of BBBike exports of OpenStreetMap data with PostgreSQL.

GeofabrikIOS

class pydriosm.ios.**GeofabrikIOS**(***kwargs*)

Implement storage I/O of Geofabrik OpenStreetMap data extracts with PostgreSQL.

Parameters

kwargs – [optional] parameters of the class `PostgresOSM`

Variables

- **postgres** (`PostgresOSM`) – instance of the class `PostgresOSM`
- **downloader** (`GeofabrikDownloader`) – instance of the class *GeofabrikDownloader*
- **reader** (`GeofabrikReader`) – instance of the class `GeofabrikReader`

Examples:

```
>>> from pydriosm.ios import GeofabrikIOS
>>> gfi = GeofabrikIOS(database_name='osmdb_test', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.
>>> type(gfi.downloader)
pydriosm.downloader._wrapper.Downloader
>>> type(gfi.reader)
pydriosm.reader._wrapper.Reader
```

See also

- Examples for all the methods of the class *PostgresOSM*.

1anti-flashwhitewhite

DATA_SOURCES	Data source.

GeofabrikIOS.DATA_SOURCES

GeofabrikIOS.**DATA_SOURCES: list = ['Geofabrik']**

Data source.

1anti-flashwhitewhite

BBBikeIOS

class pydriosm.ios.**BBBikeIOS**(***kwargs*)

Implement storage I/O of BBBike exports of OpenStreetMap data with PostgreSQL.

Parameters

kwargs – [optional] parameters of the class PostgresOSM

Variables

- **downloader** (BBBikeDownloader) – instance of the class *BBBikeDownloader*
- **reader** (BBBikeReader) – instance of the class BBBikeReader

Examples:

```
>>> from pydriosm.ios import BBBikeIOS
>>> bbi = BBBikeIOS(database_name='osmdb_test', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.
>>> type(bbi.downloader)
pydriosm.downloader._wrapper.Downloader
>>> type(bbi.reader)
pydriosm.downloader._wrapper.Reader
```

See also

- Examples for all the methods of the class *PostgresOSM*.

1anti-flashwhitewhite

DATA_SOURCES	Data source.

BBBikeIOS.DATA_SOURCES

BBBikeIOS.**DATA_SOURCES: list = ['BBBike']**

Data source.

1anti-flashwhitewhite

4.3.3 Other utilities

Utilities for the *ios* module.

1anti-flashwhitewhite

get_default_layer_name(schema_name)	Get default name (as an input schema name) of an OSM layer for the class *PostgresOSM*.
validate_schema_names([schema_names, ...])	Validate schema names for importing data into a database.
validate_table_name(table_name[, sub_space])	Validate a table name for importing OSM data into a database.
make_data_items(osm_data, schema_names)	Map OSM data layers to specific schema names.
preprocess_osm_layer(layer_data, layer_name)	Preprocess layer data into a pandas DataFrame with WKT geometries.

get_default_layer_name

pydriosm.ios.utils.**get_default_layer_name**(*schema_name*)

Get default name (as an input schema name) of an OSM layer for the class *PostgresOSM*.

See, for example, the method *pydriosm.ios.PostgresOSM.import_osm_layer()*.

Parameters
schema_name (*str*) – name of a schema (or name of an OSM layer)

Returns
default name of the layer

Return type
str

Examples:

```
>>> from pydriosm.ios.utils import get_default_layer_name
>>> lyr_name = get_default_layer_name(schema_name='point')
>>> lyr_name
'points'
>>> lyr_name = get_default_layer_name(schema_name='land')
>>> lyr_name
'landuse'
```

validate_schema_names

pydriosm.ios.utils.**validate_schema_names**(*schema_names=None*, *schema_named_as_layer=False*)

Validate schema names for importing data into a database.

