PyDriosm

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

Release 2.3.ob1

Qian Fu

Birmingham Centre for Railway Research and Education

School of Engineering

University of Birmingham

First release: **September 2019**

Last updated: **October 2023**

Table of Contents

List of Figures

Chapter 1

Installation

To install the latest release of PyDriosm from PyPI via pip:

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

To install the most recent version of PyDriosm hosted on GitHub:

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

Warning:

- Pip may fail to install the dependency package GDAL. In such a circumstance, try instead to install their .whl files, which can be downloaded from the web page of the archived "unofficial Windows binaries for Python extension packages" (by Christoph Gohlke) or a mirror site (by Erin Turnbull). For how to install a .whl file, see the answers to this StackOverflow question.

Note:

- If using a virtual environment, make sure it is activated.
- It is recommended to add pip install the option `--upgrade` (or `-U`) to ensure that you are getting the latest stable release of the package.
- Non-essential dependencies (e.g. GeoPandas) of PyDriosm are not enforced to be installed along with the installation of the package. This is intended to optimise the installation requirements. If a ModuleNotFoundError or an ImportError pops out when importing/running a function or a method, first try to install the module(s)/package(s) mentioned in the error message, and then try to import/run the function or method again.
- For more general instructions on the installation of Python packages, please refer to the official guide of Installing Packages.

To check whether PyDriosm has been correctly installed, try to import the package via an interpreter shell:

```
>>> import pydriosm

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

```
The latest version is: 2.3.0b1
```

Chapter 2

Sub-packages / modules

The package includes the following sub-packages and modules:

2.1 Sub-packages

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.

2.1.1 downloader

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

Download OSM data

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

GeofabrikDownloader

class pydriosm.downloader.**GeofabrikDownloader**(*download_dir=None*)

Download OSM data from Geofabrik free download server.

Parameters

download_dir (*str | os.PathLike[str] | 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** (*set*) – filename extensions of the data files available
- **download_index** (*pandas.DataFrame*) – 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

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 = 'osm_data\\geofabrik'**

Default download directory.

GeofabrikDownloader.DOWNLOAD_INDEX_URL

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

URL of the official download index.

GeofabrikDownloader.FILE_FORMATS

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

Valid file formats.

GeofabrikDownloader.LONG_NAME

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

Full name of the data resource.

GeofabrikDownloader.NAME

GeofabrikDownloader.**NAME = 'Geofabrik'**

Name of the free download server.

GeofabrikDownloader.URL

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

URL of the homepage to the free download server.

Methods

download_osm_data(subregion_names, ...[, ...])	Download OSM data (in a specific format) of one (or multiple) geographic (sub)region(s).
download_subregion_data(subregion_names, ...)	Download OSM data (in a specific file format) of all subregions (if available) for 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.
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_download_index([update, ...])	Get the official index of downloads for all available geographic (sub)regions.
get_raw_directory_index(url[, verbose])	Get a raw directory index (including download information of older file logs).
get_region_subregion_tier([update, ...])	Get region-subregion tier and all (sub)regions that have no subregions.
get_subregion_download_url(subregion_name, ...)	Get a download URL of a geographic (sub)region.
get_subregion_table(url[, verbose])	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.
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.

GeofabrikDownloader.download_osm_data

GeofabrikDownloader.**download_osm_data**(*subregion_names*, *osm_file_format*, *download_dir=None*, *update=False*, *confirmation_required=True*, *deep_retry=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 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 the method cdd()
- **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_retry** (*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'
>>> subrgn_names = ['london', 'rutland']  # Case-insensitive
>>> file_format = ".pbf"

>>> gfd.download_osm_data(subrgn_names, file_format, verbose=True)
To download .osm.pbf data of the following geographic (sub)region(s):
    Greater London
    Rutland
? [No]|Yes: yes
Downloading "greater-london-latest.osm.pbf"
    to "osm_data\geofabrik\europe\great-britain\england\greater-london\" ... Done.
Downloading "rutland-latest.osm.pbf"
    to "osm_data\geofabrik\europe\great-britain\england\rutland\" ... Done.

>>> len(gfd.data_paths)
2
>>> for fp in gfd.data_paths: print(os.path.basename(fp))
greater-london-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)
'osm_data\geofabrik'
>>> dwnld_dir = os.path.dirname(gfd.download_dir)

>>> # Download shapefiles of West Midlands (to a given directory "tests\osm_data")
>>> region_name = 'west midlands'  # Case-insensitive
>>> file_format = ".shp"
>>> new_dwnld_dir = "tests\osm_data"

>>> gfd.download_osm_data(region_name, file_format, new_dwnld_dir, verbose=True)
To download .shp.zip data of the following geographic (sub)region(s):
    West Midlands
? [No]|Yes: yes
Downloading "west-midlands-latest-free.shp.zip"
    to "tests\osm_data\west-midlands\" ... Done.
>>> len(gfd.data_paths)
3
>>> os.path.relpath(gfd.data_paths[-1])
'tests\osm_data\west-midlands\west-midlands-latest-free.shp.zip'

>>> # Now the `.download_dir` variable has changed to the given one
>>> os.path.relpath(gfd.download_dir) == new_dwnld_dir
True
>>> # while the `.cdd()` remains the default one
>>> os.path.relpath(gfd.cdd())
'osm_data\geofabrik'

>>> # Delete the above downloaded directories
>>> delete_dir([dwnld_dir, new_dwnld_dir], verbose=True)
To delete the following directories:
    "osm_data\" (Not empty)
    "tests\osm_data\" (Not empty)
? [No]|Yes: yes
Deleting "osm_data\" ... Done.
Deleting "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)
'tests\osm_data'

>>> # Download shapefiles of Great Britain (to the directory specified by instantiation)
>>> # (Note that .shp.zip data is not available for "Great Britain" for free download.)
>>> region_name = 'Great Britain'  # Case-insensitive
>>> file_format = ".shp"

>>> # By default, `deep_retry=False`
>>> gfd.download_osm_data(region_name, osm_file_format=file_format, verbose=True)
To download .shp.zip data of the following geographic (sub)region(s):
    Great Britain
? [No]|Yes: yes
No .shp.zip data is found for "Great Britain".
Try to download the data of its subregions instead
? [No]|Yes: yes
Downloading "england-latest-free.shp.zip"
    to "tests\osm_data\europe\great-britain\great-britain-shp-zip\" ... Done.
Downloading "scotland-latest-free.shp.zip"
    to "tests\osm_data\europe\great-britain\great-britain-shp-zip\" ... Done.
Downloading "wales-latest-free.shp.zip"
    to "tests\osm_data\europe\great-britain\great-britain-shp-zip\" ... Done.

>>> len(gfd.data_paths)
3

>>> # Now set `deep_retry=True`
>>> gfd.download_osm_data(region_name, file_format, verbose=True, deep_retry=True)
To download .shp.zip data of the following geographic (sub)region(s):
    Great Britain
? [No]|Yes: yes
No .shp.zip data is found for "Great Britain".
Try to download the data of its subregions instead
? [No]|Yes: yes
"scotland-latest-free.shp.zip" is already available at "tests\osm_data\europ...".
"wales-latest-free.shp.zip" is already available at "tests\osm_data\europe\...".
Downloading "bedfordshire-latest-free.shp.zip"
    to "tests\osm_data\europe\great-britain\great-britain-shp-zip\" ... Done.
...     ...
Downloading "west-yorkshire-latest-free.shp.zip"
    to "tests\osm_data\europe\great-britain\great-britain-shp-zip\" ... Done.
Downloading "wiltshire-latest-free.shp.zip"
    to "tests\osm_data\europe\great-britain\great-britain-shp-zip\" ... Done.
Downloading "worcestershire-latest-free.shp.zip"
    to "tests\osm_data\europe\great-britain\great-britain-shp-zip\" ... Done.

>>> # Check the file paths
>>> len(gfd.data_paths)
50
>>> # Check the current default `download_dir`
>>> os.path.relpath(gfd.download_dir)
'tests\osm_data'
>>> os.path.relpath(os.path.commonpath(gfd.data_paths))
'tests\osm_data\europe\great-britain\great-britain-shp-zip'

>>> # Delete all the downloaded files
>>> delete_dir(gfd.download_dir, verbose=True)
```

(continues on next page)

(continued from previous page)

```
To delete the directory "tests\osm_data\" (Not empty)
? [No]|Yes: yes
Deleting "tests\osm_data\" ... Done.
```

GeofabrikDownloader.download_subregion_data

GeofabrikDownloader.**download_subregion_data**(*subregion_names*, *osm_file_format*, *download_dir=None*, *deep=False*, *ret_download_path=False*, ***kwargs*)

Download OSM data (in a specific file format) of all subregions (if available) for one (or multiple) geographic (sub)region(s).

If no subregion data is available for the region(s) specified by `subregion_names`, then the data of `subregion_names` would be downloaded only.

Parameters

- **subregion_names** (*str | list*) – name of a geographic (sub)region (or names of multiple geographic (sub)regions) 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 the method `cdd()`
- **deep** (*bool*) – whether to try to search for subregions of subregion(s), 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 pydriosm.GeofabrikDownloader.download_osm_data()

Returns

the path(s) to the downloaded file(s) when `ret_download_path=True`

Return type

list | str

Examples:

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

>>> gfd = GeofabrikDownloader()

>>> subrgn_names = ['rutland', 'west yorkshire']
>>> file_format = ".pbf"
>>> dwnld_dir = "tests\osm_data"

>>> gfd.download_subregion_data(subrgn_names, file_format, dwnld_dir, verbose=True)
```

(continues on next page)

(continued from previous page)

```
To download .osm.pbf data of the following geographic (sub)region(s):
    Rutland
    West Yorkshire
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf"
    to "tests\osm_data\rutland\" ... Done.
Downloading "west-yorkshire-latest.osm.pbf"
    to "tests\osm_data\west-yorkshire\" ... Done.

>>> len(gfd.data_paths)
2
>>> for fp in gfd.data_paths: print(os.path.relpath(fp))
tests\osm_data\rutland\rutland-latest.osm.pbf
tests\osm_data\west-yorkshire\west-yorkshire-latest.osm.pbf

>>> # Delete "tests\osm_data\rutland-latest.osm.pbf"
>>> rutland_dir = os.path.dirname(gfd.data_paths[0])
>>> delete_dir(rutland_dir, confirmation_required=False, verbose=True)
Deleting "tests\osm_data\rutland\" ... Done.

>>> # Try to download data given another list which also includes 'West Yorkshire'
>>> subrgn_names = ['west midlands', 'west yorkshire']

>>> # Set `ret_download_path=True`
>>> dwnld_file_pathnames = gfd.download_subregion_data(
...     subrgn_names, file_format, dwnld_dir, verbose=True, ret_download_path=True)
"west-midlands-latest.osm.pbf" is already available
    at "tests\osm_data\west-midlands\".
To download .osm.pbf data of the following geographic (sub)region(s):
    West Midlands
? [No]|Yes: yes
Downloading "west-midlands-latest.osm.pbf"
    to "tests\osm_data\west-midlands\" ... Done.

>>> len(gfd.data_paths)  # The pathname of the newly downloaded file is added
3
>>> len(dwnld_file_pathnames)
2
>>> for fp in dwnld_file_pathnames: print(os.path.relpath(fp))
tests\osm_data\west-midlands\west-midlands-latest.osm.pbf
tests\osm_data\west-yorkshire\west-yorkshire-latest.osm.pbf

>>> # Update (or re-download) the existing data file by setting `update=True`
>>> gfd.download_subregion_data(
...     subrgn_names, file_format, download_dir=dwnld_dir, update=True, verbose=True)
"west-midlands-latest.osm.pbf" is already available
    at "tests\osm_data\west-midlands\".
"west-yorkshire-latest.osm.pbf" is already available
    at "tests\osm_data\west-yorkshire\".
To update the .osm.pbf data of the following geographic (sub)region(s):
    West Midlands
    West Yorkshire
? [No]|Yes: yes
Updating "west-midlands-latest.osm.pbf"
    at "tests\osm_data\west-midlands\" ... Done.
Updating "west-yorkshire-latest.osm.pbf"
    at "tests\osm_data\west-yorkshire\" ... Done.

>>> # To download the PBF data of all available subregions of England
```

(continues on next page)

(continued from previous page)

```
>>> subrgn_name = 'England'

>>> dwnld_file_pathnames = gfd.download_subregion_data(
...     subrgn_name, file_format, download_dir=dwnld_dir, update=True, verbose=True,
...     ret_download_path=True)
"west-midlands-latest.osm.pbf" is already available
    at "tests\osm_data\west-midlands\".
"west-yorkshire-latest.osm.pbf" is already available
    at "tests\osm_data\west-yorkshire\".
To download/update the .osm.pbf data of the following geographic (sub)region(s):
    Bedfordshire
    Berkshire
    Bristol
    ...
    West Midlands
    ...
    West Yorkshire
    Wiltshire
    Worcestershire
? [No]|Yes: yes
Downloading "bedfordshire-latest.osm.pbf"
    to "tests\osm_data\bedfordshire\" ... Done.
Downloading "berkshire-latest.osm.pbf"
    to "tests\osm_data\berkshire\" ... Done.
Downloading "bristol-latest.osm.pbf"
    to "tests\osm_data\bristol\" ... Done.
...
    ...
Updating "west-midlands-latest.osm.pbf"
    at "tests\osm_data\west-midlands\" ... Done.
...
    ...
Updating "west-yorkshire-latest.osm.pbf"
    at "tests\osm_data\west-yorkshire\" ... Done.
Downloading "wiltshire-latest.osm.pbf"
    to "tests\osm_data\wiltshire\" ... Done.
Downloading "worcestershire-latest.osm.pbf"
    to "tests\osm_data\worcestershire\" ... Done.

>>> len(dwnld_file_pathnames)
47
>>> os.path.commonpath(dwnld_file_pathnames) == gfd.download_dir
True

>>> # Delete the download directory and 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 the method cdd()
- **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

>>> # Specify a download directory
>>> dwnld_dir = "tests\osm_data"

>>> gfd = GeofabrikDownloader(download_dir=dwnld_dir)

>>> subregion_name = 'london'
>>> osm_file_format = ".pbf"

>>> # Download the PBF data of London (to the default directory)
>>> gfd.download_osm_data(subregion_name, osm_file_format, 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\europe\great-britain\england\greater-london\"...Done.

>>> # Check whether the PBF data file exists; `ret_file_path` is by default `False`
>>> pbf_exists = gfd.file_exists(subregion_name, osm_file_format)
>>> pbf_exists  # If the data file exists at the default directory
True
```

(continues on next page)

(continued from previous page)

```
>>> # Set `ret_file_path=True`
>>> path_to_pbf = gfd.file_exists(subregion_name, osm_file_format, ret_file_path=True)
>>> os.path.relpath(path_to_pbf)  # If the data file exists at the default directory
'tests\osm_data\europe\great-britain\england\greater-london\greater-londo...'

>>> # Remove the download directory:
>>> delete_dir(dwnld_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.get_catalogue

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

Get a catalogue (index) of all available downloads.

Similar to the method *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

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.core.frame.DataFrame
>>> dwnld_catalog.head()
              subregion  ...                                          .osm.bz2
0                Africa  ...  https://download.geofabrik.de/africa-latest.os...
1            Antarctica  ...  https://download.geofabrik.de/antarctica-lates...
2                  Asia  ...  https://download.geofabrik.de/asia-latest.osm.bz2
```

(continues on next page)

(continued from previous page)

```
3  Australia and Oceania  ...  https://download.geofabrik.de/australia-oceani...
4        Central America  ...  https://download.geofabrik.de/central-america-...
[5 rows x 6 columns]

>>> dwnld_catalog.columns.to_list()
['subregion',
 'subregion-url',
 '.osm.pbf',
 '.osm.pbf-size',
 '.shp.zip',
 '.osm.bz2']
```

Note:

- Information of LondonEnfield is not directly available from the web page of Greater London.
- Two subregions have the same name 'Georgia': EuropeGeorgia and USGeorgia; In the latter case, a suffix ' (US)' is appended to the name in the table.

GeofabrikDownloader.get_continent_tables

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

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`

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
```

(continues on next page)

(continued from previous page)

```
>>> 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
>>> asia_table = continent_tables['Asia']
>>> len(asia_table) >= 39
True
>>> asia_table.head()
     subregion  ...                                            .osm.bz2
0  Afghanistan  ...  https://download.geofabrik.de/asia/afghanistan...
1      Armenia  ...  https://download.geofabrik.de/asia/armenia-lat...
2   Azerbaijan  ...  https://download.geofabrik.de/asia/azerbaijan-...
3   Bangladesh  ...  https://download.geofabrik.de/asia/bangladesh-...
4       Bhutan  ...  https://download.geofabrik.de/asia/bhutan-late...
[5 rows x 6 columns]

>>> asia_table.columns.to_list()
['subregion',
 'subregion-url',
 '.osm.pbf',
 '.osm.pbf-size',
 '.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
>>> subrgn_name, file_format = 'london', ".pbf"
>>> default_fn = gfd.get_default_filename(subrgn_name, file_format)
>>> default_fn
'greater-london-latest.osm.pbf'

>>> # Default filename of the shapefile data of Great Britain
>>> subrgn_name, file_format = 'britain', ".shp"
>>> default_fn = gfd.get_default_filename(subrgn_name, file_format)
No .shp.zip data is available to download for Great Britain.
>>> default_fn is None
True
```

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
>>> subrgn_name, file_format = 'london', ".pbf"
```

(continues on next page)

(continued from previous page)

```
>>> pathname, filename = gfd.get_default_pathname(subrgn_name, file_format)
>>> os.path.relpath(os.path.dirname(pathname))
'osm_data\geofabrik\europe\great-britain\england\greater-london'
>>> filename
'greater-london-latest.osm.pbf'
```

GeofabrikDownloader.get_download_index

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

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

Similar to the method *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`

Returns

the official index of all downloads

Return type

pandas.DataFrame | None

Examples:

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

>>> gfd = GeofabrikDownloader()

>>> # Official index of all available downloads
>>> geofabrik_dwnld_idx = gfd.get_download_index()
>>> type(geofabrik_dwnld_idx)
pandas.core.frame.DataFrame
>>> geofabrik_dwnld_idx.head()
            id  ...                                            updates
0  afghanistan  ...  https://download.geofabrik.de/asia/afghanistan...
1       africa  ...       https://download.geofabrik.de/africa-updates
2      albania  ...  https://download.geofabrik.de/europe/albania-u...
3      alberta  ...  https://download.geofabrik.de/north-america/ca...
4      algeria  ...  https://download.geofabrik.de/africa/algeria-u...
[5 rows x 13 columns]

>>> geofabrik_dwnld_idx.columns.to_list()
['id',
 'parent',
 'iso3166-1:alpha2',
 'name',
 'iso3166-2',
```

