PyRCS Documentation

Release 0.2.13

Qian Fu

Mar 22, 2021

CONTENTS

1 Installation **1**

2 Quick start **3**
- 2.1 Get location codes . . . 3
  - 2.1.1 Get location codes for a given initial letter . . . 4
  - 2.1.2 Get all available location codes . . . 5
- 2.2 Get ELRs and mileages . . . 6
  - 2.2.1 Get ELR codes . . . 6
  - 2.2.2 Get mileage data for a given ELR . . . 7
- 2.3 Get railway stations data . . . 8

3 Subpackages and modules **11**
- 3.1 Subpackages . . . 11
  - 3.1.1 line_data . . . 11
    - elr_mileage . . . 11
    - elec . . . 19
    - loc_id . . . 29
    - lor_code . . . 39
    - line_name . . . 45
    - trk_diagr . . . 48
  - 3.1.2 other_assets . . . 52
    - sig_box . . . 52
    - tunnel . . . 58
    - viaduct . . . 62
    - station . . . 65
    - depot . . . 70
    - feature . . . 80
- 3.2 Modules . . . 90
  - 3.2.1 collector . . . 90
    - LineData . . . 91
    - OtherAssets . . . 93
  - 3.2.2 updater . . . 94
    - Local backup . . . 94
  - 3.2.3 utils . . . 95
    - Specification of resource homepage . . . 95
    - Data converters . . . 95

Data parsers 100
Retrieval of useful information 103
Rectification of location names 106
Data fixers 108
Miscellaneous utilities 109

4 License **113**

5 Use of data **115**

6 Acknowledgement **117**

Python Module Index **119**

Index **121**

CHAPTER

ONE

INSTALLATION

To install the latest release of PyRCS at PyPI via pip:

```
pip install --upgrade pyrcs
```

To install the more recent version hosted directly from GitHub repository:

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

To test if PyRCS is correctly installed, try importing the package via an interpreter shell:

```
>>> import pyrcs

>>> pyrcs.__version__  # Check the current release
```

```
The current release version is: 0.2.13
```

Note:

- If using a virtual environment, ensure that it is activated.
- To ensure you get the most recent version, it is always recommended to add `--upgrade` (or `-U`) to `pip install`.
- The package has not yet been tested with Python 2. For users who have installed both Python 2 and Python 3, it would be recommended to replace `pip` with `pip3`. But you are more than welcome to volunteer testing the package with Python 2 and any issues should be logged/reported onto the Issues page.
- For more general instructions, check the "Installing Packages".

CHAPTER

TWO

QUICK START

To demonstrate how PyRCS works, this part of the documentation provides a quick guide with examples of getting location codes, ELRs and railway stations data.

2.1 Get location codes

The location codes (including CRS, NLC, TIPLOC and STANOX) are categorised as line data. Import the class *LocationIdentifiers()* as follows:

```
>>> from pyrcs.line_data import LocationIdentifiers

>>> # Or simply
>>> # from pyrcs import LocationIdentifiers
```

Now we can create an instance for getting the location codes:

```
>>> lid = LocationIdentifiers()
```

Note: An alternative way of creating the instance is through the class *LineData()* (see below).

```
>>> from pyrcs import LineData

>>> ld = LineData()
>>> lid_ = ld.LocationIdentifiers
```

Note: The instance `ld` contains all classes under the category of line data. Here `lid_` is equivalent to `lid`.

2.1.1 Get location codes for a given initial letter

By using the method *LocationIdentifiers.collect_loc_codes_by_initial()*, we can get the location codes that start with a specific letter, say `'A'` or `'a'`:

```
>>> # The input is case-insensitive
>>> loc_codes_a = lid.collect_loc_codes_by_initial('A')

>>> type(loc_codes_a)
dict
>>> list(loc_codes_a.keys())
['A', 'Additional notes', 'Last updated date']
```

`loc_codes_a` is a dictionary (i.e. in dict type), with the following keys:

- `'A'`
- `'Additional notes'`
- `'Last updated date'`

Their corresponding values are

- `loc_codes_a['A']`: a pandas.DataFrame of the location codes that begin with 'A'. We may compare it with the table on the web page of Locations beginning with 'A';
- `loc_codes_a['Additional notes']`: some additional information on the web page (if available);
- `loc_codes_a['Last updated date']`: the date when the web page was last updated.