Parameters

- **schema_names** (*Iterable | None*) – one or multiple names of layers, e.g. 'points', 'lines', defaults to None
- **schema_named_as_layer** (*bool*) – whether to use default PBF layer name as the schema name, defaults to False

Returns

valid names of the schemas in the database

Return type

list

Examples:

```
>>> from pydriosm.ios.utils import validate_schema_names
>>> valid_names = validate_schema_names()
>>> valid_names
[]
>>> input_schema_names = ['point', 'polygon']
>>> valid_names = validate_schema_names(input_schema_names)
>>> valid_names
['point', 'polygon']
>>> valid_names = validate_schema_names(input_schema_names, schema_named_as_layer=True)
>>> valid_names
['points', 'multipolygons']
```

validate_table_name

pydriosm.ios.utils.**validate_table_name**(*table_name*, *sub_space=''*)

Validate a table name for importing OSM data into a database.

Parameters

- **table_name** (*str*) – name as input of a table in a PostgreSQL database
- **sub_space** (*str*) – substitute for space, defaults to ''

Returns

valid name of the table in the database

Return type

str

Examples:

```
>>> from pydriosm.ios.utils import validate_table_name
>>> subrgn_name = 'greater london'
>>> valid_table_name = validate_table_name(subrgn_name)
>>> valid_table_name
'greater london'
```

(continues on next page)

(continued from previous page)

```
>>> subrgn_name = 'Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch, Wales'
>>> valid_table_name = validate_table_name(subrgn_name, sub_space='_')
>>> valid_table_name
'Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch_W..'
```

make_data_items

pydriosm.ios.utils.**make_data_items**(*osm_data*, *schema_names*)

Map OSM data layers to specific schema names.

Parameters

- **osm_data** (*dict*) – Dictionary of OSM data where keys are layer names.
- **schema_names** (*list | dict | None*) – List of layers, dict mapping schema to layer, or None.

Returns

An iterable of (schema_name, data) tuples.

Return type

zip

Raises

KeyError – If a requested layer name is missing from osm_data.

preprocess_osm_layer

pydriosm.ios.utils.**preprocess_osm_layer**(*layer_data*, *layer_name*)

Preprocess layer data into a pandas DataFrame with WKT geometries.

Parameters

- **layer_data** (*list | pandas.Series | pandas.DataFrame*) – Layer data as a list of OGR features, a Series, or a DataFrame.
- **layer_name** (*str*) – Name of the layer for column labeling.

Returns

Processed DataFrame with serialized geometries.

Return type

pandas.DataFrame

Chapter 5

Modules

The following modules provide supporting functions and utilities for the subpackages *downloader*, *reader* and *ios*.

1anti-flashwhitewhite

errors	Define custom errors/exceptions.
utils	Provide various helper functions for use across the package.

5.1 errors

Define custom errors/exceptions.

5.1.1 Validation errors

1anti-flashwhitewhite

InvalidSubregionNameError(subregion_name[, msg])	Exception raised when an input *subregion_name* is not recognizable.
InvalidFileFormatError(osm_file_format[, ...])	Exception raised when an input *osm_file_format* is not recognizable.

InvalidSubregionNameError

class pydriosm.errors.**InvalidSubregionNameError**(*subregion_name*, *msg=None*)

Exception raised when an input *subregion_name* is not recognizable.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on a free download server
- **msg** (*int* / *None*) – index of optional messages, defaults to None; options include {1, 2}

Ivar
str subregion_name: name of a (sub)region available on a free download server

Ivar
int | None msg: index of optional messages; options include {1, 2}

Ivar
str: error message

Examples:

```
>>> from pydriosm.errors import InvalidSubregionNameError
>>> raise InvalidSubregionNameError(subregion_name='abc')
Traceback (most recent call last):
  ...
pydriosm.errors.InvalidSubregionNameError:
  `subregion_name='abc'` -> The input of `subregion_name` is not recognizable.
  Check the `.data_source`, or try another one instead.

>>> from pydriosm.downloader import GeofabrikDownloader, BBBikeDownloader
>>> gfd = GeofabrikDownloader()
>>> gfd.validate_subregion_name(subregion_name='birmingham')
Traceback (most recent call last):
  ...
pydriosm.errors.InvalidSubregionNameError:
  `subregion_name='birmingham'`
    1) `subregion_name` fails to match any in `<downloader>.valid_subregion_names`; or
    2) The queried (sub)region is not available on the free download server.

>>> bbd = BBBikeDownloader()
>>> bbd.validate_subregion_name(subregion_name='bham')
Traceback (most recent call last):
  ...
pydriosm.errors.InvalidSubregionNameError:
  `subregion_name='bham'` -> The input of `subregion_name` is not recognizable.
  Check the `.data_source`, or try another one instead.
```

InvalidFileFormatError

class pydriosm.errors.**InvalidFileFormatError**(*osm_file_format*, *valid_file_formats=None*)

Exception raised when an input *osm_file_format* is not recognizable.