(continues on next page)

(continued from previous page)

```
'geometry',
'.osm.pbf',
'.osm.bz2',
'.shp.zip',
'pbf-internal',
'history',
'taginfo',
'updates']
```

GeofabrikDownloader.get_raw_directory_index

classmethod GeofabrikDownloader.**get_raw_directory_index**(*url*, *verbose=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)
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to `False`

Returns

information of raw directory index

Return type

pandas.DataFrame | None

Examples:

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

>>> gfd = GeofabrikDownloader()

>>> homepage_url = gfd.URL
>>> homepage_url
'https://download.geofabrik.de/'
>>> raw_index = gfd.get_raw_directory_index(homepage_url, verbose=True)
Collecting the raw directory index on 'https://download.geofabrik.de/' ... Failed.
No raw directory index is available on the web page.
>>> raw_index is None
True

>>> great_britain_url = 'https://download.geofabrik.de/europe/great-britain.html'
>>> raw_index = gfd.get_raw_directory_index(great_britain_url)
>>> type(raw_index)
pandas.core.frame.DataFrame
>>> raw_index.columns.tolist()
['file', 'date', 'size', 'metric_file_size', 'url']
```

GeofabrikDownloader.get_region_subregion_tier

GeofabrikDownloader.**get_region_subregion_tier**(*update=False*, *confirmation_required=True*, *verbose=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`

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 tier, and all regions that have no subregions
>>> rgn_subrgn_tier, no_subrgn_list = gfd.get_region_subregion_tier()
>>> type(rgn_subrgn_tier)
dict
>>> # Keys of the region-subregion tier
>>> list(rgn_subrgn_tier.keys())
['Africa',
 'Antarctica',
 'Asia',
 'Australia and Oceania',
 'Central America',
 'Europe',
 'North America',
 'South America']

>>> type(no_subrgn_list)
list
>>> # Example: five regions that have no subregions
>>> no_subrgn_list[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*)

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`

Returns

name and URL of the subregion

Return type

Tuple[str, str | None]

Examples:

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

>>> gfd = GeofabrikDownloader()

>>> subrgn_name = 'England'
>>> file_format = ".pbf"
>>> valid_name, dwnld_link = gfd.get_subregion_download_url(subrgn_name, file_format)
>>> valid_name  # The name of the subregion on the free downloader server
'England'
>>> dwnld_link  # The URL of the PBF data file
'https://download.geofabrik.de/europe/great-britain/england-latest.osm.pbf'

>>> subrgn_name = 'britain'
>>> file_format = ".shp"
>>> valid_name, dwnld_link = gfd.get_subregion_download_url(subrgn_name, file_format)
>>> valid_name
'Great Britain'
>>> dwnld_link 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*)

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

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
>>> homepage = gfd.get_subregion_table(url=gfd.URL)
>>> homepage
               subregion  ...                                          .osm.bz2
0                 Africa  ...  https://download.geofabrik.de/africa-latest.os...
1             Antarctica  ...  https://download.geofabrik.de/antarctica-lates...
2                   Asia  ...  https://download.geofabrik.de/asia-latest.osm.bz2
3  Australia and Oceania  ...  https://download.geofabrik.de/australia-oceani...
4        Central America  ...  https://download.geofabrik.de/central-america-...
5                 Europe  ...  https://download.geofabrik.de/europe-latest.os...
6          North America  ...  https://download.geofabrik.de/north-america-la...
7          South America  ...  https://download.geofabrik.de/south-america-la...
[8 rows x 6 columns]

>>> homepage.columns.to_list()
['subregion',
 'subregion-url',
 '.osm.pbf',
 '.osm.pbf-size',
 '.shp.zip',
 '.osm.bz2']

>>> # Download information about 'Great Britain'
>>> great_britain_url = 'https://download.geofabrik.de/europe/great-britain.html'
>>> great_britain = gfd.get_subregion_table(great_britain_url)
>>> great_britain
  subregion  ...                                          .osm.bz2
0   England  ...  https://download.geofabrik.de/europe/great-bri...
1  Scotland  ...  https://download.geofabrik.de/europe/great-bri...
2     Wales  ...  https://download.geofabrik.de/europe/great-bri...
[3 rows x 6 columns]

>>> # Download information about 'Antarctica'
>>> antarctica_url = 'https://download.geofabrik.de/antarctica.html'
>>> antarctica = gfd.get_subregion_table(antarctica_url, verbose=True)
Compiling information about subregions of "Antarctica" ... Failed.
```

(continues on next page)

(continued from previous page)

```
>>> antarctica is None
True

>>> # To get more information about the above failure, set `verbose=2`
>>> antarctica2 = gfd.get_subregion_table(antarctica_url, verbose=2)
Compiling information about subregions of "Antarctica" ... Failed.
No subregion data is available for "Antarctica" on Geofabrik's free download server.
>>> antarctica2 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 the method *get_region_subregion_tier()*.

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
```

(continues on next page)

(continued from previous page)

```
>>> # Names of subregions of Great Britain
>>> gb_subrgn_names = gfd.get_subregions('britain')
>>> len(gb_subrgn_names) == 3
True

>>> # Names of all subregions of Great Britain's subregions
>>> gb_subrgn_names_ = gfd.get_subregions('britain', 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 the method `cdd()`
- **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()

>>> subrgn_name = 'london'
>>> file_format = "pbf"

>>> # valid subregion name, filename, download url and absolute file path
>>> info1 = gfd.get_valid_download_info(subrgn_name, file_format)
>>> valid_subrgn_name, pbf_filename, dwnld_url, path_to_pbf = info1

>>> valid_subrgn_name
'Greater London'
>>> pbf_filename
```

(continues on next page)

(continued from previous page)

```
'greater-london-latest.osm.pbf'
>>> os.path.dirname(dwnld_url)
'https://download.geofabrik.de/europe/great-britain/england'
>>> os.path.relpath(os.path.dirname(path_to_pbf))
'osm_data\geofabrik\europe\great-britain\england\greater-london'

>>> # Specify a new directory for downloaded data
>>> dwnld_dir = "tests\osm_data"

>>> info2 = gfd.get_valid_download_info(subrgn_name, file_format, dwnld_dir)
>>> _, _, _, path_to_pbf2 = info2

>>> os.path.relpath(os.path.dirname(path_to_pbf2))
'tests\osm_data\greater-london'

>>> gfd_ = GeofabrikDownloader(download_dir=dwnld_dir)

>>> info3 = gfd_.get_valid_download_info(subrgn_name, file_format)
>>> _, _, _, path_to_pbf3 = info3

>>> os.path.relpath(os.path.dirname(path_to_pbf3))
'tests\osm_data\europe\great-britain\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
```

(continues on next page)

(continued from previous page)

```
>>> valid_subrgn_names = gfd.get_valid_subregion_names()
>>> type(valid_subrgn_names)
set
```

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 the method cdd()
- **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()

>>> subrgn_name = 'london'
>>> file_format = ".pbf"

>>> # Default download directory (if the requested data file is not available)
>>> dwnld_dir = gfd.specify_sub_download_dir(subrgn_name, file_format)
>>> os.path.dirname(os.path.relpath(dwnld_dir))
'osm_data\geofabrik\europe\great-britain\england\greater-london'

>>> # When a download directory is specified
>>> dwnld_dir = "tests\osm_data"

>>> subrgn_name = 'britain'
>>> file_format = ".shp"

>>> dwnld_pathname = gfd.specify_sub_download_dir(subrgn_name, file_format, dwnld_dir)
```

(continues on next page)

(continued from previous page)

```
>>> os.path.relpath(dwnld_pathname)
'tests\osm_data\great-britain-shp-zip'

>>> gfd_ = GeofabrikDownloader(download_dir=dwnld_dir)
>>> dwnld_pathname_ = gfd_.specify_sub_download_dir(subrgn_name, file_format)
>>> os.path.relpath(dwnld_pathname_)
'tests\osm_data\europe\great-britain\great-britain-shp-zip'
```

GeofabrikDownloader.validate_file_format

GeofabrikDownloader.**validate_file_format**(*osm_file_format*, *valid_file_formats=None*, *raise_err=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_file_formats** (*Iterable*) – fil extensions of the data available on a free download server
- **raise_err** (*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()

>>> input_file_format = ".pbf"
>>> valid_file_format = gfd.validate_file_format(osm_file_format=input_file_format)
>>> valid_file_format
'.osm.pbf'

>>> input_file_format = "shp"
>>> valid_file_format = gfd.validate_file_format(osm_file_format=input_file_format)
>>> valid_file_format
'.shp.zip'
```

GeofabrikDownloader.validate_subregion_name

GeofabrikDownloader.**validate_subregion_name**(*subregion_name*, *valid_subregion_names=None*, *raise_err=True*, ***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_subregion_names** (*Iterable*) – names of all (sub)regions available on a free download server
- **raise_err** (*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()

>>> input_subrgn_name = 'london'
>>> valid_subrgn_name = gfd.validate_subregion_name(subregion_name=input_subrgn_name)
>>> valid_subrgn_name
'Greater London'

>>> input_subrgn_name = 'https://download.geofabrik.de/europe/great-britain.html'
>>> valid_subrgn_name = gfd.validate_subregion_name(subregion_name=input_subrgn_name)
>>> valid_subrgn_name
'Great Britain'
```

BBBikeDownloader

class pydriosm.downloader.**BBBikeDownloader**(*download_dir=None*)

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

CITIES_COORDS_URL	URL of coordinates of all the available cities.
CITIES_URL	URL of a list of cities that are available on the free download server.
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.CITIES_COORDS_URL

BBBikeDownloader.**CITIES_COORDS_URL = 'https://raw.githubusercontent.com/wosch/bbbike-world/world/etc/cities.csv'**

URL of coordinates of all the available cities.

BBBikeDownloader.CITIES_URL

BBBikeDownloader.**CITIES_URL = 'https://raw.githubusercontent.com/wosch/bbbike-world/world/etc/cities.txt'**

URL of a list of cities that are available on the free download server.

BBBikeDownloader.DEFAULT_DOWNLOAD_DIR

BBBikeDownloader.**DEFAULT_DOWNLOAD_DIR = 'osm_data\\bbbike'**

Default download directory.

BBBikeDownloader.FILE_FORMATS

BBBikeDownloader.**FILE_FORMATS = {'.csv.xz', '.garmin-onroad-latin1.zip', '.garmin-onroad.zip', '.garmin-opentopo.zip', '.garmin-osm.zip', '.geojson.xz', '.gz', '.mapsforge-osm.zip', '.pbf', '.shp.zip', '.svg-osm.zip'}**

Valid file formats.

BBBikeDownloader.LONG_NAME

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

Full name of the data resource.

BBBikeDownloader.NAME

BBBikeDownloader.**NAME = 'BBBike'**

Name of the free downloader server.

BBBikeDownloader.URL

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

URL of the homepage to the free download server.

Methods

download_osm_data(subregion_names, ...[, ...])	Download OSM data (of a specific file format) of one (or multiple) geographic (sub)region(s).
download_subregion_data(subregion_name[, ...])	Download OSM data of all available formats for a geographic (sub)region.
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.
get_catalogue([update, ...])	Get a dict-type index of available formats, data types and a download catalogue.
get_coordinates_of_cities([update, ...])	Get location information of all cities available on the download server.
get_names_of_cities([update, ...])	Get the names of all the available cities.
get_subregion_catalogue(subregion_name[, ...])	Get a download catalogue of OSM data available for a given geographic (sub)region.
get_subregion_download_url(subregion_name, ...)	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.
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.

BBBikeDownloader.download_osm_data

BBBikeDownloader.**download_osm_data**(*subregion_names*, *osm_file_format*, *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_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 the method `cdd()`

- **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
>>> subrgn_name = 'London'
>>> file_format = "pbf"

>>> bbd.download_osm_data(subrgn_name, file_format, verbose=True)
To download .pbf data of the following geographic (sub)region(s):
    London
? [No]|Yes: yes
Downloading "London.osm.pbf"
    to "osm_data\bbbike\london\" ... Done.

>>> len(bbd.data_paths)
1
>>> os.path.relpath(bbd.data_paths[0])
'osm_data\bbbike\london\London.osm.pbf'

>>> london_dwnld_dir = os.path.relpath(bbd.download_dir)
>>> london_dwnld_dir
'osm_data\bbbike'

>>> # Download PBF data of Leeds and Birmingham to a given directory
>>> subrgn_names = ['leeds', 'birmingham']
>>> dwnld_dir = "tests\osm_data"

>>> dwnld_paths = bbd.download_osm_data(
...     subrgn_names, file_format, dwnld_dir, verbose=True, ret_download_path=True)
To download .pbf data of the following geographic (sub)region(s):
    Leeds
```

(continues on next page)

(continued from previous page)

```
    Birmingham
? [No]|Yes: yes
Downloading "Leeds.osm.pbf"
    to "tests\osm_data\leeds\" ... Done.
Downloading "Birmingham.osm.pbf"
    to "tests\osm_data\birmingham\" ... Done.
>>> len(dwnld_paths)
2
>>> len(bbd.data_paths)
3
>>> os.path.relpath(bbd.download_dir) == os.path.relpath(dwnld_dir)
True
>>> os.path.relpath(os.path.commonpath(dwnld_paths))
'tests\osm_data'

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

BBBikeDownloader.download_subregion_data

BBBikeDownloader.**download_subregion_data**(*subregion_name*, *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 all available formats for a geographic (sub)region.

Parameters

- **subregion_name** (*str*) – name of a (sub)region available on BBBike free download server
- **download_dir** (*str | None*) – directory where the downloaded file is saved, defaults to None
- **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`
- **kwargs** – optional parameters of pyhelpers.ops.download_file_from_url()

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 the BBBike OSM data of Birmingham (to the default download directory)
>>> subrgn_name = 'birmingham'

>>> bbd.download_subregion_data(subrgn_name, verbose=True)
To download all available BBBike OSM data of Birmingham
? [No]|Yes: yes
Downloading:
    Birmingham.osm.pbf ... Done.
    Birmingham.osm.gz ... Done.
    Birmingham.osm.shp.zip ... Done.
    Birmingham.osm.garmin-onroad-latin1.zip ... Done.
    Birmingham.osm.garmin-osm.zip ... Done.
    Birmingham.osm.garmin-ontrail-latin1.zip ... Done.
    Birmingham.osm.geojson.xz ... Done.
    Birmingham.osm.svg-osm.zip ... Done.
    Birmingham.osm.mapsforge-osm.zip ... Done.
    Birmingham.osm.garmin-opentopo-latin1.zip ... Done.
    Birmingham.osm.mbtiles-openmaptiles.zip ... Done.
    Birmingham.osm.csv.xz ... Done.
    Birmingham.poly ... Done.
    CHECKSUM.txt ... Done.
Check out the downloaded OSM data at "osm_data\bbbike\birmingham\".

>>> len(bbd.data_paths)
14
>>> os.path.relpath(os.path.commonpath(bbd.data_paths))
'osm_data\bbbike\birmingham'
>>> os.path.relpath(bbd.download_dir)
'osm_data\bbbike'
>>> bham_dwnld_dir = os.path.dirname(bbd.download_dir)

>>> # Download the BBBike OSM data of Leeds (to a given download directory)
>>> subrgn_name = 'leeds'
>>> dwnld_dir = "tests\osm_data"

>>> dwnld_paths = bbd.download_subregion_data(
...     subrgn_name, download_dir=dwnld_dir, verbose=True, ret_download_path=True)
To download all available BBBike OSM data of Leeds
? [No]|Yes: yes
Downloading:
    Leeds.osm.pbf ... Done.
```

(continues on next page)

(continued from previous page)

```
    Leeds.osm.gz ... Done.
    Leeds.osm.shp.zip ... Done.
    Leeds.osm.garmin-onroad-latin1.zip ... Done.
    Leeds.osm.garmin-osm.zip ... Done.
    Leeds.osm.garmin-ontrail-latin1.zip ... Done.
    Leeds.osm.geojson.xz ... Done.
    Leeds.osm.svg-osm.zip ... Done.
    Leeds.osm.mapsforge-osm.zip ... Done.
    Leeds.osm.garmin-opentopo-latin1.zip ... Done.
    Leeds.osm.mbtiles-openmaptiles.zip ... Done.
    Leeds.osm.csv.xz ... Done.
    Leeds.poly ... Done.
    CHECKSUM.txt ... Done.
Check out the downloaded OSM data at "tests\osm_data\leeds\".

>>> # Now the variable `.download_dir` has changed to `dwnld_dir`
>>> leeds_dwnld_dir = bbd.download_dir
>>> os.path.relpath(leeds_dwnld_dir) == dwnld_dir
True

>>> len(dwnld_paths)
14
>>> len(bbd.data_paths)  # New pathnames have been added to `.data_paths`
28
>>> os.path.relpath(os.path.commonpath(dwnld_paths))
'tests\osm_data\leeds'

>>> # Delete the download directories
>>> delete_dir([bham_dwnld_dir, leeds_dwnld_dir], verbose=True)
To delete the following directories:
    "osm_data\" (Not empty)
    "tests\osm_data\" (Not empty)
? [No]|Yes: yes
Deleting "osm_data\" ... Done.
Deleting "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 the method `cdd()`
- **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()

>>> subrgn_name = 'birmingham'
>>> file_format = ".pbf"
>>> dwnld_dir = "tests\osm_data"

>>> # Check whether the PBF data file exists; `ret_file_path` is by default `False`
>>> pbf_exists = bbd.file_exists(subrgn_name, file_format, dwnld_dir)
>>> pbf_exists
False

>>> # Download the PBF data of Birmingham (to the default directory)
>>> bbd.download_osm_data(subrgn_name, file_format, dwnld_dir, verbose=True)
To download .pbf data of the following geographic (sub)region(s):
    Birmingham
? [No]|Yes: yes
Downloading "Birmingham.osm.pbf"
    to "tests\osm_data\birmingham\" ... Done.

>>> bbd.file_exists(subrgn_name, file_format, dwnld_dir)
True

>>> # Set `ret_file_path=True`
>>> pbf_pathname = bbd.file_exists(subrgn_name, file_format, ret_file_path=True)
>>> os.path.relpath(pbf_pathname)
'tests\osm_data\birmingham\Birmingham.osm.pbf'

>>> os.path.relpath(dwnld_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(subrgn_name, file_format, dwnld_dir)
False
```

BBBikeDownloader.get_catalogue

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

Returns

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

Return type

dict | None

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
```

BBBikeDownloader.get_coordinates_of_cities

classmethod BBBikeDownloader.**get_coordinates_of_cities**(*update=False*, *confirmation_required=True*, *verbose=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`

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

Return type
pandas.DataFrame | None

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbd = BBBikeDownloader()

>>> # Location information of BBBike cities
>>> coords_of_cities = bbd.get_coordinates_of_cities()

>>> type(coords_of_cities)
pandas.core.frame.DataFrame
>>> coords_of_cities.head()
          City  ... ur_latitude
0       Aachen  ...       50.99
1       Aarhus  ...      56.287
2     Adelaide  ...     -34.753
3  Albuquerque  ...     35.2173
4   Alexandria  ...       31.34
[5 rows x 13 columns]

>>> coords_of_cities.columns.to_list()
['city',
 'real_name',
 'pref._language',
 'local_language',
 'country',
 'area_or_continent',
 'population',
 'step',
 'other_cities',
 'll_longitude',
 'll_latitude',
 'ur_longitude',
 'ur_latitude']
```

BBBikeDownloader.get_names_of_cities

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

Get the names of all the available cities.