Below is a snapshot of the data of the location codes beginning with 'A':

```
>>> print(loc_codes_a['A'])
                                  Location CRS  ... STANME_Note STANOX_Note
0                                    Aachen      ...
1                       Abbeyhill Junction      ...
2                    Abbeyhill Signal E811      ...
3                Abbeyhill Turnback Sidings      ...
4    Abbey Level Crossing (Staffordshire)      ...
..                                     ...  ..  ...         ...         ...
715                      Ayr Signal PA335      ...
716                      Ayr Signal PA853      ...
717                      Ayr Signal PA858      ...
718                      Ayr Signal PA859      ...
719                 Ayr Wagon Repair Depot      ...
[720 rows x 12 columns]

>>> print("Last updated date: {}".format(loc_codes_a['Last updated date']))
Last updated date: 2021-01-02
```

2.1.2 Get all available location codes

To get all available location codes in this category, use the method *LocationIdentifiers.fetch_location_codes()*:

```
>>> loc_codes = lid.fetch_location_codes()

>>> type(loc_codes)
dict
>>> list(loc_codes.keys())
['Location codes', 'Other systems', 'Additional notes', 'Last updated date']
```

`loc_codes` is also a dictionary, of which the keys are as follows:

- `'Location codes'`
- `'Other systems'`
- `'Additional notes'`
- `'Latest update date'`

Their corresponding values are

- `loc_codes['Location codes']`: a pandas.DataFrame of all location codes (from 'A' to 'Z');
- `loc_codes['Other systems']`: a dictionary for other systems;
- `loc_codes['Additional notes']`: some additional information on the web page (if available);
- `loc_codes['Latest update date']`: the latest `'Last updated date'` among all initial letter-specific codes.

Below is a snapshot of a random sample of the location codes data:

```
>>> print(loc_codes['Location codes'].sample(10, random_state=1))
                                  Location  CRS  ... STANME_Note STANOX_Note
5369      Fiddlers Ferry Power Station Edison       ...
11311         Princes Risborough Signal ME178       ...
8551      Llandudno Junction Terminal Complex       ...
3856                Darlington North Junction       ...
1961   Bristol Barton Hill Wagon Repair Depot  XHL  ...
1604            Boat of Garten GB Railfreight       ...
1710                          Boundary Zone 2       ...
3822                         Dalston Junction  XJD  ...
11624                   Redbridge Signal E973       ...
1963         Bristol Bath Goods Signal BL1924       ...
[10 rows x 12 columns]
```

2.2 Get ELRs and mileages

To get ELRs (Engineer's Line References) and mileages, use the class *ELRMileages()*:

```
>>> from pyrcs.line_data import ELRMileages
>>> # Or simply
>>> # from pyrcs import ELRMileages

>>> em = ELRMileages()
```

2.2.1 Get ELR codes

To get ELR codes which start with `'A'`, use the method *ELRMileages.collect_elr_by_initial()*, which returns a dictionary:

```
>>> elrs_a = em.collect_elr_by_initial('A')

>>> type(elrs_a)
dict
>>> print(list(elrs_a.keys()))
['A', 'Last updated date']
```

The keys of `elrs_a` include:

- `'A'`
- `'Last updated date'`

Their corresponding values are

- `elrs_a['A']`: a pandas.DataFrame of ELRs that begin with 'A'. We may compare it with the table on the web page of ELRs beginning with 'A';
- `elrs_a['Last updated date']`: the date when the web page was last updated.

Below is a snapshot of the data of the ELR codes beginning with 'A':

```
>>> print(elrs_a['A'])
      ELR  ...          Notes
0     AAL  ...       Now NAJ3
1     AAM  ...  Formerly AML
2     AAV  ...
3     ABB  ...        Now AHB
4     ABB  ...
..    ...  ...            ...
186  AYR4  ...
187  AYR5  ...
188  AYR6  ...
189   AYS  ...
190   AYT  ...
[191 rows x 5 columns]
```

(continues on next page)

(continued from previous page)

```
>>> print("Last updated date: {}".format(elrs_a['Last updated date']))
Last updated date: 2020-10-27
```

To get all available ELR codes, use the method *ELRMileages.fetch_elr()*, which also returns a dictionary:

```
>>> elrs_dat = em.fetch_elr()

>>> type(elrs_dat)
dict
>>> list(elrs_dat.keys())
['ELRs', 'Last updated date']
```

The keys of `elrs_dat` include:

- `'ELRs'`
- `'Latest update date'`

Their corresponding values are

- `elrs_dat['ELRs']`: a pandas.DataFrame of all available ELRs (from 'A' to 'Z');
- `elrs_dat['Latest update date']`: the latest `'Last updated date'` among all initial letter-specific codes.

Below is a snapshot of a random sample of the ELR codes data:

```
>>> print(elrs_dat['ELRs'].sample(10, random_state=1))
       ELR  ...                        Notes
756    CFS  ...                Formerly CSW
589    BUI  ...
1230   DNB  ...
724   CDM1  ...
4399   WVH  ...  Possibly included in DAE2
636    BYN  ...
90    ALN1  ...
1128   DAE  ...
1123   CYM  ...                Formerly CMR
1373  EGS1  ...
[10 rows x 5 columns]
```

2.2.2 Get mileage data for a given ELR

To get detailed mileage data for a given ELR, for example, AAM, use the method *ELRMileages.fetch_mileage_file()*, which returns a dictionary as well:

```
>>> em_amm = em.fetch_mileage_file('AAM')

>>> type(em_amm)
dict
```

(continues on next page)

(continued from previous page)

```
>>> list(em_amm.keys())
['ELR', 'Line', 'Sub-Line', 'Mileage', 'Notes']
```

The keys of `em_amm` include:

- `'ELR'`
- `'Line'`
- `'Sub-Line'`
- `'Mileage'`
- `'Notes'`

Their corresponding values are

- `em_amm['ELR']`: the name of the given ELR (which in this example is 'AAM');
- `em_amm['Line']`: the associated line name;
- `em_amm['Sub-Line']`: the associated sub line name (if available);
- `em_amm['Mileage']`: a pandas.DataFrame of the mileage file data;
- `em_amm['Notes']`: additional information/notes (if any).