Parameters

- **osm_file_format** (*str*) – file format/extension of the OSM data on the free download server
- **valid_file_formats** (*Iterable | None*) – filename extensions of the data files available on the free download server, defaults to None

Ivar
str osm_file_format: file format/extension of the OSM data available on the free download server

Ivar
int | None message: error message

Examples:

```
>>> from pydriosm.errors import InvalidFileFormatError
>>> raise InvalidFileFormatError(osm_file_format='abc')
Traceback (most recent call last):
  ...
pydriosm.errors.InvalidFileFormatError:
  `osm_file_format='abc'` -> The input `osm_file_format` is unidentifiable.

>>> from pydriosm.downloader import GeofabrikDownloader, BBBikeDownloader
>>> gfd = GeofabrikDownloader()
>>> gfd.validate_file_format(osm_file_format='abc')
Traceback (most recent call last):
  ...
pydriosm.errors.InvalidFileFormatError:
  `osm_file_format='abc'` -> The input `osm_file_format` is unidentifiable.
    Valid options include: {'.shp.zip', '.osm.pbf', '.osm.bz2'}.

>>> bbd = BBBikeDownloader()
>>> bbd.validate_file_format(osm_file_format='abc')
Traceback (most recent call last):
  ...
pydriosm.errors.InvalidFileFormatError:
  `osm_file_format='abc'` -> The input `osm_file_format` is unidentifiable.
    Valid options include: {'.shp.zip', '.geojson.xz', '.mapsforge-osm.zip', '.pbf', ...
```

5.1.2 Parse errors

1anti-flashwhitewhite

OtherTagsReformatError(other_tags)	Exception raised when errors occur in the process of parsing `other_tags` in a PBF data file.

OtherTagsReformatError

class pydriosm.errors.**OtherTagsReformatError**(*other_tags*)

Exception raised when errors occur in the process of parsing `other_tags` in a PBF data file.

Parameters

other_tags (*str | None*) – data of `'other_tags'` of a single feature in a PBF data file

Variables

- **other_tags** (*str | None*) – data of `'other_tags'` of a single feature in a PBF data file
- **message** (*str*) – error message

Examples:

```
>>> from pydriosm.errors import OtherTagsReformatError
>>> raise OtherTagsReformatError(other_tags='abc')
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
  ...
pydriosm.errors.OtherTagsReformatError:
  `other_tags='abc'` -> Failed to reformat the ...
```

5.2 utils

Provide various helper functions for use across the package.

5.2.1 General utilities

1anti-flashwhitewhite

first_unique(iterable)	Return unique items in an input iterable variable given the same order of presence.
check_json_engine([engine])	Check an available module used for loading JSON data.
remove_osm_file(path_to_file[, verbose])	Remove a downloaded OSM data file.

first_unique

pydriosm.utils.**first_unique**(*iterable*)

Return unique items in an input iterable variable given the same order of presence.

Parameters
iterable (*Iterable*) – iterable variable

Returns
unique items in the same order of presence as in the input

Return type
Generator[list]

Examples:

```
>>> from pydriosm.utils import first_unique

>>> list_example1 = [1, 2, 2, 3, 4, 5, 6, 6, 2, 3, 1, 6]
>>> list(first_unique(list_example1))
[1, 2, 3, 4, 5, 6]

>>> list_example2 = [6, 1, 2, 2, 3, 4, 5, 6, 6, 2, 3, 1]
>>> list(first_unique(list_example2))
[6, 1, 2, 3, 4, 5]
```

check_json_engine

pydriosm.utils.**check_json_engine**(*engine=None*)

Check an available module used for loading JSON data.