This can be an alternative to the method *get_valid_subregion_names()*.

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
list of names of cities available on BBBike free download server

Return type
list | None

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbd = BBBikeDownloader()

>>> # A list of BBBike cities' names
>>> bbbike_cities = bbd.get_names_of_cities()
>>> type(bbbike_cities)
list
```

BBBikeDownloader.get_subregion_catalogue

BBBikeDownloader.**get_subregion_catalogue**(*subregion_name*, *confirmation_required=True*, *verbose=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
- **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
a catalogues for subregion downloads

Return type
pandas.DataFrame | None

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbd = BBBikeDownloader()

>>> subrgn_name = 'birmingham'

>>> # A download catalogue for Leeds
>>> bham_dwnld_cat = bbd.get_subregion_catalogue(subrgn_name, verbose=True)
To compile data of a download catalogue for "Birmingham"
```

(continues on next page)

(continued from previous page)

```
? [No]|Yes: yes
Compiling the data ... Done.
>>> type(bham_dwnld_cat)
pandas.core.frame.DataFrame
>>> bham_dwnld_cat.columns.tolist()
['filename', 'url', 'data_type', 'size', 'last_update']
```

BBBikeDownloader.get_subregion_download_url

BBBikeDownloader.**get_subregion_download_url**(*subregion_name*, *osm_file_format*, ***kwargs*)

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

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()

>>> subrgn_name = 'birmingham'
>>> file_format = "pbf"

>>> # Get a valid subregion name and its download URL
>>> subrgn_name_, dwnld_url = bbd.get_subregion_download_url(subrgn_name, file_format)
>>> subrgn_name_
'Birmingham'
>>> dwnld_url
'https://download.bbbike.org/osm/bbbike/Birmingham/Birmingham.osm.pbf'

>>> file_format = "csv.xz"
>>> subrgn_name_, dwnld_url = bbd.get_subregion_download_url(subrgn_name, file_format)

>>> subrgn_name_
'Birmingham'
>>> dwnld_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*)

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

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
>>> subrgn_idx = bbd.get_subregion_index()

>>> type(subrgn_idx)
pandas.core.frame.DataFrame
>>> subrgn_idx.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 the method cdd()

- **kwargs** – [optional] parameters of pyhelpers.dirs.cd(), including `mkdir`

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

Return type
tuple

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader
>>> import os

>>> bbd = BBBikeDownloader()

>>> subrgn_name = 'birmingham'
>>> file_format = "pbf"

>>> # valid subregion name, filename, download url and absolute file path
>>> info = bbd.get_valid_download_info(subrgn_name, file_format)
>>> valid_subrgn_name, pbf_filename, dwnld_url, pbf_pathname = info

>>> valid_subrgn_name
'Birmingham'
>>> pbf_filename
'Birmingham.osm.pbf'
>>> dwnld_url
'https://download.bbbike.org/osm/bbbike/Birmingham/Birmingham.osm.pbf'
>>> os.path.relpath(pbf_pathname)
'osm_data\bbbike\birmingham\Birmingham.osm.pbf'

>>> # Create a new instance with a given download directory
>>> bbd = BBBikeDownloader(download_dir="tests\osm_data")
>>> _, _, _, pbf_pathname = bbd.get_valid_download_info(subrgn_name, file_format)

>>> os.path.relpath(pbf_pathname)
'tests\osm_data\birmingham\Birmingham.osm.pbf'
```

BBBikeDownloader.get_valid_subregion_names

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

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

This can be an alternative to the method *get_names_of_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`

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
>>> subrgn_names = bbd.get_valid_subregion_names()

>>> type(subrgn_names)
list
```

BBBikeDownloader.validate_file_format

BBBikeDownloader.**validate_file_format**(*osm_file_format*, *valid_file_formats=None*, *raise_err=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_file_formats** (*Iterable*) – fil extensions of the data available on a free download server
- **raise_err** (*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()

>>> valid_file_format = bbd.validate_file_format(osm_file_format='PBF')
>>> valid_file_format
'.pbf'

>>> valid_file_format = bbd.validate_file_format(osm_file_format='.osm.pbf')
```

(continues on next page)

(continued from previous page)

```
>>> valid_file_format
'.pbf'
```

BBBikeDownloader.validate_subregion_name

BBBikeDownloader.**validate_subregion_name**(*subregion_name*, *valid_subregion_names=None*, *raise_err=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_subregion_names** (*Iterable*) – names of all (sub)regions available on a free download server
- **raise_err** (*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()

>>> subrgn_name = 'birmingham'

>>> valid_name = bbd.validate_subregion_name(subregion_name=subrgn_name)
>>> valid_name
'Birmingham'
```

2.1.2 reader

Read the OSM data extracts in various file formats.

Transform OSM data

Transformer()	Transform / reformat data.

Transformer

class pydriosm.reader.**Transformer**

Transform / reformat data.

Examples:

```
>>> from pydriosm.reader import Transformer

>>> geometry = {'type': 'Point', 'coordinates': [-0.5134241, 52.6555853]}
>>> geometry_ = Transformer.transform_unitary_geometry(geometry)
>>> type(geometry_)
shapely.geometry.point.Point
>>> geometry_.wkt
'POINT (-0.5134241 52.6555853)'
```

Methods

point_as_polygon(multi_poly_coords)	Make the coordinates of a single 'Point' (in a 'MultiPolygon') be reformatted to a 'Polygon'-like coordinates.
transform_geometry(layer_data, layer_name)	Reformat the field of `'geometry'` into shapely.geometry object.
transform_geometry_collection(geometry[, ...])	Transform a collection of geometry from dict into a shapely.geometry object.
transform_other_tags(other_tags)	Reformat a `'other_tags'` from string into dictionary type.
transform_unitary_geometry(geometry[, mode, ...])	Transform a unitary geometry from dict into a shapely.geometry object.
update_other_tags(prop_or_feat[, mode])	Update the original data of `'other_tags'` with parsed data.

Transformer.point_as_polygon

classmethod Transformer.**point_as_polygon**(*multi_poly_coords*)

Make the coordinates of a single 'Point' (in a 'MultiPolygon') be reformatted to a 'Polygon'-like coordinates.

The list of coordinates of some 'MultiPolygon' features may contain single points. In order to reformat such multipart geometry (from dict into shapely.geometry type), there is a need to ensure each of the constituent parts is a shapely.geometry.Polygon.

Parameters
multi_poly_coords (*list*) – original data of coordinates of a shapely.geometry.MultiPolygon feature

Returns
coordinates that are reformatted as appropriate

Return type
list

Examples:

```
>>> from pydriosm.reader import Transformer

>>> geometry = {
...     'type': 'MultiPolygon',
...     'coordinates': [[[[-0.6920145, 52.6753268], [-0.6920145, 52.6753268]]]]
... }
>>> mp_coords = geometry['coordinates']

>>> mp_coords_ = Transformer.point_as_polygon(mp_coords)
>>> mp_coords_
[[[[-0.6920145, 52.6753268],
   [-0.6920145, 52.6753268],
   [-0.6920145, 52.6753268]]]]
```

Transformer.transform_geometry

classmethod Transformer.**transform_geometry**(*layer_data*, *layer_name*)

Reformat the field of 'geometry' into shapely.geometry object.

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

Returns
(OSM feature with) reformatted geometry field

Return type
pandas.DataFrame | pandas.Series

Examples:

```
>>> from pydriosm.reader import Transformer

>>> # An example of points layer data
>>> lyr_name = 'points'
>>> dat_ = {
...     'type': 'Feature',
...     'geometry': {
...         'type': 'Point',
...         'coordinates': [-0.5134241, 52.6555853]
...     },
...     'properties': {
```

(continues on next page)

(continued from previous page)

```
...         'osm_id': '488432',
...         'name': None,
...         'barrier': None,
...         'highway': None,
...         'ref': None,
...         'address': None,
...         'is_in': None,
...         'place': None,
...         'man_made': None,
...         'other_tags': '"odbl"=>"clean"'
...     },
...     'id': 488432
... }
>>> lyr_data = pd.DataFrame.from_dict(dat_, orient='index').T

>>> geom_dat = Transformer.transform_geometry(layer_data=lyr_data, layer_name=lyr_name)
>>> geom_dat
0    POINT (-0.5134241 52.6555853)
Name: geometry, dtype: object
```

See also:

- Examples for the method *PBFReadParse.read_pbf()*.

Transformer.transform_geometry_collection

classmethod Transformer.**transform_geometry_collection**(*geometry*, *mode=1*, *to_wkt=False*)

Transform a collection of geometry from dict into a shapely.geometry object.

Parameters

- **geometry** (*list | dict*) – geometry data for a feature of GeometryCollection
- **mode** (*int*) – indicate the way of parsing the input;
  - when mode=1 **(default)**, the input geometry should be directly accessible and would be in the format of {'type': <shape type>, 'coordinates': <coordinates>} or as a row of a pandas.DataFrame;
  - when mode=2, the input geometry is in the GeoJSON format
- **to_wkt** (*bool*) – whether to represent the geometry in the WKT (well-known text) format, defaults to False

Returns

reformatted geometry data

Return type

shapely.geometry.base.HeterogeneousGeometrySequence | dict | str

Examples:

```
>>> from pydriosm.reader import PBFReadParse
>>> from shapely.geometry import GeometryCollection

>>> g1_dat_ = {
...     'type': 'GeometryCollection',
...     'geometries': [
...         {'type': 'Point', 'coordinates': [-0.5096176, 52.6605168]},
...         {'type': 'Point', 'coordinates': [-0.5097337, 52.6605812]}
...     ]
... }
>>> g1_dat = g1_dat_['geometries']
>>> g1_data = PBFReadParse.transform_geometry_collection(g1_dat)
>>> type(g1_data)
shapely.geometry.collection.GeometryCollection
>>> g1_data.wkt
'GEOMETRYCOLLECTION (POINT (-0.5096176 52.6605168), POINT (-0.5097337 52.6605812))'

>>> g2_dat = {
...     'type': 'Feature',
...     'geometry': {
...         'type': 'GeometryCollection',
...         'geometries': [
...             {'type': 'Point', 'coordinates': [-0.5096176, 52.6605168]},
...             {'type': 'Point', 'coordinates': [-0.5097337, 52.6605812]}]
...      },
...     'properties': {
...         'osm_id': '256254',
...         'name': 'Fife Close',
...         'type': 'site',
...         'other_tags': '"naptan:StopAreaCode"=>"270G02701525"'
...     },
...     'id': 256254
... }
>>> g2_data = PBFReadParse.transform_geometry_collection(g2_dat, mode=2)
>>> type(g2_data)
dict
>>> list(g2_data.keys())
['type', 'geometry', 'properties', 'id']
>>> g2_data['geometry']
'GEOMETRYCOLLECTION (POINT (-0.5096176 52.6605168), POINT (-0.5097337 52.6605812))'
```

Transformer.transform_other_tags

classmethod Transformer.**transform_other_tags**(*other_tags*)

Reformat a `'other_tags'` from string into dictionary type.

Parameters

other_tags (*str | None*) – data of `'other_tags'` of a single feature in a PBF data file

Returns

reformatted data of `'other_tags'`

Return type

dict | None

Examples:

```
>>> from pydriosm.reader import Transformer

>>> other_tags_dat = Transformer.transform_other_tags(other_tags='"odbl"=>"clean"')
>>> other_tags_dat
{'odbl': 'clean'}
```

See also:

- Examples for the method *PBFReadParse.read_pbf()*.

Transformer.transform_unitary_geometry

classmethod Transformer.**transform_unitary_geometry**(*geometry*, *mode=1*, *to_wkt=False*)

Transform a unitary geometry from dict into a shapely.geometry object.

Parameters

- **geometry** (*dict | pandas.DataFrame*) – geometry data for a feature of one of the geometry types including `'Point'`, `'LineString'`, `'MultiLineString'` and `'MultiPolygon'`
- **mode** (*int*) – indicate the way of parsing the input;
  - when `mode=1` **(default)**, the input `geometry` should be directly accessible and would be in the format of `{'type': <shape type>, 'coordinates': <coordinates>}` or as a row of a pandas.DataFrame;
  - when `mode=2`, the input `geometry` is in the GeoJSON format
- **to_wkt** (*bool*) – whether to represent the geometry in the WKT (well-known text) format, defaults to `False`

Returns

reformatted geometry data

Return type

shapely.geometry.Point | dict | str

Examples:

```
>>> from pydriosm.reader import PBFReadParse

>>> g1_dat = {'type': 'Point', 'coordinates': [-0.5134241, 52.6555853]}
>>> g1_data = PBFReadParse.transform_unitary_geometry(g1_dat)
>>> type(g1_data)
shapely.geometry.point.Point
>>> g1_data.wkt
'POINT (-0.5134241 52.6555853)'

>>> g2_dat = {
...     'type': 'Feature',
...     'geometry': {
...         'type': 'Point',
...         'coordinates': [-0.5134241, 52.6555853]
...     },
...     'properties': {
...         'osm_id': '488432',
```

(continues on next page)

(continued from previous page)

```
...         'name': None,
...         'barrier': None,
...         'highway': None,
...         'ref': None,
...         'address': None,
...         'is_in': None,
...         'place': None,
...         'man_made': None,
...         'other_tags': '"odbl"=>"clean"'
...     },
...     'id': 488432
... }
>>> g2_data = PBFReadParse.transform_unitary_geometry(g2_dat, mode=2)
>>> type(g2_data)
dict
>>> list(g2_data.keys())
['type', 'geometry', 'properties', 'id']
>>> g2_data['geometry']
'POINT (-0.5134241 52.6555853)'
```

Transformer.update_other_tags

classmethod Transformer.**update_other_tags**(*prop_or_feat*, *mode=1*)

Update the original data of `'other_tags'` with parsed data.

Parameters

- **prop_or_feat** (*dict*) – original data of a feature or a `'properties'` field
- **mode** (*int*) – options include {1, 2} indicating what action to take; when `mode=1` (default), `prop_or_feat` should be data of a feature; when `mode=2`, `prop_or_feat` should be data of a `'properties'` field

Returns

updated data of a feature or a 'properties' field

Return type

dict

Examples:

```
>>> from pydriosm.reader import Transformer

>>> prop_dat = {
...     'properties': {
...         'osm_id': '488432',
...         'name': None,
...         'barrier': None,
...         'highway': None,
...         'ref': None,
...         'address': None,
...         'is_in': None,
...         'place': None,
...         'man_made': None,
...         'other_tags': '"odbl"=>"clean"'
...     },
```

(continues on next page)

(continued from previous page)

```
... }

>>> prop_dat_ = Transformer.update_other_tags(prop_dat['properties'])
>>> prop_dat_
{'osm_id': '488432',
 'name': None,
 'barrier': None,
 'highway': None,
 'ref': None,
 'address': None,
 'is_in': None,
 'place': None,
 'man_made': None,
 'other_tags': {'odbl': 'clean'}}
```

See also:

- Examples for the method *PBFReadParse.read_pbf()*.

Parse OSM data

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

SHPReadParse

class pydriosm.reader.**SHPReadParse**

Read/parse Shapefile data.

Examples:

```
>>> from pydriosm.reader import SHPReadParse

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

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

Attributes

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

SHPReadParse.ENCODING

SHPReadParse.**ENCODING = 'UTF-8'**

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

SHPReadParse.EPSG4326_WGS84_ESRI_WKT

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

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

SHPReadParse.EPSG4326_WGS84_PROJ4

SHPReadParse.**EPSG4326_WGS84_PROJ4 = '+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs'**

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

SHPReadParse.EPSG4326_WGS84_PROJ4_

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

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

SHPReadParse.LAYER_NAMES

SHPReadParse.**LAYER_NAMES = {'buildings', 'landuse', 'natural', 'places', 'pofw', 'points', 'pois', 'railways', 'roads', 'traffic', 'transport', 'water', 'waterways'}**

set: Valid layer names for an OSM shapefile.

SHPReadParse.SHAPE_TYPE_GEOM

SHPReadParse.**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'>}**

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

SHPReadParse.SHAPE_TYPE_GEOM_NAME

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

dict: Shape type codes of shapefiles and their corresponding geometry object names

SHPReadParse.SHAPE_TYPE_NAME_LOOKUP

SHPReadParse.**SHAPE_TYPE_NAME_LOOKUP** = {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'}

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

SHPReadParse.VECTOR_DRIVER

SHPReadParse.**VECTOR_DRIVER** = 'ESRI Shapefile'

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

Methods

find_shp_layer_name(shp_filename)	Find the layer name of OSM shapefile given its filename.
merge_layer_shps(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_shp_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.

SHPReadParse.find_shp_layer_name

classmethod SHPReadParse.**find_shp_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 import SHPReadParse

>>> SHPReadParse.find_shp_layer_name("") is None
True

>>> SHPReadParse.find_shp_layer_name("gis_osm_railways_free_1.shp")
'railways'

>>> SHPReadParse.find_shp_layer_name("gis_osm_transport_a_free_1.shp")
'transport'
```

SHPReadParse.merge_layer_shps

classmethod SHPReadParse.**merge_layer_shps**(*shp_zip_pathnames*, *layer_name*, *engine='pyshp'*, *rm_zip_extracts=True*, *output_dir=None*, *rm_shp_temp=True*, *ret_shp_pathname=False*, *verbose=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'` (default) and `'geopandas'` (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`

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

>>> # Download the .shp.zip file of Manchester and West Yorkshire
>>> subrgn_names = ['Greater Manchester', 'West Yorkshire']
>>> file_fmt = ".shp"
>>> data_dir = "tests\osm_data"

>>> gfd = GeofabrikDownloader()

>>> gfd.download_osm_data(subrgn_names, file_fmt, data_dir, verbose=True)
To download .shp.zip data of the following geographic (sub)region(s):
    Greater Manchester
    West Yorkshire
? [No]|Yes: yes
Downloading "greater-manchester-latest-free.shp.zip"
    to "tests\osm_data\greater-manchester\" ... Done.
Downloading "west-yorkshire-latest-free.shp.zip"
    to "tests\osm_data\west-yorkshire\" ... Done.