Parameters

engine (*str | None*) – name of a module for loading JSON data; when `engine=None` (default), use the built-in json module;

Returns

the module for loading JSON data

Type

types.ModuleType | None

Examples:

```
>>> from pydriosm.utils import check_json_engine
>>> import types
>>> result = check_json_engine()
>>> isinstance(result, types.ModuleType)
True
>>> result.__name__ == 'json'
True
```

remove_osm_file

pydriosm.utils.**remove_osm_file**(*path_to_file*, *verbose=True*)

Remove a downloaded OSM data file.

Parameters

- **path_to_file** (*str*) – absolute path to a downloaded OSM data file
- **verbose** (*bool*) – defaults to `True`

Examples:

```
>>> from pydriosm.utils import remove_osm_file
>>> from pyhelpers.dirs import cd
>>> import os

>>> path_to_pseudo_pbf_file = cd('tests/pseudo.osm.pbf')
>>> try:
...     open(path_to_pseudo_pbf_file, 'a').close()
... except OSError:
...     print('Failed to create the file.')
... else:
...     print('File created successfully.')
File created successfully.

>>> os.path.exists(path_to_pseudo_pbf_file)
True
>>> remove_osm_file(path_to_pseudo_pbf_file, verbose=True)
Deleting "tests\pseudo.osm.pbf" ... Done.
>>> os.path.exists(path_to_pseudo_pbf_file)
False
```

5.2.2 Check data pathnames

1anti-flashwhitewhite

cdd_geofabrik(*sub_dir[, mkdir, default_dir])	Change directory to osm_geofabrik\ and its subdirectories within a package.
cdd_bbbike(*sub_dir[, mkdir, default_dir])	Change directory to osm_bbbike\ and its subdirectories.

cdd_geofabrik

pydriosm.utils.**cdd_geofabrik**(**sub_dir*, *mkdir=False*, *default_dir='osm_geofabrik'*, ***kwargs*)

Change directory to osm_geofabrik\ and its subdirectories within a package.

Parameters

- **sub_dir** (*str | os.PathLike*) – name of directory; names of directories (and/or a filename)
- **mkdir** (*bool*) – whether to create a directory, defaults to False
- **default_dir** (*str*) – default folder name of the root directory for downloading data from Geofabrik, defaults to "osm_geofabrik"
- **kwargs** – [optional] parameters of pyhelpers.dir.cd()

Returns

an absolute path to a directory (or a file) under data_dir

Return type

str | os.PathLike

Examples:

```
>>> from pydriosm.utils import cdd_geofabrik
>>> import os

>>> os.path.relpath(cdd_geofabrik())
'osm_geofabrik'
```

cdd_bbbike

pydriosm.utils.**cdd_bbbike**(**sub_dir*, *mkdir=False*, *default_dir='osm_bbbike'*, ***kwargs*)

Change directory to osm_bbbike\ and its subdirectories.

Parameters

- **sub_dir** (*str*) – name of directory; names of directories (and/or a filename)
- **mkdir** (*bool*) – whether to create a directory, defaults to False
- **default_dir** (*str*) – default folder name of the root directory for downloading data from BBBike, defaults to "osm_bbbike"
- **kwargs** – [optional] parameters of pyhelpers.dir.cd()

Returns
an absolute path to a directory (or a file) under `data_dir`

Return type
str

Examples:

```
>>> from pydriosm.utils import cdd_bbbike
>>> import os

>>> os.path.relpath(cdd_bbbike())
'osm_bbbike'
```

5.2.3 Data processing utilities

1anti-flashwhitewhite

get_layer_name(stem)	Extract the core OSM layer name from a file stem or layer string.
find_matched_layer_names(layer_names, ...[, ...])	Find corresponding OSM layer names by identifying common semantic roots.
merge_dicts_by_values(data_dict, mapping_dict)	Group and concatenate DataFrames from a dictionary based on a mapping.

get_layer_name

pydriosm.utils.**get_layer_name**(*stem*)

Extract the core OSM layer name from a file stem or layer string.