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

>>> # Merge the layers of 'railways' of the two subregions
>>> merged_shp_path = SHPReadParse.merge_layer_shps(
...     gfd.data_paths, layer_name='railways', verbose=True, ret_shp_pathname=True)
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\".

>>> # 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\gre_man-wes_yor-railways\linestring.shp'

>>> # Read the merged .shp file
>>> merged_shp_data = SHPReadParse.read_shp(merged_shp_path[0], emulate_gpd=True)
>>> merged_shp_data.head()
    osm_id  code  ... tunnel                                           geometry
0   928999  6101  ...      F  LINESTRING (-2.2844621 53.4802635, -2.2851997 ...
1   929904  6101  ...      F  LINESTRING (-2.2917977 53.4619559, -2.2924877 ...
2   929905  6102  ...      F  LINESTRING (-2.2794048 53.4605819, -2.2799722 ...
3  3663332  6102  ...      F  LINESTRING (-2.2382139 53.4817985, -2.2381708 ...
```

(continues on next page)

(continued from previous page)

```
4  3996086  6101  ...      F  LINESTRING (-2.6003053 53.4604346, -2.6005261 ...
[5 rows x 8 columns]

>>> # Delete the test data 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.
```

See also:

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

SHPReadParse.merge_shps

classmethod SHPReadParse.**merge_shps**(*shp_pathnames*, *path_to_merged_dir*, *engine='pyshp'*, ***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 the function *merge_layer_shps()*.
- Resource: https://github.com/GeospatialPython/pyshp

SHPReadParse.read_layer_shps

classmethod SHPReadParse.**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 *SHPReadParse.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 import SHPReadParse
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pyhelpers.dirs import cd, delete_dir
>>> import os

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

>>> gfd = GeofabrikDownloader()

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

>>> london_shp_zip = gfd.data_paths[0]
>>> os.path.relpath(london_shp_zip)
'tests\osm_data\greater-london\greater-london-latest-free.shp.zip'

>>> # Extract the downloaded .shp.zip file
>>> london_shp_dir = SHPReadParse.unzip_shp_zip(
...     london_shp_zip, layer_names='railways', ret_extract_dir=True)
>>> os.listdir(london_shp_dir)
```

(continues on next page)

(continued from previous page)

```
['gis_osm_railways_free_1.cpg',
 'gis_osm_railways_free_1.dbf',
 'gis_osm_railways_free_1.prj',
 'gis_osm_railways_free_1.shp',
 'gis_osm_railways_free_1.shx']
>>> london_railways_shp_path = cd(london_shp_dir, "gis_osm_railways_free_1.shp")

>>> # Read the 'railways' layer
>>> london_railways_shp = SHPReadParse.read_layer_shps(london_railways_shp_path)
>>> london_railways_shp.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]

>>> # Extract only the features labelled 'rail' and save the extracted data to file
>>> railways_rail_shp, railways_rail_shp_path = SHPReadParse.read_layer_shps(
...     london_railways_shp_path, feature_names='rail', save_feat_shp=True,
...     ret_feat_shp_path=True)
>>> railways_rail_shp['fclass'].unique()
array(['rail'], dtype=object)

>>> type(railways_rail_shp_path)
list
>>> len(railways_rail_shp_path)
1
>>> os.path.basename(railways_rail_shp_path[0])
'gis_osm_railways_free_1_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.
```

SHPReadParse.read_shp

classmethod SHPReadParse.**read_shp**(*shp_pathname*, *engine='pyshp'*, *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'` (default) and `'geopandas'` (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 import SHPReadParse
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pyhelpers.dirs import cd, delete_dir
>>> import os
>>> import glob

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

>>> gfd = GeofabrikDownloader()

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

>>> london_shp_zip = gfd.data_paths[0]
>>> os.path.relpath(london_shp_zip)
'tests\osm_data\greater-london\greater-london-latest-free.shp.zip'

>>> # Extract all
>>> london_shp_dir = SHPReadParse.unzip_shp_zip(london_shp_zip, ret_extract_dir=True)

>>> # Get the pathname of the .shp data of 'railways'
>>> path_to_railways_shp = glob.glob(cd(london_shp_dir, "*railways*.shp"))[0]
>>> os.path.relpath(path_to_railways_shp)  # Check the pathname of the .shp file
'tests\osm_data\greater-london\greater-london-latest-free-shp\gis_osm_railwa...

>>> # Read the data of 'railways'
>>> london_railways = SHPReadParse.read_shp(path_to_railways_shp)
>>> london_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]
```

(continues on next page)

(continued from previous page)

```
>>> # Set `emulate_gpd=True` to return data of similar format to what GeoPandas does
>>> london_railways = SHPReadParse.read_shp(path_to_railways_shp, emulate_gpd=True)
>>> london_railways.head()
   osm_id  code  ... tunnel                                           geometry
0   30804  6101  ...      F  LINESTRING (0.0048644 51.6279262, 0.0061979 51...
1  101298  6103  ...      F  LINESTRING (-0.2249906 51.493682, -0.2251678 5...
2  101486  6103  ...      F  LINESTRING (-0.2055497 51.5195429, -0.2051377 ...
3  101511  6101  ...      F  LINESTRING (-0.2119027 51.5241906, -0.2108059 ...
4  282898  6103  ...      F  LINESTRING (-0.1862586 51.6159083, -0.1868721 ...
[5 rows x 8 columns]

>>> # Alternatively, set `engine` to be 'geopandas' (or 'gpd') to use GeoPandas
>>> london_railways_ = SHPReadParse.read_shp(path_to_railways_shp, engine='geopandas')
>>> london_railways_.head()
   osm_id  code  ... tunnel                                           geometry
0   30804  6101  ...      F    LINESTRING (0.00486 51.62793, 0.00620 51.62927)
1  101298  6103  ...      F  LINESTRING (-0.22499 51.49368, -0.22517 51.494...
2  101486  6103  ...      F  LINESTRING (-0.20555 51.51954, -0.20514 51.519...
3  101511  6101  ...      F  LINESTRING (-0.21190 51.52419, -0.21081 51.523...
4  282898  6103  ...      F  LINESTRING (-0.18626 51.61591, -0.18687 51.61384)
[5 rows x 8 columns]

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

>>> # Delete the download/data 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.
```

SHPReadParse.unzip_shp_zip

classmethod SHPReadParse.**unzip_shp_zip**(*shp_zip_pathname*, *extract_to=None*, *layer_names=None*, *separate=False*, *ret_extract_dir=False*, *verbose=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`

Returns

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

Return type

str

Examples:

```
>>> from pydriosm.reader import SHPReadParse
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pyhelpers.dirs import cd, delete_dir
>>> import os

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

>>> gfd = GeofabrikDownloader()

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

>>> path_to_shp_zip = gfd.data_paths[0]
>>> os.path.relpath(path_to_shp_zip)
'tests\osm_data\greater-london\greater-london-latest-free.shp.zip'

>>> # To extract data of a specific layer 'railways'
>>> london_railways_dir = SHPReadParse.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(london_railways_dir)  # Check the directory
'tests\osm_data\greater-london\greater-london-latest-free-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', 'transport', 'traffic']
>>> dirs_of_layers = SHPReadParse.unzip_shp_zip(
...     path_to_shp_zip, layer_names=lyr_names, separate=True, verbose=2,
...     ret_extract_dir=True)
Extracting the following layer(s):
```

(continues on next page)

(continued from previous page)

```
    'railways'
    'transport'
    'traffic'
    from "tests\osm_data\greater-london\greater-london-latest-free.shp.zip"
      to "tests\osm_data\greater-london\greater-london-latest-free-shp\" ... Done.
Grouping files by layers ...
    railways ... Done.
    transport_a ... Done.
    transport ... Done.
    traffic_a ... Done.
    traffic ... Done.
Done.

>>> len(dirs_of_layers) == 3
True
>>> os.path.relpath(os.path.commonpath(dirs_of_layers))
'tests\osm_data\greater-london\greater-london-latest-free-shp'
>>> set(map(os.path.basename, dirs_of_layers))
{'railways', 'traffic', 'transport'}

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

>>> # To extract all (without specifying `layer_names`
>>> london_shp_dir = SHPReadParse.unzip_shp_zip(
...     path_to_shp_zip, verbose=True, ret_extract_dir=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.

>>> # Check the directory
>>> os.path.relpath(london_shp_dir)
'tests\osm_data\greater-london\greater-london-latest-free-shp'
>>> len(os.listdir(london_shp_dir))
91
>>> # Get the names of all available layers
>>> set(filter(None, map(SHPReadParse.find_shp_layer_name, os.listdir(london_shp_dir))))
{'buildings',
 'landuse',
 'natural',
 'places',
 'pofw',
 'pois',
 'railways',
 'roads',
 'traffic',
 'transport',
 'water',
 'waterways'}

>>> # Delete the download/data 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.
```

SHPReadParse.validate_shp_layer_names

classmethod SHPReadParse.**validate_shp_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 import SHPReadParse

>>> SHPReadParse.validate_shp_layer_names(None)
[]

>>> SHPReadParse.validate_shp_layer_names('point')
['points']

>>> SHPReadParse.validate_shp_layer_names(['point', 'land'])
['points', 'landuse']

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

SHPReadParse.write_to_shapefile

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

Examples:

```
>>> from pydriosm.reader import SHPReadParse
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pyhelpers.dirs import cd, delete_dir
>>> import os
>>> import glob

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

>>> gfd = GeofabrikDownloader()

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

>>> london_shp_zip = gfd.data_paths[0]
>>> os.path.relpath(london_shp_zip)
'tests\osm_data\greater-london\greater-london-latest-free.shp.zip'

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

>>> railways_shp_dir = SHPReadParse.unzip_shp_zip(
...     london_shp_zip, layer_names=lyr_name, 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.
>>> # Check out the output directory
>>> os.path.relpath(railways_shp_dir)
'tests\osm_data\greater-london\greater-london-latest-free-shp'

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

>>> # Read the .shp file
>>> london_railways_shp = SHPReadParse.read_shp(path_to_railways_shp)
```

(continues on next page)

(continued from previous page)

```
>>> # 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\greater-london\railways'

>>> # Save the data of 'railways' to the new directory
>>> path_to_railways_shp_ = SHPReadParse.write_to_shapefile(
...     london_railways_shp, railways_subdir, ret_shp_pathname=True, verbose=True)
Writing data to "tests\osm_data\greater-london\railways\railways.*" ... Done.
>>> os.path.basename(path_to_railways_shp_)
'railways.shp'

>>> # If `shp_filename` is specified
>>> path_to_railways_shp_ = SHPReadParse.write_to_shapefile(
...     london_railways_shp, railways_subdir, shp_filename="rail_data",
...     ret_shp_pathname=True, verbose=True)
Writing data to "tests\osm_data\greater-london\railways\rail_data.*" ... Done.
>>> os.path.basename(path_to_railways_shp_)
'rail_data.shp'

>>> # Retrieve the saved the .shp file
>>> london_railways_shp_ = SHPReadParse.read_shp(path_to_railways_shp_)

>>> # Check if the retrieved .shp data is equal to the original one
>>> london_railways_shp_.equals(london_railways_shp)
True

>>> # Delete the download/data 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.
```

PBFReadParse

class pydriosm.reader.**PBFReadParse**

Read/parse PBF data.

Examples:

```
>>> from pydriosm.reader import PBFReadParse

>>> PBFReadParse.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

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

PBFReadParse.LAYER_GEOM

PBFReadParse.**LAYER_GEOM = {'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'>}**

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

Methods

get_pbf_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_pbf_layer_names(pbf_pathname[, verbose])	Get names (and indices) of all available layers in a PBF data file.
read_pbf(pbf_pathname[, readable, expand, ...])	Parse a PBF data file (by GDAL).
read_pbf_layer(layer[, readable, expand, ...])	Parse a layer of a PBF data file.
transform_pbf_layer_field(layer_data, layer_name)	Parse data of a layer of PBF data.

PBFReadParse.get_pbf_layer_geom_types

classmethod PBFReadParse.**get_pbf_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 import PBFReadParse

>>> PBFReadParse.get_pbf_layer_geom_types()
{'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}

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

PBFReadParse.get_pbf_layer_names

classmethod PBFReadParse.**get_pbf_layer_names**(*pbf_pathname*, *verbose=False*)

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

Parameters

- **pbf_pathname** (*str | os.PathLike[str]*) – path to a PBF data file
- **verbose** (*bool | int*) – whether to print relevant information in console, defaults to False

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

Return type
dict

Examples:

```
>>> from pydriosm.reader import PBFReadParse
>>> 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_osm_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"
```

(continues on next page)

(continued from previous page)

```
    to "tests\osm_data\greater-london\" ... Done.

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

PBFReadParse.read_pbf

classmethod PBFReadParse.**read_pbf**(*pbf_pathname*, *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

- **pbf_pathname** (*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 import PBFReadParse
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pyhelpers.dirs import delete_dir
>>> import os

>>> # Download the PBF data file of 'Rutland' as an example
>>> subrgn_name = 'rutland'
>>> file_format = ".pbf"
>>> dwnld_dir = "tests\osm_data"

>>> gfd = GeofabrikDownloader()

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

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

>>> # Read the downloaded PBF data
```

(continues on next page)

(continued from previous page)

```
>>> rutland_pbf = PBFReadParse.read_pbf(rutland_pbf_path)
>>> 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...
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 = PBFReadParse.read_pbf(rutland_pbf_path, 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 = PBFReadParse.read_pbf(rutland_pbf_path, 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 (-0.7266581 52.6695058)
Name: geometry, dtype: object

>>> # Set both `expand` and `parse_properties` to be `True`
>>> pbf_2 = PBFReadParse.read_pbf(rutland_pbf_path, 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"
```

(continues on next page)

(continued from previous page)

```
4        "direction"=>"clockwise"
Name: other_tags, dtype: object

>>> # Set both `expand` and `parse_other_tags` to be `True`
>>> pbf_3 = PBFReadParse.read_pbf(rutland_pbf_path, 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``
>>> 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(gfd.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_osm_pbf()* and *BBBikeReader.read_osm_pbf()*.

PBFReadParse.read_pbf_layer

classmethod PBFReadParse.**read_pbf_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 the method *PBFReadParse.read_pbf()*.

PBFReadParse.transform_pbf_layer_field

classmethod PBFReadParse.**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 the method *PBFReadParse.read_pbf()*.

VarReadParse

class pydriosm.reader.**VarReadParse**

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

Attributes

FILE_FORMATS	set: Valid file formats.

VarReadParse.FILE_FORMATS

VarReadParse.**FILE_FORMATS = {'.csv.xz', 'geojson.xz'}**

set: Valid file formats.

Methods

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

VarReadParse.read_csv_xz

classmethod VarReadParse.**read_csv_xz**(*csv_xz_pathname*, *col_names=None*)

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

Parameters

- **csv_xz_pathname** (*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()*.

VarReadParse.read_geojson_xz

classmethod VarReadParse.**read_geojson_xz**(*geojson_xz_pathname*, *engine=None*, *parse_geometry=False*)

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

Parameters

- **geojson_xz_pathname** (*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()*.

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	str: Default download directory.
FILE_FORMATS	set: Valid file formats.

GeofabrikReader.DEFAULT_DATA_DIR

GeofabrikReader.**DEFAULT_DATA_DIR = 'osm_data\\geofabrik'**

str: Default download directory.

GeofabrikReader.FILE_FORMATS

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

set: Valid file formats.

Methods

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).
merge_subregion_layer_shp(subregion_names, ...)	Merge shapefiles for a specific layer of two or multiple geographic regions.
read_osm_pbf(subregion_name[, data_dir, ...])	Read a PBF (.osm.pbf) data file of a geographic (sub)region.
read_shp_zip(subregion_name[, layer_names, ...])	Read a .shp.zip data file of a geographic (sub)region.

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_osm_data(subrgn_name, file_format, dat_dir, verbose=True)
To download .osm.pbf data of the following geographic (sub)region(s):
    Rutland
? [No]|Yes: yes
Downloading "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 .shp.zip file of Rutland as an example
>>> subrgn_name = 'london'
>>> file_format = ".pbf"
>>> dat_dir = "tests\osm_data"

>>> gfr.downloader.download_osm_data(subrgn_name, file_format, dat_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_path = gfr.data_paths[0]
>>> os.path.relpath(london_pbf_path)
'tests\osm_data\greater-london\greater-london-latest.osm.pbf'

>>> lyr_idx_names = gfr.get_pbf_layer_names(london_pbf_path)
>>> 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(dat_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_osm_data(
...     subrgn_name, file_format, dat_dir, verbose=True, ret_download_path=True)
To download .shp.zip data of the following geographic (sub)region(s):
    Greater London
? [No]|Yes: yes
Downloading "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

>>> # 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_fr...

>>> # Get/save shapefile data of features labelled 'rail' only
```

(continues on next page)