Uses a regular expression to strip Geofabrik-specific prefixes (`gis_osm_`) and suffixes (`_a_free_1`, `_free`) to return a clean category name.

Parameters
stem (*str*) – File stem of a shapefile or a raw layer name from a GeoPackage.

Returns
The cleaned layer name (e.g., 'waterways'), or `None` if no match is found.

Return type
str | None

Examples:

```
>>> from pydriosm.utils import get_layer_name
>>> get_layer_name("") is None
True
>>> get_layer_name("gis_osm_railways_free_1.shp")
'railways'
>>> get_layer_name("gis_osm_transport_a_free_1")
'transport'
>>> get_layer_name("gis_osm_protected_areas_a_free")
'protected_areas'
```

(continues on next page)

(continued from previous page)

```
>>> get_layer_name("gis_osm_water_a_free")
'water'
```

find_matched_layer_names

pydriosm.utils.**find_matched_layer_names**(*layer_names*, *available_layer_names*, *cutoff=0.4*)

Find corresponding OSM layer names by identifying common semantic roots.

This function attempts to match input strings to a list of available layer names by extracting the core "root" of the word (e.g., stripping Geofabrik prefixes and plural suffixes). If no direct root match is found, it falls back to fuzzy string matching.

Parameters

- **layer_names** (*str | list*) – One or more layer names to search for.
- **available_layer_names** (*list | collections.abc.Iterable*) – A list of valid layer names (e.g., from a GeoPackage).
- **cutoff** (*float*) – Similarity threshold for the fuzzy match fallback; defaults to 0.4.

Returns

A list of unique matched layer names.

Return type

list

Examples:

```
>>> from pydriosm.utils import find_matched_layer_names
>>> layers = ['gis_osm_water_a_free_1', 'gis_osm_roads_free_1', 'pois']
>>> find_matched_layer_names('water', layers)
['gis_osm_water_a_free_1']
>>> find_matched_layer_names(['road', 'water'], layers)
['gis_osm_roads_free_1', 'gis_osm_water_a_free_1']
```

merge_dicts_by_values

pydriosm.utils.**merge_dicts_by_values**(*data_dict*, *mapping_dict*)

Group and concatenate DataFrames from a dictionary based on a mapping.

This utility takes a dictionary of data (e.g., raw layers) and a mapping dictionary that defines categories. All DataFrames belonging to the same category are concatenated into a single GeoDataFrame/DataFrame.

Parameters

- **data_dict** (*dict*) – A dictionary where keys match those in mapping_dict and values are pandas/geopandas objects.
- **mapping_dict** (*dict*) – A dictionary mapping data keys to target category names.

Returns

A dictionary where keys are categories and values are concatenated DataFrames.

Return type

dict

Examples:

```
>>> from pydriosm.utils import merge_dicts_by_values
>>> import pandas as pd
>>> d1 = {'a': pd.DataFrame([1]), 'b': pd.DataFrame([2]), 'c': pd.DataFrame([3])}
>>> m1 = {'a': 'group_1', 'b': 'group_1', 'c': 'group_2'}
>>> merged = merge_dicts_by_values(d1, m1)
>>> merged['group_1']
   0
0  1
1  2
```

License

- **PyDriosm**
  - PyDriosm is licensed under GNU General Public License v3.0 or later (GPLv3+).
- **OpenStreetMap data**
  - The free OpenStreetMap data, which is used for the development of PyDriosm, is licensed under the Open Data Commons Open Database License (ODbL) by the OpenStreetMap Foundation (OSMF).
  - For more details about the use of the OpenStreetMap data, refer to the web page of Copyright and Licence.

Acknowledgement

The development of pydriosm, including the example code that demonstrates how to use the package, heavily relies on freely available OpenStreetMap data. The author would like to express sincere gratitude to all the OpenStreetMap contributors for their invaluable contributions in making this data accessible to the community.

Contributors

- Qian Fu

Python Module Index

p

Index

G

I

L

M

N

O

P

R

S

T

U

V

W