(continued from previous page)

```
>>> 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\railways\rail.shp'

>>> # 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.merge_subregion_layer_shp

GeofabrikReader.**merge_subregion_layer_shp**(*subregion_names*, *layer_name*, *data_dir=None*, *engine='pyshp'*, *update=False*, *download=True*, *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_subregion_layer_shp(
...     subrgn_name, lyr_name, dat_dir, verbose=True, ret_merged_shp_path=True)
To download .shp.zip data of the following geographic (sub)region(s):
    Greater Manchester
    West Yorkshire
? [No]|Yes: yes
Downloading "greater-manchester-latest-free.shp.zip"
    to "tests\osm_data\greater-manchester\" ... Done.
Downloading "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"
```

(continues on next page)

(continued from previous page)

```
        In progress ... Done.
        Find the merged shapefile at "tests\osm_data\gre_man-wes_yor-railways\".

>>> 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
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.
Deleting "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_subregion_layer_shp(
...     subrgn_name, lyr_name, dat_dir, verbose=True, ret_merged_shp_path=True)
To download .shp.zip data of the following geographic (sub)region(s):
    Greater London
    Kent
    Surrey
? [No]|Yes: yes
Downloading "greater-london-latest-free.shp.zip"
    to "tests\osm_data\greater-london\" ... Done.
Downloading "kent-latest-free.shp.zip"
    to "tests\osm_data\kent\" ... Done.
Downloading "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"
```

(continues on next page)

(continued from previous page)

```
        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'
>>> 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_osm_pbf

GeofabrikReader.**read_osm_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 the method *PBFReadParse.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()

>>> subrgn_name = 'rutland'
>>> dat_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_osm_pbf(subrgn_name, data_dir=dat_dir, verbose=True)
Downloading "rutland-latest.osm.pbf"
    to "tests\osm_data\rutland\" ... Done.
Reading "tests\osm_data\rutland\rutland-latest.osm.pbf" ... Done.
>>> type(pbf_raw)
dict
```

(continues on next page)

(continued from previous page)

```
>>> 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_osm_pbf(subrgn_name, dat_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_osm_pbf(subrgn_name, dat_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   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]

>>> # Set `readable` and `parse_geometry` to be `True`
>>> pbf_parsed_1 = gfr.read_osm_pbf(subrgn_name, dat_dir, readable=True,
...                                 parse_geometry=True)
>>> pbf_parsed_1_point = pbf_parsed_1['points'][0]
>>> pbf_parsed_1_point['geometry']
'POINT (-0.5134241 52.6555853)'
>>> pbf_parsed_1_point['properties']['other_tags']
'"odbl"=>"clean"'

>>> # Set `readable` and `parse_other_tags` to be `True`
>>> pbf_parsed_2 = gfr.read_osm_pbf(subrgn_name, dat_dir, readable=True,
...                                 parse_other_tags=True)
>>> pbf_parsed_2_point = pbf_parsed_2['points'][0]
>>> pbf_parsed_2_point['geometry']
{'type': 'Point', 'coordinates': [-0.5134241, 52.6555853]}
>>> pbf_parsed_2_point['properties']['other_tags']
{'odbl': 'clean'}

>>> # Set `readable`, `parse_geometry` and `parse_other_tags` to be `True`
>>> pbf_parsed_3 = gfr.read_osm_pbf(subrgn_name, dat_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 (-0.5134241 52.6555853)'
>>> pbf_parsed_3_point['properties']['other_tags']
```

(continues on next page)

(continued from previous page)

```
{'odbl': 'clean'}

>>> # 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_shp_zip

GeofabrikReader.**read_shp_zip**(*subregion_name*, *layer_names=None*, *feature_names=None*, *data_dir=None*, *update=False*, *download=True*, *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 | collections.OrderedDict | 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_zip(
...     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_zip(
...     subregion_name=subrgn_name, data_dir=dat_dir, download=True, verbose=True)
Downloading "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-free-shp\" ... Done.
>>> type(london_shp_data)
collections.OrderedDict
>>> list(london_shp_data.keys())
['buildings',
 'landuse',
 'natural',
 'places',
 'pofw',
 'pois',
 'railways',
 'roads',
 'traffic',
 'transport',
 '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_zip(
...     subregion_name=subrgn_name, layer_names=subrgn_layer, data_dir=dat_dir,
```

(continues on next page)

(continued from previous page)

```
...     rm_extracts=True, verbose=True)
Reading the shapefile(s) at
    "tests\osm_data\greater-london\greater-london-latest-free-shp\" ... Done.
Deleting the extracts "tests\osm_data\greater-london\greater-london-latest-free-sh...
>>> type(london_shp_transport)
collections.OrderedDict
>>> 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_zip(
...     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\" ... Done.
Reading the shapefile(s) at
    "tests\osm_data\greater-london\greater-london-latest-free-shp\" ... Done.
Deleting the extracts "tests\osm_data\greater-london\greater-london-latest-free-sh...
>>> type(london_bus_stop)
collections.OrderedDict
>>> list(london_bus_stop.keys())
['transport']

>>> fclass = london_bus_stop['transport'].fclass.unique()
>>> fclass
array(['bus_stop'], dtype=object)

>>> # 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_zip(
...     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\" ... Done.
Reading the shapefile(s) at
    "tests\osm_data\greater-london\greater-london-latest-free-shp\" ... Done.
Deleting the extracts "tests\osm_data\greater-london\greater-london-latest-free-sh...
Deleting "tests\osm_data\greater-london\greater-london-latest-free.shp.zip" ... Done.
>>> type(london_shp_tra_roa_par_tru)
collections.OrderedDict
>>> list(london_shp_tra_roa_par_tru.keys())
```

(continues on next page)

(continued from previous page)

```
['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.
```

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

DEFAULT_DOWNLOAD_DIR	str: Default download directory.
FILE_FORMATS	set: Valid file formats.

BBBikeReader.DEFAULT_DOWNLOAD_DIR

BBBikeReader.**DEFAULT_DOWNLOAD_DIR = 'osm_data\\bbbike'**

str: Default download directory.

BBBikeReader.FILE_FORMATS

BBBikeReader.**FILE_FORMATS = {'.csv.xz', '.garmin-onroad-latin1.zip', '.garmin-onroad.zip', '.garmin-opentopo.zip', '.garmin-osm.zip', '.geojson.xz', '.gz', '.mapsforge-osm.zip', '.pbf', '.shp.zip', '.svg-osm.zip'}**

set: Valid file formats.

Methods

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_osm_pbf(subregion_name[, data_dir, ...])	Read a PBF (.osm.pbf) data file of a geographic (sub)region.
read_shp_zip(subregion_name[, layer_names, ...])	Read a shapefile of a geographic (sub)region.

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()

>>> subrgn_name = 'Leeds'
>>> dat_dir = "tests\osm_data"

>>> leeds_csv_xz = bbr.read_csv_xz(subrgn_name, dat_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(subrgn_name, dat_dir, verbose=True, download=True)
Downloading "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  154921    None  None
3  node  154922    None  None
4  node  154923    None  None

>>> # Delete the downloaded .csv.xz data file
>>> 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_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"
    to "tests\osm_data\leeds\" ... Done.
Parsing the data ... Done.
>>> leeds_geoj.head()
                                            geometry                           properties
0  {'type': 'Point', 'coordinates': [-1.5558097, ...  {'highway': 'motorway_junction'...
1  {'type': 'Point', 'coordinates': [-1.34293, 53...  {'highway': 'motorway_junction'...
2  {'type': 'Point', 'coordinates': [-1.517335, 5...  {'highway': 'motorway_junction'...
3  {'type': 'Point', 'coordinates': [-1.514124, 5...  {'highway': 'motorway_junction'...
4  {'type': 'Point', 'coordinates': [-1.516511, 5...  {'highway': 'motorway_junction'...

>>> # 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 (-1.514124 53.7416937)
4     POINT (-1.516511 53.7256632)
Name: geometry, dtype: object

>>> # Delete the download directory
>>> delete_dir(dat_dir, verbose=True)
```

BBBikeReader.read_osm_pbf

BBBikeReader.**read_osm_pbf**(*subregion_name*, *data_dir=None*, *readable=False*, *expand=False*, *parse_geometry=False*, *parse_other_tags=False*, *parse_properties=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 the method _Reader.read_osm_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()

>>> subrgn_name = 'Leeds'
>>> dat_dir = "tests\osm_data"

>>> leeds_pbf_raw = bbr.read_osm_pbf(subrgn_name, data_dir=dat_dir, verbose=True)
Downloading "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_osm_pbf(
...     subrgn_name, data_dir=dat_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.head()
      id                                           geometry  ... tourism other_tags
0  10595  (POLYGON ((-1.5030223 53.6725382, -1.5034495 5...  ...    None       None
1  10600  (POLYGON ((-1.5116994 53.6764287, -1.5099361 5...  ...    None       None
2  10601  (POLYGON ((-1.5142403 53.6710831, -1.5143686 5...  ...    None       None
3  10612  (POLYGON ((-1.5129341 53.6704885, -1.5131883 5...  ...    None       None
4  10776  (POLYGON ((-1.5523801 53.7029081, -1.5524772 5...  ...    None       None
[5 rows x 26 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.
```

See also:

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

BBBikeReader.read_shp_zip

BBBikeReader.**read_shp_zip**(*subregion_name*, *layer_names=None*, *feature_names=None*, *data_dir=None*, *update=False*, *download=True*, *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_zip(
...     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_zip(
...     subregion_name=subrgn_name, data_dir=dat_dir, download=True, verbose=True)
Downloading "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\" ... Done.
>>> type(bham_shp)
collections.OrderedDict
>>> 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.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_zip(
...     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)
collections.OrderedDict
>>> list(bham_roads_shp.keys())
['roads']
>>> bham_roads_shp[lyr_name].head()
   osm_id  ... shape_type
0      37  ...          3
```

(continues on next page)

(continued from previous page)

```
1      38  ...          3
2      41  ...          3
3      45  ...          3
4      46  ...          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_zip(
...     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 data at "tests\osm_data\birmingham\Birmingham-shp\shape\" ... Done.
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)
collections.OrderedDict
>>> 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
8   canal                      Birmingham and Fazeley Canal
9   canal  Birmingham Old Line Canal Navigations - Rotton P
10  canal                               Oozells Street Loop
11  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
Deleting "tests\osm_data\" ... Done.
```

2.1.3 ios

Implement storage I/O of (parsed) OSM data extracts with PostgreSQL.

Manage I/O storage of parsed OSM data

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')
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
>>> type(osmdb.reader)
pydriosm.reader.GeofabrikReader

>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> type(osmdb.downloader)
pydriosm.downloader.BBBikeDownloader
>>> 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.
```

Attributes

DATA_SOURCES	list: Names of the data sources.
DATA_TYPES	dict: Specify a data-type dictionary for data or columns corresponding to Pandas.
downloader	Instance of either the class *GeofabrikDownloader* or *BBBikeDownloader*, depending on the specified `data_source` for creating an instance of the class *PostgresOSM*.
name	Name of the current property *downloader*.
reader	Instance of either *GeofabrikReader* or *BBBikeReader*, depending on the specified `data_source` for creating an instance of the calss *PostgresOSM*.
url	Homepage URL of data resource for current property *downloader*.

PostgresOSM.DATA_SOURCES

PostgresOSM.**DATA_SOURCES = ['Geofabrik', 'BBBike']**

list: Names of the data sources.

PostgresOSM.DATA_TYPES

PostgresOSM.**DATA_TYPES = {'bigint': <class 'numpy.int64'>, 'json': <class 'str'>, 'text': <class 'str'>}**

dict: Specify a data-type dictionary for data or columns corresponding to Pandas.

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 import PostgresOSM

>>> osmdb = PostgresOSM(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.name

property PostgresOSM.**name**

Name of the current property *downloader*.

Examples:

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

(continues on next page)

(continued from previous page)

```
>>> osmdb = PostgresOSM(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.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 import PostgresOSM

>>> osmdb = PostgresOSM(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> 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.
```

PostgresOSM.url

property PostgresOSM.**url**

Homepage URL of data resource for current property *downloader*.

Examples:

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

>>> osmdb = PostgresOSM(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
'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.
```

Methods

decode_pbf_layer(layer_dat[, decode_geojson])	Process raw data of a PBF layer retrieved from database.
drop_subregion_tables(subregion_names[, ...])	Delete all or specific schemas/layers of subregion data from the database being connected.
fetch_osm_data(subregion_name[, ...])	Fetch OSM data (of one or multiple layers) of a geographic (sub)region.
get_table_column_info(subregion_name, layer_name)	Get information about columns of a specific schema and table data of a geographic (sub)region.
get_table_name(subregion_name[, ...])	Get the default table name for a specific geographic (sub)region.
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_subregion_osm_pbf(subregion_names[, ...])	Import data of geographic (sub)region(s) that do not have (sub-)subregions into a database.
post_process_layer_dat(layer_dat[, ...])	Post-process the data of a specific layer.
subregion_table_exists(subregion_name, ...)	Check if a table (for a geographic (sub)region) exists.

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_osm_data()*.

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*)

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

Examples:

```
>>> from pydriosm.ios import PostgresOSM
>>> from pyhelpers.dirs import delete_dir

>>> osmdb = PostgresOSM(database_name='osmdb_test')
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_subregion_osm_pbf(
...     subrgn_name_1, data_dir=dat_dir, expand=True, parse_geometry=True,
```

(continues on next page)

(continued from previous page)

```
...     parse_properties=True, parse_other_tags=True, verbose=True)
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"
    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"
    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_zip(
...     subrgn_name_2, data_dir=dat_dir, rm_extracts=True, download=True, verbose=True)
Downloading "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)
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)
```

(continues on next page)

(continued from previous page)

```
None of the data exists.

>>> # Delete 'points' layer data of Rutland
>>> lyr_name = 'points'
>>> osmdb.drop_subregion_tables(subrgn_name, lyr_name, verbose=True)
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)
To drop table from postgres:***@localhost:5432/osmdb_test: "Rutland"
  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)
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)
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"
```

(continues on next page)

(continued from previous page)

```
    "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.
    "natural"."London" ... Done.
    "places"."London" ... Done.
```

Delete the test database and downloaded data files:

```
>>> # 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.

>>> # 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.fetch_osm_data

PostgresOSM.**fetch_osm_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*, ***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`

Returns
PBF (.osm.pbf) data

Return type
dict | collections.OrderedDict

Examples:

```
>>> from pydriosm.ios import PostgresOSM
>>> from pyhelpers.dirs import delete_dir

>>> osmdb = PostgresOSM(database_name='osmdb_test')
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_subregion_osm_pbf(subrgn_name, data_dir=dat_dir, verbose=True)
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"
    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)

>>> # Import shapefile data of Rutland
>>> rutland_shp = osmdb.reader.read_shp_zip(
...     subrgn_name, data_dir=dat_dir, rm_extracts=True, verbose=True)
Downloading "rutland-latest-free.shp.zip"
    to "tests\osm_data\rutland\" ... Done.
Extracting "tests\osm_data\rutland\rutland-latest-free.shp.zip"
    to "tests\osm_data\rutland\rutland-latest-free-shp\" ... Done.
Reading the shapefile(s) at
    "tests\osm_data\rutland\rutland-latest-free-shp\" ... Done.
Deleting the extracts "tests\osm_data\rutland\rutland-latest-free-shp\" ... Done.
```

(continues on next page)

(continued from previous page)

```
>>> osmdb.import_osm_data(rutland_shp, table_name=subrgn_name, verbose=True)
To import data into table "Rutland" 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)
    "pofw" ... Done. (<total of rows> features)
    "pois" ... Done. (<total of rows> features)
    "railways" ... Done. (<total of rows> features)
    "roads" ... Done. (<total of rows> features)
    "traffic" ... Done. (<total of rows> features)
    "transport" ... Done. (<total of rows> features)
    "water" ... Done. (<total of rows> features)
    "waterways" ... Done. (<total of rows> features)

>>> # Retrieve the data of specific layers
>>> lyr_names = ['points', 'multipolygons']
>>> rutland_data_ = osmdb.fetch_osm_data(subrgn_name, lyr_names, verbose=True)
Fetching the data of "Rutland" ...
    "points" ... Done.
    "multipolygons" ... Done.
>>> type(rutland_data_)
collections.OrderedDict
>>> 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_osm_data(subrgn_name, layer_names=None, verbose=True)
Fetching the data of "Rutland" ...
    "points" ... Done.
    "lines" ... Done.
    "multilinestrings" ... Done.
    "multipolygons" ... Done.
    "other_relations" ... Done.
    "buildings" ... Done.
    "landuse" ... Done.
    "natural" ... Done.
    "places" ... Done.
    "pofw" ... Done.
    "pois" ... Done.
    "railways" ... Done.
    "roads" ... Done.
    "traffic" ... Done.
    "transport" ... Done.
    "water" ... Done.
    "waterways" ... Done.
>>> type(rutland_data)
```

(continues on next page)

(continued from previous page)

```
collections.OrderedDict
>>> list(rutland_data.keys())
['points',
 'lines',
 'multilinestrings',
 'multipolygons',
 'other_relations',
 'buildings',
 'landuse',
 'natural',
 'places',
 'pofw',
 'pois',
 'railways',
 'roads',
 'traffic',
 'transport',
 'water',
 'waterways']

>>> # Data of the 'waterways' layer
>>> rutland_waterways = rutland_data['waterways']
>>> rutland_waterways.head()
    osm_id  code  ...                                        coordinates  shape_type
0  3701346  8102  ...  [(-0.7536654, 52.6495358), (-0.7536236, 52.649...           3
1  3701347  8102  ...  [(-0.7948821, 52.6569468), (-0.7946128, 52.656...           3
2  3707149  8103  ...  [(-0.7262381, 52.6790459), (-0.7258244, 52.680...           3
3  3707303  8102  ...  [(-0.7213277, 52.6765954), (-0.7206778, 52.676...           3
4  4470795  8101  ...  [(-0.4995349, 52.6418825), (-0.4984075, 52.642...           3
[5 rows x 7 columns]
```

Delete the test database and downloaded data files:

```
>>> # 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.

>>> # 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 the methods *import_osm_data()* and *import_subregion_osm_pbf()*.
- Similar examples about *fetching data from the database* are available in *Quick start*.

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 of 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 import PostgresOSM

>>> osmdb = PostgresOSM(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)
```

(continues on next page)

(continued from previous page)

```
>>> type(tbl_col_info_dict)
dict
>>> list(tbl_col_info_dict.keys())[:5]
['table_catalog',
 'table_schema',
 'table_name',
 'column_name',
 'ordinal_position']

>>> # 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 import PostgresOSM

>>> osmdb = PostgresOSM(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)
```

(continues on next page)

(continued from previous page)

```
>>> 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.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*, ***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`
- **kwargs** – [optional] parameters of the method *import_osm_layer()*

Examples:

```
>>> from pydriosm.ios import PostgresOSM
>>> from pyhelpers.dirs import delete_dir

>>> osmdb = PostgresOSM(database_name='osmdb_test')
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_osm_pbf(subrgn_name, dat_dir, verbose=True)
Downloading "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)
To import data into 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)

>>> # Get parsed PBF data
>>> parsed_rutland_pbf = osmdb.reader.read_osm_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)
To import data into table "Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
    "schema_0" ... Done. (<total of rows> features)
    "schema_1" ... Done. (<total of rows> features)
    "schema_2" ... Done. (<total of rows> features)
```

(continues on next page)

(continued from previous page)

```
>>> # 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 data of a shapefile:

```
>>> # Read shapefile data of Rutland
>>> rutland_shp = osmdb.reader.read_shp_zip(
...     subregion_name=subrgn_name, data_dir=dat_dir, rm_extracts=True, verbose=True)
Downloading "rutland-latest-free.shp.zip"
    to "tests\osm_data\rutland\" ... Done.
Extracting "tests\osm_data\rutland\rutland-latest-free.shp.zip"
    to "tests\osm_data\rutland\rutland-latest-free-shp\" ... Done.
Reading the shapefile(s) at
    "tests\osm_data\rutland\rutland-latest-free-shp\" ... Done.
Deleting the extracts "tests\osm_data\rutland\rutland-latest-free-shp\" ... Done.
>>> type(rutland_shp)
collections.OrderedDict
>>> list(rutland_shp.keys())
['buildings',
 'landuse',
 'natural',
 'places',
 'pofw',
 'pois',
 'railways',
 'roads',
 'traffic',
 'transport',
 'water',
 'waterways']

>>> # Import all layers of the shapefile data of Rutland
>>> osmdb.import_osm_data(osm_data=rutland_shp, table_name=subrgn_name, verbose=True)
To import data into table "Rutland" 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)
    "pofw" ... Done. (<total of rows> features)
    "pois" ... Done. (<total of rows> features)
    "railways" ... Done. (<total of rows> features)
    "roads" ... Done. (<total of rows> features)
    "traffic" ... Done. (<total of rows> features)
    "transport" ... Done. (<total of rows> features)
    "water" ... Done. (<total of rows> features)
    "waterways" ... Done. (<total of rows> features)
```

Example 3 - Import BBBike shapefile data file of Leeds:

```
>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
```

(continues on next page)

(continued from previous page)

```
>>> subrgn_name = 'Leeds'

>>> # Read shapefile data of Leeds
>>> leeds_shp = osmdb.reader.read_shp_zip(
...     subregion_name=subrgn_name, data_dir=dat_dir, rm_extracts=True, verbose=True)
Downloading "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.
Deleting the extracts "tests\osm_data\leeds\Leeds-shp\" ... Done.
>>> type(leeds_shp)
collections.OrderedDict
>>> 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)
To import data into table "Leeds" 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 the test database and downloaded data files:

```
>>> # 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.

>>> # 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'*, *force_replace=False*, *chunk_size=None*, *confirmation_required=True*, *verbose=False*, ***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, 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`
- **kwargs** – [optional] parameters of pyhelpers.sql.PostgreSQL.dump_data()

Examples:

```
>>> from pydriosm.ios import PostgresOSM
>>> from pyhelpers.dirs import delete_dir

>>> osmdb = PostgresOSM(database_name='osmdb_test')
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)
```

(continues on next page)

(continued from previous page)

```
>>> raw_pbf = osmdb.reader.read_osm_pbf(subrgn_name, data_dir=dat_dir, verbose=True)
Downloading "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'
>>> 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 "points"."Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Creating a schema: "points" ... Done.
Importing the data into the table "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_osm_pbf(
...     subregion_name=subrgn_name, data_dir=dat_dir, expand=True, parse_geometry=True)
>>> type(parsed_pbf)
dict
>>> list(parsed_pbf.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']
>>> parsed_pbf_points = parsed_pbf[points_key]  # Get the parsed data of 'points' layer
>>> type(parsed_pbf_points)
pandas.core.series.Series
>>> parsed_pbf_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]

>>> # 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,
...     verbose=True, if_exists='replace')
To import data into "points"."Rutland" at postgres:***@localhost:5432/osmdb_test
```

(continues on next page)

(continued from previous page)

```
? [No]|Yes: yes
The table "points"."Rutland" already exists and is replaced.
Importing the data into the table "points"."Rutland" ... 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
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 shapefile*:

```
>>> # Read the data of 'railways' layer and delete the extracts
>>> lyr_name = 'railways'
>>> rutland_railways_shp = osmdb.reader.read_shp_zip(
...     subregion_name=subrgn_name, layer_names=lyr_name, data_dir=dat_dir,
...     rm_extracts=True, verbose=True)
Downloading "rutland-latest-free.shp.zip"
    to "tests\osm_data\rutland\" ... Done.
Extracting the following layer(s):
    'railways'
    from "tests\osm_data\rutland\rutland-latest-free.shp.zip"
      to "tests\osm_data\rutland\rutland-latest-free-shp\" ... Done.
Reading "tests\osm_data\rutland\rutland-latest-free-shp\gis_osm_railways_free_1.s...
Deleting the extracts "tests\osm_data\rutland\rutland-latest-free-shp\" ... Done.
>>> type(rutland_railways_shp)
collections.OrderedDict
>>> list(rutland_railways_shp.keys())
['railways']

>>> # Get the data of 'railways' layer
>>> rutland_railways_shp_ = rutland_railways_shp[lyr_name]
>>> rutland_railways_shp_.head()
    osm_id  code  ...                                        coordinates shape_type
0  2162114  6101  ...  [(-0.4528083, 52.6993402), (-0.4521571, 52.698...          3
1  3681043  6101  ...  [(-0.6531215, 52.5730787), (-0.6531793, 52.572...          3
2  3693985  6101  ...  [(-0.7323403, 52.6782102), (-0.7319059, 52.678...          3
3  3693986  6101  ...  [(-0.6173072, 52.6132317), (-0.6241869, 52.614...          3
4  4806329  6101  ...  [(-0.4576926, 52.7035194), (-0.4565358, 52.702...          3
[5 rows x 9 columns]

>>> # Import the 'railways' data into the PostgreSQL database
>>> osmdb.import_osm_layer(
...     layer_data=rutland_railways_shp_, table_name=subrgn_name, schema_name=lyr_name,
...     verbose=True)
To import data into "railways"."Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Creating a schema: "railways" ... Done.
Importing the data into the table "railways"."Rutland" ... 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_7    column_8
table_catalog     osmdb_test  osmdb_test  ...   osmdb_test  osmdb_test
```

(continues on next page)

(continued from previous page)

```
table_schema        railways    railways  ...     railways    railways
table_name           Rutland     Rutland  ...      Rutland     Rutland
column_name           osm_id        code  ...  coordinates  shape_type
ordinal_position           1           2  ...            8           9
[5 rows x 9 columns]
```

Delete the test database and downloaded data files:

```
>>> # 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.

>>> # 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_subregion_osm_pbf

PostgresOSM.**import_subregion_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_subregion_osm_pbf() or _import_subregion_osm_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')
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_subregion_osm_pbf(subrgn_name, data_dir=dat_dir, verbose=True)
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"
    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)
```

Example 2 - Import PBF data of Leeds and London:

```
>>> # Change the data source
>>> osmdb.data_source = 'BBBike'
>>> subrgn_names = ['Leeds', 'London']
```

(continues on next page)

(continued from previous page)

```
>>> # Note this may take a few minutes (or longer)
>>> osmdb.import_subregion_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)
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"
    to "tests\osm_data\leeds\" ... Done.
Reading "tests\osm_data\leeds\Leeds.osm.pbf" ... Done.
Importing the data into table "Leeds" ...
    "points" ... Done. (82137 features)
    "lines" ... Done. (164411 features)
    "multilinestrings" ... Done. (390 features)
    "multipolygons" ... Done. (439144 features)
    "other_relations" ... Done. (6938 features)
Saving "Leeds-pbf.pickle" to "tests\osm_data\leeds\" ... Done.
Deleting "tests\osm_data\leeds\Leeds.osm.pbf" ... Done.
Downloading "London.osm.pbf"
    to "tests\osm_data\london\" ... Done.
Importing the data of "London" chunk-wisely
  into postgres:***@localhost:5432/osmdb_test ...
    "points" ... Done. (654517 features)
    "lines" ... Done. (769631 features)
    "multilinestrings" ... Done. (7241 features)
    "multipolygons" ... Done. (5432 features)
    "other_relations" ... Done. (21792 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                   None
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 (-1.514124 53.7416937)  ...     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
```

(continues on next page)

(continued from previous page)

```
>>> 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)
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.post_process_layer_dat

PostgresOSM.**post_process_layer_dat**(*layer_dat*, *decode_geojson=True*, *sort_by='id'*)

Post-process the data of a specific layer.

Parameters

- **layer_dat** –
- **decode_geojson** –
- **sort_by** –

Returns

Return type

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 import PostgresOSM

>>> osmdb = PostgresOSM(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.
```

Customised alternatives (Optional)

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')
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> type(gfi.dbms)
pydriosm.ios.PostgresOSM

>>> gfi.dbms.name
'Geofabrik OpenStreetMap data extracts'
```

See also:

- Examples for all the methods of the class *PostgresOSM*.

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')
```

(continues on next page)

(continued from previous page)

```
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> type(bbi.dbms)
pydriosm.ios.PostgresOSM

>>> bbi.dbms.name
'BBBike exports of OpenStreetMap data'
```

See also:

- Examples for all the methods of the class *PostgresOSM*.

Other utilities

Utilities for the *ios* module.

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.

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 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')
```

(continues on next page)

(continued from previous page)

```
>>> 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 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 import validate_table_name

>>> subrgn_name = 'greater london'
>>> valid_table_name = validate_table_name(subrgn_name)
>>> valid_table_name
'greater london'

>>> subrgn_name = 'Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch, Wales'
>>> valid_table_name = validate_table_name(subrgn_name, sub_space='_')
>>> valid_table_name
'Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch_W..'
```

2.2 Modules

errors	Define custom errors/exceptions.
utils	Provide various helper functions for use across the package.

2.2.1 errors

Define custom errors/exceptions.

Validation errors

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
```

(continues on next page)

(continued from previous page)

```
>>> 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', ...
```

Parse errors

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')
Traceback (most recent call last):
  ...
```

(continues on next page)

(continued from previous page)

```
pydriosm.errors.OtherTagsReformatError:
  `other_tags='abc'` -> Failed to reformat the ...
```

2.2.2 utils

Provide various helper functions for use across the package.

Check data pathnames

check_relpath(pathname[, start])	Check and return a relative pathname to the given pathname.
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.

check_relpath

pydriosm.utils.**check_relpath**(*pathname*, *start='.'*)

Check and return a relative pathname to the given pathname.

On Windows, when pathname and start are on different drives, the function returns the given pathname.

Parameters

- **pathname** (*str | os.PathLike[str]*) – pathname of a file or a directory
- **start** (*str | os.PathLike[str]*) – optional start directory, defaults to os.curdir (i.e. the current working directory)

Returns

relative pathname to the given pathname

Type

str | os.PathLike[str]

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'
```

General utilities

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
```

Chapter 3

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.

Chapter 4

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.

Chapter 5

Contributors

- Qian Fu

Chapter 6

Quick start

For a demonstration of how PyDriosm works with OpenStreetMap (OSM) data, this section of the documentation provides a quick guide with practical examples. It showcases the usage of the package for tasks such as downloading, parsing, and storage I/O of OSM data.

(Also check out GitHub and Documentation.)

Note:

- All the data used in this quick-start tutorial will be downloaded and saved to a directory named **"tests\osm_data\"** (which will be created if it does not exist) at the current working directory.
- At the end of the tutorial, you will be asked to confirm whether you would like to retain or remove the directory (i.e. **"tests\osm_data\"**). If ***yes***, all the downloaded data and those generated during the tutorial will be deleted permanently.

6.1 Download data

The current release of the package works for the (sub)region-based OSM data extracts, which are available from the free download servers: Geofabrik and BBBike. To start with, let's use the class *GeofabrikDownloader* from the module *downloader* to download a data file from the Geofabrik free download server.

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

>>> # Create an instance for downloading the Geofabrik free data extracts
>>> gfd = GeofabrikDownloader()

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

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

To explore what data is available for download, you may check a download catalogue by using the method *GeofabrikDownloader.get_catalogue()*:

```
>>> # A download catalogue for all subregions
>>> geofabrik_download_catalogue = gfd.get_catalogue()
>>> geofabrik_download_catalogue.head()
               subregion  ...                                          .osm.bz2
0                 Africa  ...  https://download.geofabrik.de/africa-latest.os...
1             Antarctica  ...  https://download.geofabrik.de/antarctica-lates...
2                   Asia  ...  https://download.geofabrik.de/asia-latest.osm.bz2
3  Australia and Oceania  ...  https://download.geofabrik.de/australia-oceani...
4        Central America  ...  https://download.geofabrik.de/central-america-...

[5 rows x 6 columns]
```

If we would like to download a protocolbuffer binary format (PBF) data file of a specific geographic region, we need to specify the name of the (sub)region and the file format (i.e. `".pbf"` or `".osm.pbf"`). For example, let's try to download the PBF data of `'London'` and save it to a directory `"tests\\osm_data"`:

```
>>> subrgn_name = 'London'  # Name of a (sub)region; case-insensitive
>>> file_format = ".pbf"  # OSM data file format
>>> dwnld_dir = "tests\\osm_data"  # Name of or path to a directory where the data is saved

>>> # Download the OSM PBF data of London from Geofabrik download server
>>> gfd.download_osm_data(
...     subregion_names=subrgn_name, osm_file_format=file_format, download_dir=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.
```

Since the data has been successfully downloaded, it will not be downloaded again if you run the method given the same arguments:

```
>>> gfd.download_osm_data(
...     subregion_names=subrgn_name, osm_file_format=file_format, download_dir=dwnld_dir,
...     verbose=True)
"greater-london-latest.osm.pbf" is already available
    at "tests\osm_data\greater-london\".
```

Note:

- If the data file does not exist at the specified directory, we would need to confirm whether to proceed to download it as, by default, `confirmation_required=True`. To skip the confirmation requirement, we could set `confirmation_required=False`.
- The parameter `download_dir` is by default `None`, in which case the downloaded data file is saved to the default data directory. For example, the default directory for in the case above should be **"geofabrik\europe\great-britain\england\greater-london\"**.
- 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**.
- The parameter `update` is by default `False`. When the data file already exists at the specified or default download directory and we set `update=True`, the method would replace the existing file with a freshly downloaded one.

If we would also like to have the path to the downloaded file, we could set `ret_download_path=True`. See the example below:

```
>>> path_to_london_pbf = gfd.download_osm_data(
...     subregion_names=subrgn_name, osm_file_format=file_format, download_dir=dwnld_dir,
...     update=True, verbose=2, ret_download_path=True)
"greater-london-latest.osm.pbf" is already available
    at "tests\osm_data\greater-london\".
To update the .osm.pbf data of the following geographic (sub)region(s):
    Greater London
? [No]|Yes: yes
Updating "greater-london-latest.osm.pbf"
    at "tests\osm_data\greater-london\" ...
"tests\osm_data\greater-london\greater-london-latest.osm.pbf": 82.9MB [00:01, 52.8MB/s]
Done.
```

In the example above, `update=True` allowed us to download the PBF data file again and replace the existing one. In addition, we also set `verbose=2`, which requires tqdm, to print more details about the downloading process.

Now let's 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) file path: "{os.path.relpath(path_to_london_pbf_)}"')
Current (relative) file 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"
```

Alternatively, you could also make use of the method *`.get_default_pathname()`* to get the default path to the data file (even when it does not exist):

We could also make use of the method *`get_default_pathname()`* to directly 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) file path: "{os.path.relpath(london_pbf_pathname)}"')
Current (relative) file 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, let's now download the PBF data of both `'West Yorkshire'` and `'West Midlands'`, and return their file paths:

```
>>> subrgn_names = ['West Yorkshire', 'West Midlands']
>>> paths_to_pbf = gfd.download_osm_data(
...     subregion_names=subrgn_names, osm_file_format=file_format, download_dir=dwnld_dir,
```

(continues on next page)

(continued from previous page)

```
...     verbose=True, ret_download_path=True)
To download .osm.pbf data of the following geographic (sub)region(s):
    West Yorkshire
    West Midlands
? [No]|Yes: yes
Downloading "west-yorkshire-latest.osm.pbf"
    to "tests\osm_data\west-yorkshire\" ... Done.
Downloading "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"
```

6.2 Read/parse data

To read/parse any of the downloaded data files above, we can use the class *PBFReadParse* or *GeofabrikReader*, which requires the python package GDAL.

6.2.1 PBF data (.pbf / .osm.pbf)

Now, let's try to use the method *GeofabrikReader.read_osm_pbf()* to read the PBF data of the subregion `'Rutland'`:

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

>>> # Create an instance for reading the downloaded Geofabrik data extracts
>>> gfr = GeofabrikReader()

>>> subrgn_name = 'Rutland'
>>> dat_dir = dwnld_dir  # i.e. "tests\\osm_data"

>>> rutland_pbf_raw = gfr.read_osm_pbf(
...     subregion_name=subrgn_name, data_dir=dat_dir, verbose=True)
Downloading "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\t{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\t{raw_data_keys}')
The "keys" of `rutland_pbf_parsed`:
    ['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']
```

(continues on next page)

(continued from previous page)

```
>>> raw_layer_data_type = type(rutland_pbf_raw['points'])
>>> print(f'Data type of the corresponding layer:\n\t{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\t{raw_value_type}')
Data type of the individual feature:
    <class 'osgeo.ogr.Feature'>
```

As we see from the above, the variable `rutland_pbf_raw` is in dict type. It has five keys: `'points'`, `'lines'`, `'multilinestrings'`, `'multipolygons'` and `'other_relations'`, each of which corresponds to the name of a layer of the PBF data.

However, the raw data is not human-readable. We can set `readable=True` to parse the individual features using GDAL.

Note:

- The method *`GeofabrikReader.read_osm_pbf()`*, which relies on GDAL, may take tens of minutes (or even much longer) to parse a PBF data file, depending on the size of the data file.
- If the size of a data file is greater than the specified `chunk_size_limit` (which defaults to `50` MB), the data will be parsed in a chunk-wise manner.

```
>>> # Set `readable=True`
>>> rutland_pbf_parsed_0 = gfr.read_osm_pbf(
...     subregion_name=subrgn_name, data_dir=dat_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\t{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\t{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\t{parsed_layer_type}')
Data type of the corresponding layer:
    <class 'pandas.core.series.Series'>
```

Let's further check out 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...
```

(continues on next page)

(continued from previous page)

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

>>> rutland_pbf_points_0_0 = rutland_pbf_points_0[0]  # A feature of the 'points' layer
>>> rutland_pbf_points_0_0
{'type': 'Feature',
 'geometry': {'type': 'Point', 'coordinates': [-0.5134241, 52.6555853]},
 'properties': {'osm_id': '488432',
  'name': None,
  'barrier': None,
  'highway': None,
  'ref': None,
  'address': None,
  'is_in': None,
  'place': None,
  'man_made': None,
  'other_tags': '"odbl"=>"clean"'},
 'id': 488432}
```

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

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

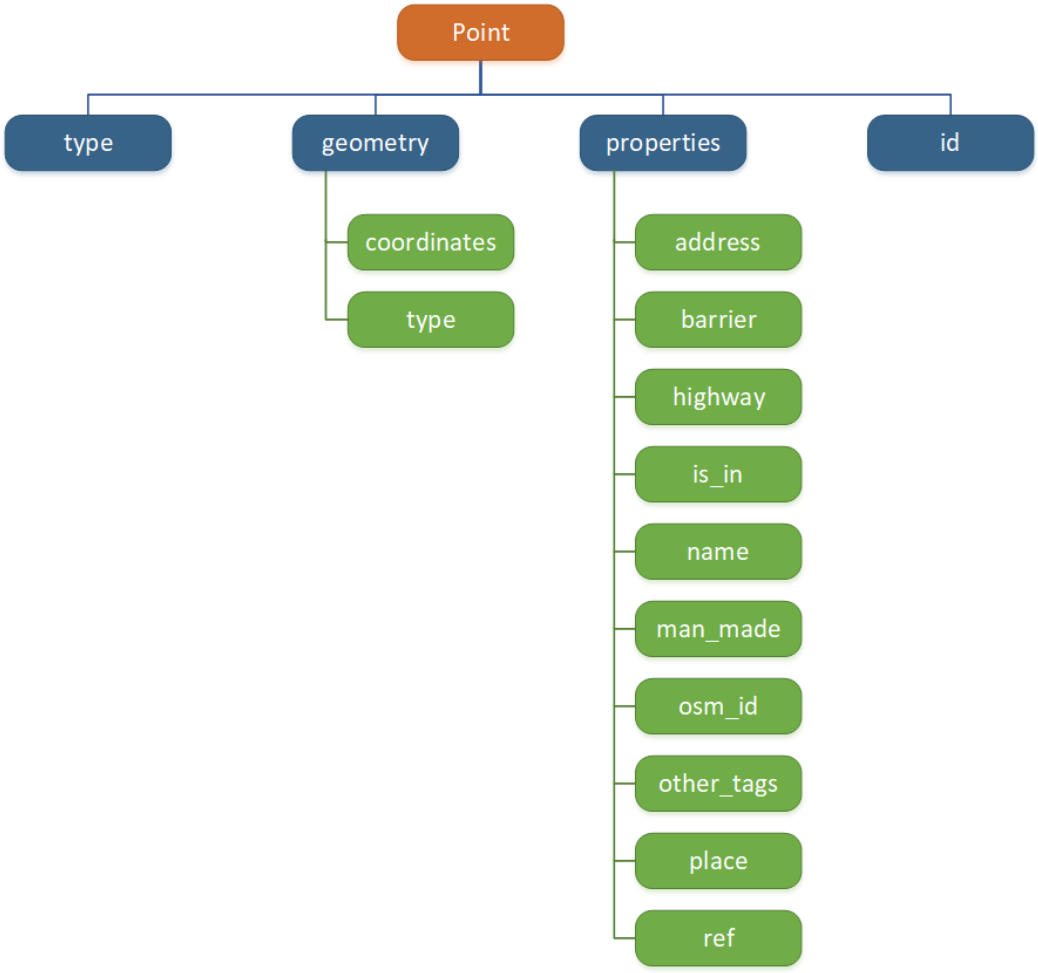

Fig. 1: Type of the geometry object and keys within the nested dictionary of `'points'`.

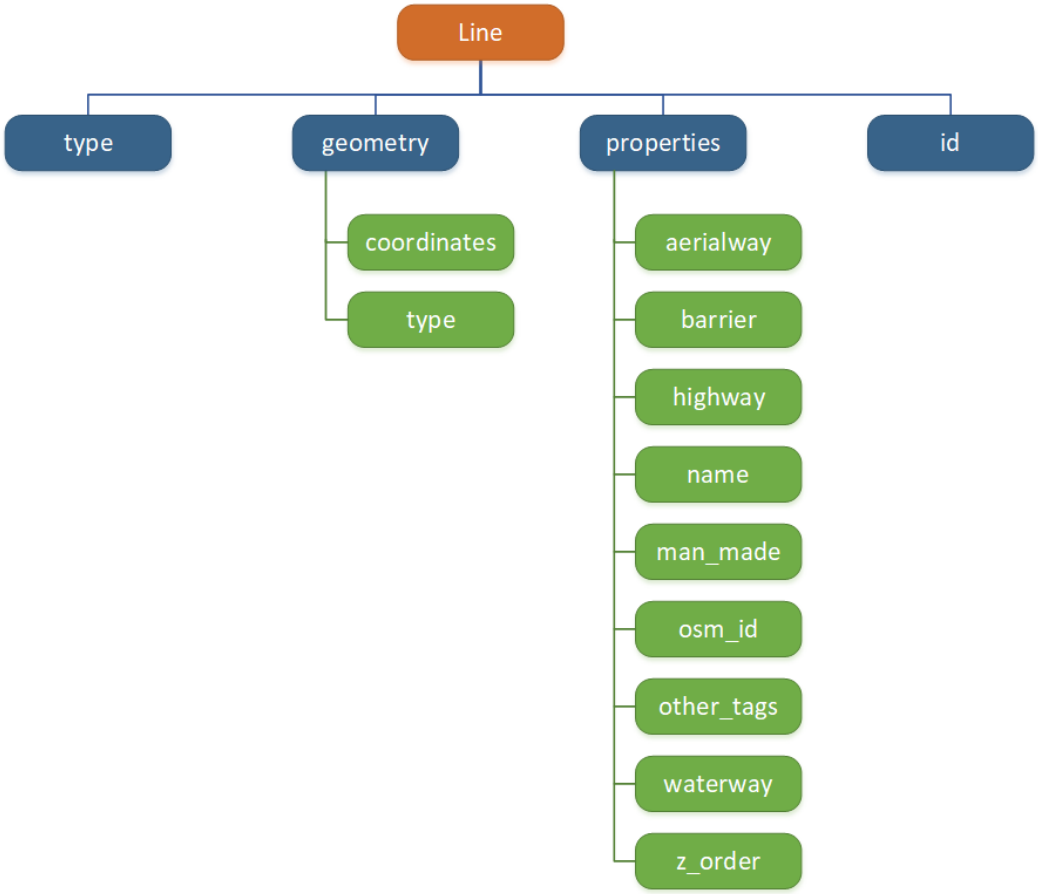

Fig. 2: Type of the geometry object and keys within the nested dictionary of `'lines'`.

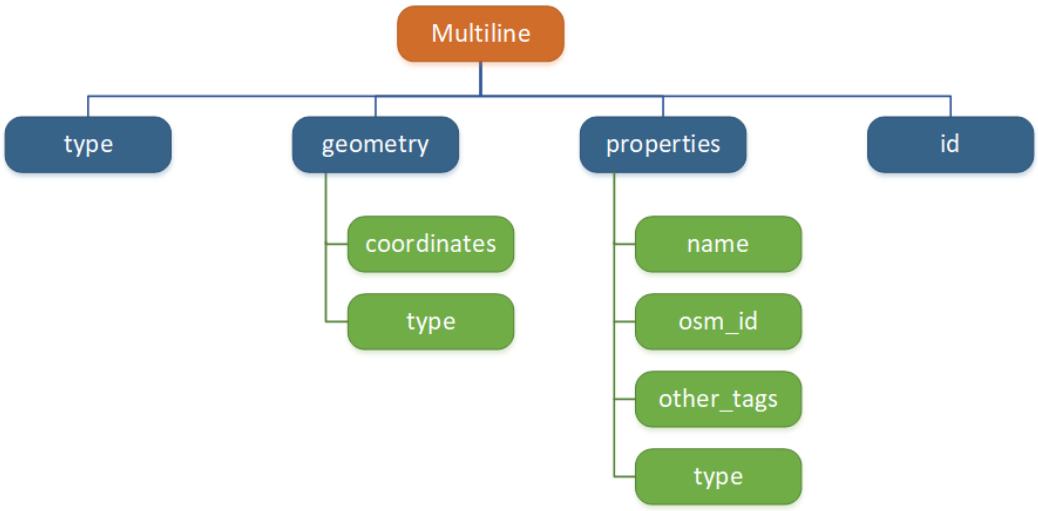

Fig. 3: Type of the geometry object and keys within the nested dictionary of `'multilinestrings'`.

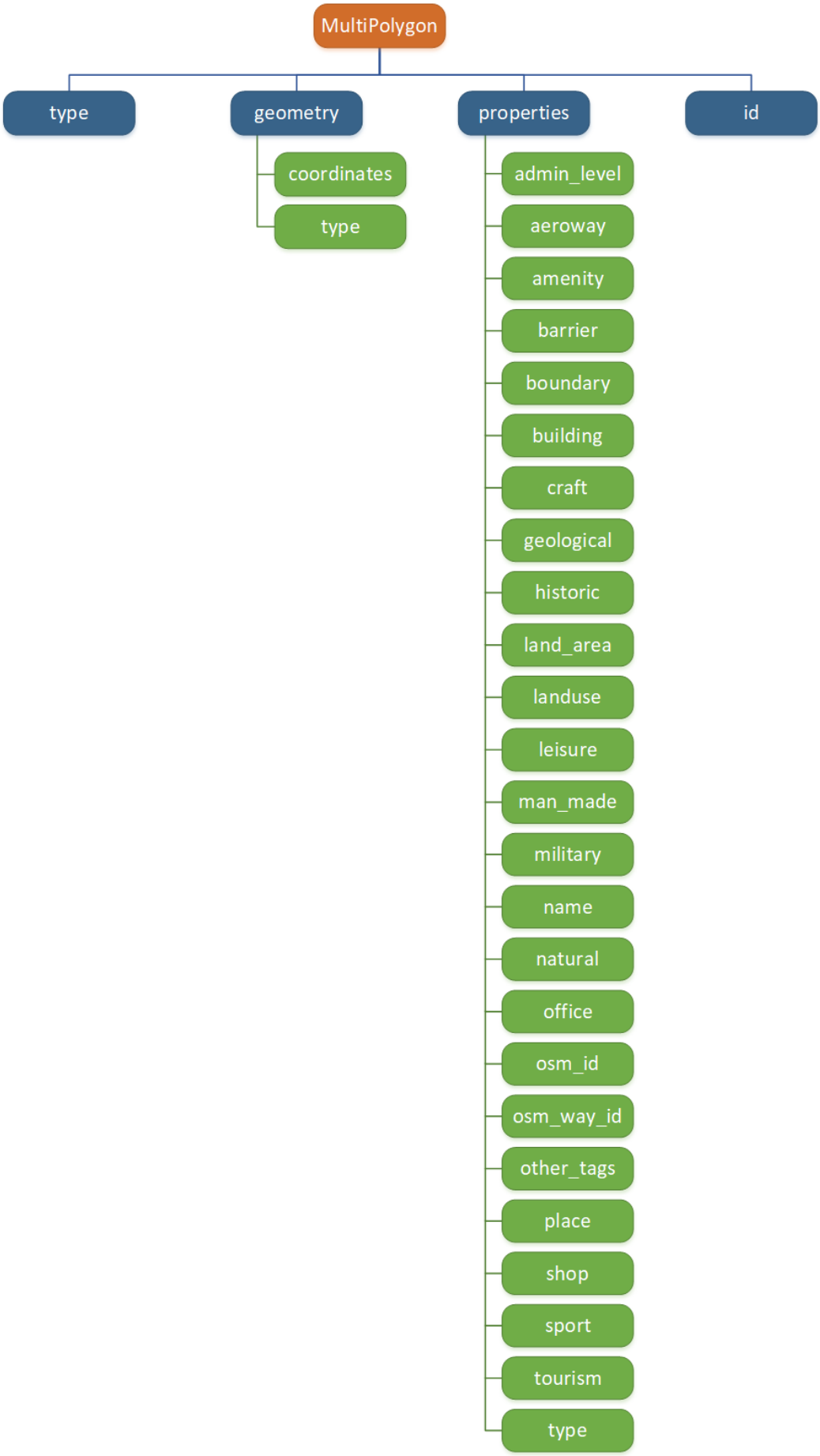

Fig. 4: Type of the geometry object and keys within the nested dictionary of `'multipolygons'`.

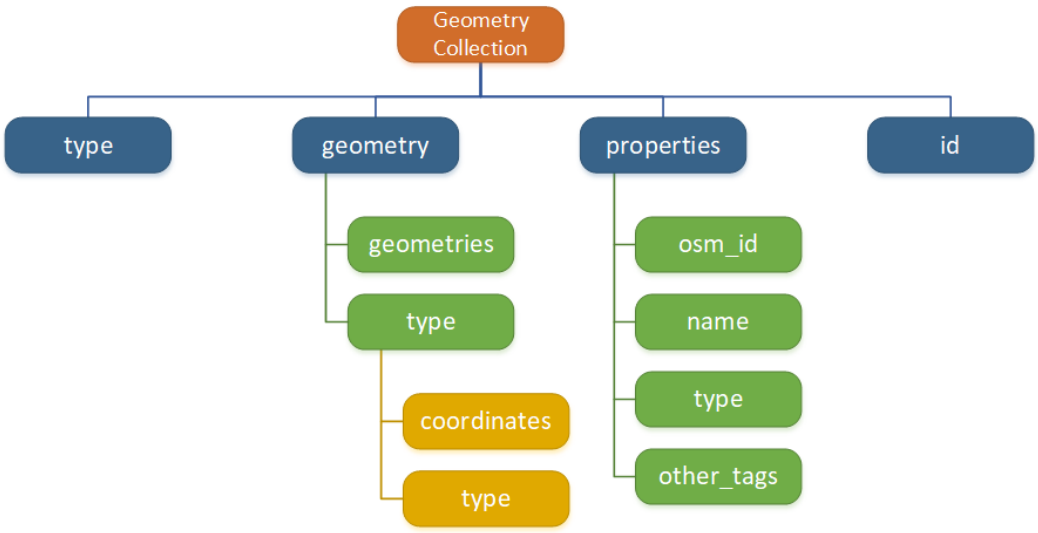

Fig. 5: Type of the geometry object and keys within the nested dictionary of `'other_relations'`.

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_osm_pbf(
...     subregion_name=subrgn_name, data_dir=dat_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    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]

>>> rutland_pbf_points_1['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
```

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_osm_pbf(
...     subrgn_name, data_dir=dat_dir, expand=True, parse_geometry=True, verbose=True)
>>> rutland_pbf_points_2 = rutland_pbf_parsed_2['points']
Parsing "tests\osm_data\rutland\rutland-latest.osm.pbf" ... Done.
```

(continues on next page)

(continued from previous page)

```
>>> rutland_pbf_points_2['geometry'].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]

>>> rutland_pbf_points_2['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 (-0.7266581 52.6695058)
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 geographic (sub)region is provided, e.g. `rutland_pbf = gfr.read_osm_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`.

6.2.2 Shapefiles (.shp.zip / .shp)

To read shapefile data, we can use the method *GeofabrikReader.read_shp_zip()* or *SHPReadParse.read_shp()*, which relies on PyShp (or optionally, GeoPandas.

Note:

- GeoPandas 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 *GeofabrikReader.read_shp_zip()*:

```
>>> subrgn_name = 'London'
>>> lyr_name = 'railways'
```

(continues on next page)

(continued from previous page)

```
>>> london_shp = gfr.read_shp_zip(
...     subregion_name=subrgn_name, layer_names=lyr_name, data_dir=dat_dir, verbose=True)
Downloading "greater-london-latest-free.shp.zip"
    to "tests\osm_data\greater-london\" ... Done.
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.
Reading "tests\osm_data\greater-london\greater-london-latest-free-shp\gis_osm_railways_free_1...
```

Check the data:

```
>>> data_type = type(london_shp)
>>> print(f'Data type of `london_shp`:\n\t{data_type}')
Data type of `london_shp`:
    <class 'collections.OrderedDict'>

>>> data_keys = list(london_shp.keys())
>>> print(f'The "keys" of `london_shp`:\n\t{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\t{layer_type}")
Data type of the 'railways' layer:
    <class 'pandas.core.frame.DataFrame'>
```

Similar to the parsed PBF data, `london_shp` is also in dict type, with the `layer_name` being its key by default.

```
>>> london_railways_shp = london_shp[lyr_name]  # london_shp['railways']
>>> london_railways_shp.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]
```

Note:

- When `layer_name=None` (default), all layers will be included.
- The parameter `feature_names` is related to `'fclass'` in `london_railways_shp`. You can specify one feature name (or multiple feature names) to get a subset of `london_railways_shp`.
- If the method *`GeofabrikReader.read_shp_zip()`* could not find the target *.shp* file at the default or specified directory (i.e. `dat_dir`), it will try to extract the *.shp* file from the *.shp.zip* file.
- If the *.shp.zip* file is not available either, the method *`GeofabrikReader.read_shp_zip()`* will try download the data first, provided that `download=True`; otherwise, setting `update=True` would allow the method to download the latest version of the data despite the availability of the *.shp.zip* file.

- If you'd like to delete the *.shp* files and/or the downloaded *.shp.zip* file, set the parameters `rm_extracts=True` and/or `rm_shp_zip=True`.

If we would like to combine multiple (sub)regions over a certain layer, we can use the method *GeofabrikReader.merge_subregion_layer_shp()* to concatenate the *.shp* files of the specific layer.

For example, let's now merge the `'railways'` layers of `'London'` and `'Kent'`:

```
>>> subrgn_names = ['London', 'Kent']
>>> lyr_name = 'railways'

>>> path_to_merged_shp = gfr.merge_subregion_layer_shp(
...     subregion_names=subrgn_names, layer_name=lyr_name, data_dir=dat_dir, verbose=True,
...     ret_merged_shp_path=True)
"greater-london-latest-free.shp.zip" is already available
    at "tests\osm_data\greater-london\".
To download .shp.zip data of the following geographic (sub)region(s):
    Kent
? [No]|Yes: yes
Downloading "kent-latest-free.shp.zip"
    to "tests\osm_data\kent\" ... Done.
Merging the following shapefiles:
    "greater-london_gis_osm_railways_free_1.shp"
    "kent_gis_osm_railways_free_1.shp"
        In progress ... Done.
        Find the merged shapefile at "tests\osm_data\gre_lon-ken-railways\".

>>> # Relative path of the merged shapefile
>>> print(f"\"{os.path.relpath(path_to_merged_shp)}\"")
"tests\osm_data\gre_lon-ken-railways\linestring.shp"
```

We can read the merged shapefile data by using the method *SHPReadParse.read_layer_shps()*:

```
>>> from pydriosm.reader import SHPReadParse  # from pydriosm import SHPReadParse

>>> london_kent_railways = SHPReadParse.read_layer_shps(path_to_merged_shp)
>>> london_kent_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]
```

For more details, also check out the methods *SHPReadParse.merge_shps()* and *SHPReadParse.merge_layer_shps()*.

6.3 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')
Password (postgres@localhost:5432): ***
Creating a database: "osmdb_test" ... Done.
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.
```

The example is illustrated in Fig. 6:

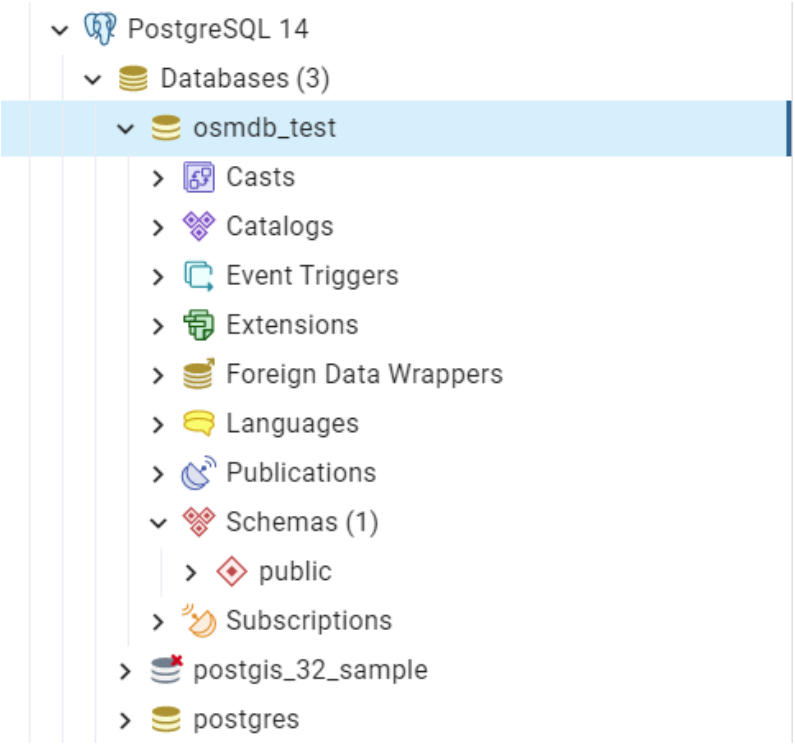

Fig. 6: An illustration of the database named *'osmdb_test'*.

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*.

6.3.1 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()`* or *`import_subregion_osm_pbf()`*.

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:

```
>>> subrgn_name = 'Rutland'

>>> osmdb.import_osm_data(
...     rutland_pbf_parsed_1, table_name=subrgn_name, schema_names=None, verbose=True)
To import data into 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 five layers of the PBF data into the database.

In the example above, five schemas are *'points'*, *'lines'*, *'multilinestrings'*, *'multipolygons'* and *'other_relations'*. If they do not exist, they will be 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 Fig. 7); the data of each layer is imported into a table named as "Rutland" under the corresponding schema (as illustrated in Fig. 8).

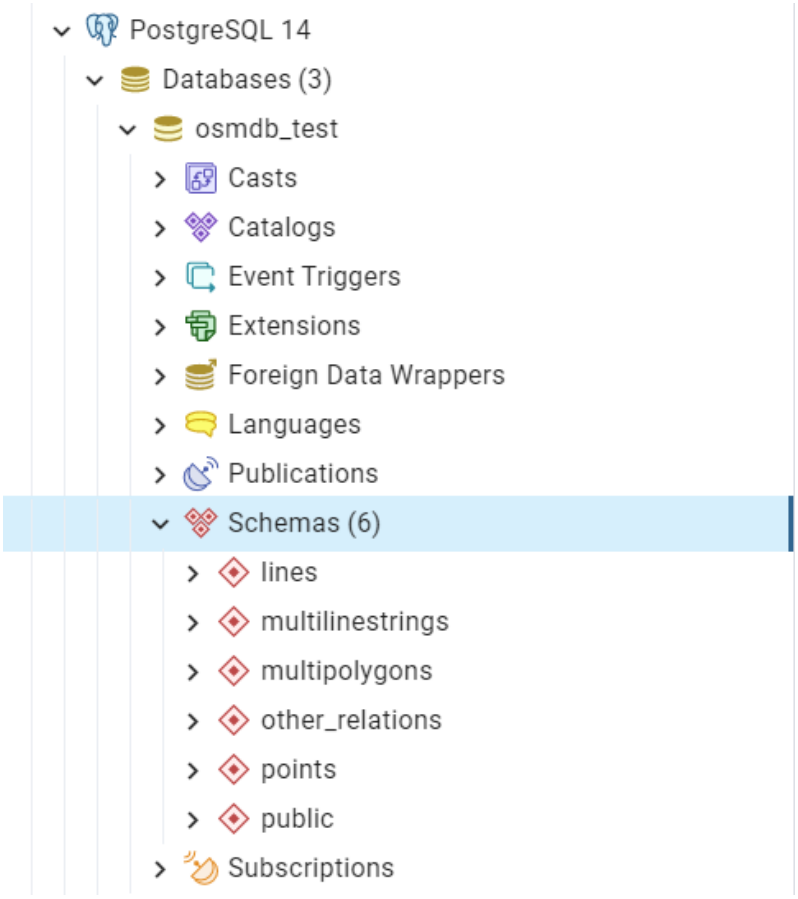

Fig. 7: An illustration of schemas for importing OSM PBF data into a PostgreSQL database.

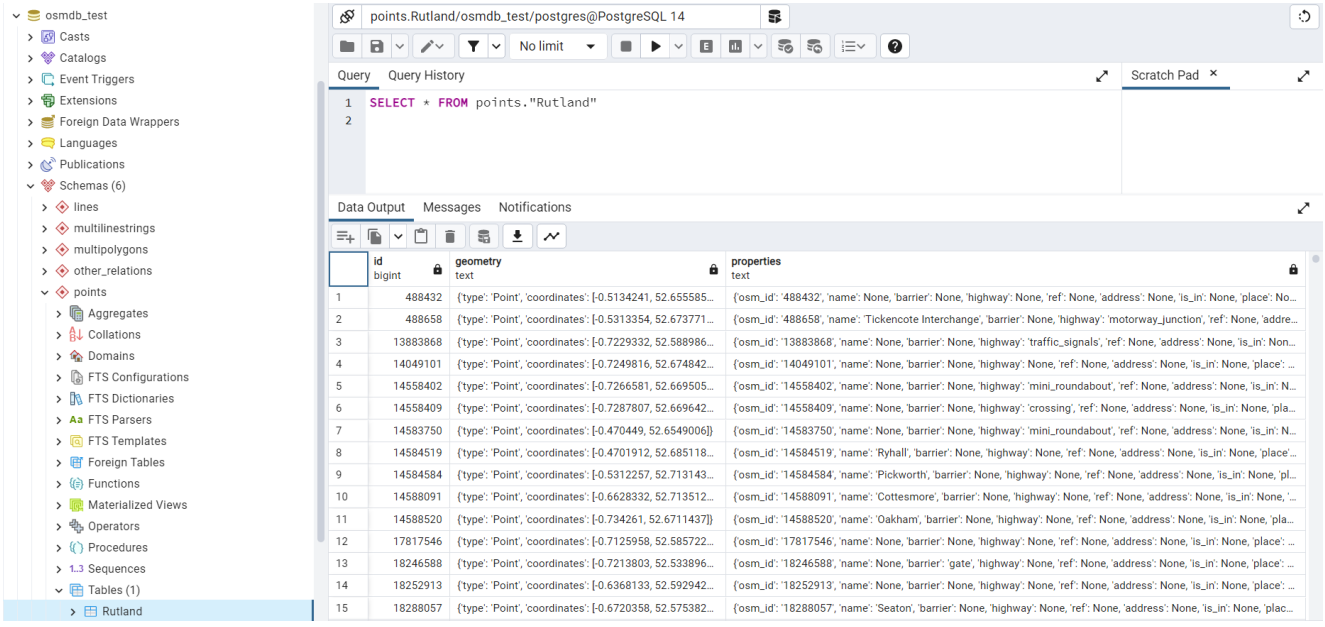

Fig. 8: An illustration of table name for storing the *'points'* layer of the OSM PBF data of Rutland.

6.3.2 Fetch data from the database

To fetch all or specific layers of the imported data, we can use the method *fetch_osm_data()*. For example, let's retrieve all the PBF data of Rutland with `layer_names=None` (by default):

```
>>> # Retrieve the data from the database
>>> rutland_pbf_parsed_1_ = osmdb.fetch_osm_data(subrgn_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\t{retr_data_type}')
Data type of `rutland_pbf_parsed_1_`:
    <class 'collections.OrderedDict'>

>>> retr_data_keys = list(rutland_pbf_parsed_1_.keys())
>>> print(f'The "keys" of `rutland_pbf_parsed_1_`:\n\t{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\t{retr_layer_type}')
Data type of the corresponding layer:
    <class 'pandas.core.frame.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    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]
```

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

```
>>> # 'points', 'lines', 'multilinestrings', 'multipolygons' or 'other_relations'
>>> lyr_name = 'points'

>>> 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_` is equivalent to `rutland_pbf_parsed`: {check_equivalence}")
`rutland_pbf_parsed_` is equivalent to `rutland_pbf_parsed`: 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_osm_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_osm_data()*, the parameter `decode_geojson` is by default `True`.

6.3.3 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 Birmingham shapefile data.

Note:

- `'Birmingham'` 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'

>>> subrgn_name = 'Birmingham'

>>> bham_shp = osmdb.reader.read_shp_zip(subrgn_name, data_dir=dat_dir, verbose=True)
Downloading "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\" ... Done.
```

Check the data *bham_shp*:

```
>>> retr_data_type = type(bham_shp)
>>> print(f'Data type of `bham_shp`:\n\t{retr_data_type}')
Data type of `bham_shp`:
    <class 'collections.OrderedDict'>

>>> retr_data_keys = list(bham_shp.keys())
>>> print(f'The "keys" of `bham_shp`:\n\t{retr_data_keys}')
The "keys" of `bham_shp`:
    ['buildings', 'landuse', 'natural', 'places', 'points', 'railways', 'roads', 'waterways']

>>> retr_layer_type = type(bham_shp[lyr_name])
>>> print(f'Data type of the corresponding layer:\n\t{retr_layer_type}')
Data type of the corresponding layer:
    <class 'pandas.core.frame.DataFrame'>
```

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

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

>>> osmdb.import_osm_data(
```

(continues on next page)

(continued from previous page)

```
...     bham_shp, table_name=subrgn_name, schema_names=lyr_names, verbose=True)
To import data into table "Birmingham" 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 Fig. 9, three schemas: *'railways'*, *'roads'* and *'waterways'* are created in the *'osmdb_test'* database for storing the data of the three shapefile layers of Birmingham.

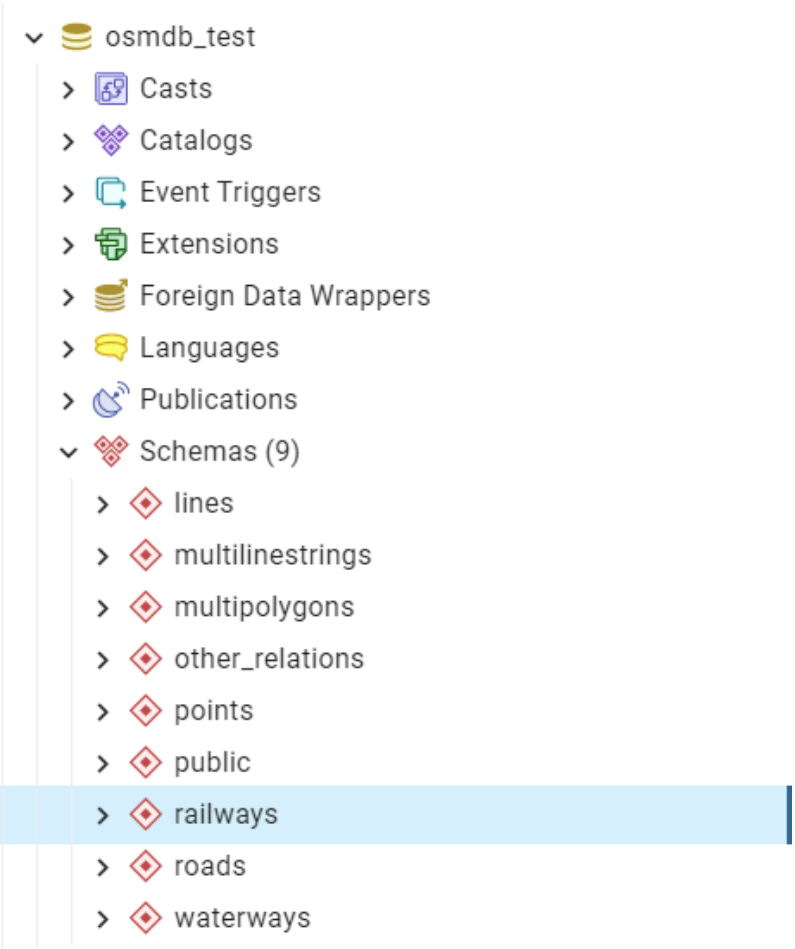

Fig. 9: An illustration of the newly created schemas for the selected layers of Birmingham shapefile data.

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

```
>>> lyr_name = 'railways'

>>> bham_shp_ = osmdb.fetch_osm_data(
...     subrgn_name, layer_names=lyr_name, sort_by='osm_id', verbose=True)
Fetching the data of "Birmingham" ...
    "railways" ... Done.
```

Check the data *bham_shp_*:

```
>>> retr_data_type = type(bham_shp_)
>>> print(f'Data type of `bham_shp_`:\n\t{retr_data_type}')
Data type of `bham_shp_`:
    <class 'collections.OrderedDict'>

>>> retr_data_keys = list(bham_shp_.keys())
>>> print(f'The "keys" of `bham_shp_`:\n\t{retr_data_keys}')
The "keys" of `bham_shp_`:
    ['railways']

>>> # Data frame of the 'railways' layer
>>> bham_shp_railways_ = bham_shp_[lyr_name]
>>> bham_shp_railways_.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]
```

Note:

- `bham_shp_railways` and `bham_shp_railways_` both in pandas.DataFrame type.
- It must be noted that empty strings, `''`, may be automatically saved as `None` when importing `bham_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 `bham_shp_railways_` may not be exactly equal to *bham_shp_railways*. However, they contain exactly the same information. We could sort the data by `'id'` (or `'osm_id'`) to make a comparison (see the test code below).

Check whether `bham_shp_railways_` is equivalent to `bham_shp_railways` (before filling `None` with `''`):

```
>>> bham_shp_railways = bham_shp[lyr_name]

>>> check_eq = bham_shp_railways_.equals(bham_shp_railways)
>>> print(f"`bham_shp_railways_` is equivalent to `bham_shp_railways`: {check_eq}")
`bham_shp_railways_` is equivalent to `bham_shp_railways`: False
```

Let's fill `None` values with `''` and check the equivalence again:

```
>>> # Try filling `None` values with `''`
>>> bham_shp_railways_.fillna('', inplace=True)

>>> # Check again whether `birmingham_shp_railways_` is equal to `birmingham_shp_railways`
>>> check_eq = bham_shp_railways_.equals(bham_shp_railways)
>>> print(f"`bham_shp_railways_` is equivalent to `bham_shp_railways`: {check_eq}")
`bham_shp_railways_` is equivalent to `bham_shp_railways`: True
```

6.3.4 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 Birmingham:

```
>>> # Recall that: subrgn_name == 'Birmingham'; lyr_name == 'railways'
>>> osmdb.drop_subregion_tables(subrgn_name, schema_names=lyr_name, verbose=True)
To drop table "railways"."Birmingham"
  from postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Dropping the table ...
    "railways"."Birmingham" ... Done.
```

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

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

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

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

6.4 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(dat_dir)  # Recall that dat_dir == "tests\\osm_data"
['birmingham',
 'greater-london',
 'gre_lon-ken-railways',
 'kent',
 'rutland',
 'west-midlands',
 'west-yorkshire']
```

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

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

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

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

This is the end of the *Quick start*.

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 *Sub-packages / modules*.

Python Module Index

p

Index

I

L

M

N

O

P

R

S

T

U

V

W