Below is a snapshot of the mileage data of AAM:

```
>>> print(em_amm['Mileage'])
   Mileage Mileage_Note  ... Link_2_ELR Link_2_Mile_Chain
0   0.0000               ...
1   0.0154               ...
2   0.0396               ...
3   1.1012               ...
4   1.1408               ...
5   5.0330               ...
6   7.0374               ...
7  11.1298               ...
8  13.0638               ...
[9 rows x 11 columns]
```

2.3 Get railway stations data

The railway station data (incl. the station name, ELR, mileage, status, owner, operator, degrees of longitude and latitude, and grid reference) is categorised into other assets in the source data.

```
>>> from pyrcs.other_assets import Stations
>>> # Or simply
>>> # from pyrcs import Stations

>>> stn = Stations()
```

Note: Alternatively, the instance `stn` can also be defined through *`OtherAssets()`* that contains all classes under the category of other assets (see below).

```
>>> from pyrcs import OtherAssets

>>> oa = OtherAssets()
>>> stn_ = oa.Stations
```

Note: `stn_` is equivalent to `stn`.

To get the data of railway stations whose names start with a specific letter, e.g. `'A'`, use the method *`Stations.collect_station_data_by_initial()`*:

```
>>> stn_data_a = stn.collect_station_data_by_initial('A')

>>> type(stn_data_a)
dict
>>> list(stn_data_a.keys())
['A', 'Last updated date']
```

The keys of `stn_data_a` include:

- `'A'`
- `'Last updated date'`

The corresponding values are

- `stn_data_a['A']`: a pandas.DataFrame of the data of railway stations whose names begin with 'A'. We may compare it with the table on the web page of Stations beginning with 'A';
- `stn_data_a['Last updated date']`: the date when the web page was last updated.

Below is a snapshot of the data of the railway stations beginning with 'A':

```
>>> print(stn_data_a['A'])
                    Station   ELR  ... Prev_Operator_6 Prev_Date_6
0                 Abbey Wood   NKL  ...            None        None
1                 Abbey Wood  XRS3  ...            None        None
2                      Aber   CAR  ...            None        None
3            Abercynon North   ABD  ...            None        None
4                              ABD  ...            None        None
..                       ...   ...  ...             ...         ...
133  Aylesbury Vale Parkway  MCJ2  ...            None        None
134               Aylesford  PWS2  ...            None        None
135                Aylesham   FDM  ...            None        None
136                     Ayr  AYR6  ...            None        None
137                     Ayr  STR1  ...            None        None
[138 rows x 23 columns]
```

(continues on next page)

(continued from previous page)

```
>>> print("Last updated date: {}".format(stn_data_a['Last updated date']))
Last updated date: 2020-11-14
```

To get available railway station data (from 'A' to 'Z') in this category, use the method *Stations.fetch_station_data()*

```
>>> stn_data = stn.fetch_station_data()

>>> type(stn_data)
dict
>>> list(stn_data.keys())
['Railway station data', 'Last updated date']
```

The keys of `stn_data` include:

- `'Railway station data'`
- `'Latest update date'`

Their corresponding values are

- `stn_data['Railway station data']`: a pandas.DataFrame of available railway station data (from 'A' to 'Z');
- `stn_data['Latest update date']`: the latest `'Last updated date'` among all initial letter-specific codes.

Below is a snapshot of a random sample of the railway station data:

```
>>> print(stn_data['Railway station data'].sample(10, random_state=1))
                      Station   ELR  ... Prev_Operator_6 Prev_Operator_Period_6
2670  Greenhithe for Bluewater   HDR  ...            None                    None
301           Bedford St Johns   BBM  ...            None                    None
47                   Fiskerton  NOB1  ...            None                    None
1493    Windsor & Eton Central   WIN  ...            None                    None
2282                  Sleights  MBW3  ...            None                    None
2457                Swaythling  BML1  ...            None                    None
418                 Bottesford  NOG1  ...            None                    None
2656             Great Bentley   COC  ...            None                    None
2472                   Yeoford   NDN  ...            None                    None
1134        London Kings Cross  ECM1  ...            None                    None
[10 rows x 32 columns]

>>> print("Last updated date: {}".format(stn_data['Last updated date']))
Last updated date: 2021-01-08
```

(The end of the quick start)

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

CHAPTER

THREE

SUBPACKAGES AND MODULES

3.1 Subpackages

line_data	A collection of modules for collecting line data.
other_assets	A collection of modules for collecting other assets.

3.1.1 line_data

A collection of modules for collecting line data. See also *pyrcs.collector.LineData*.

Submodules

elr_mileage	Collect Engineer's Line References (ELRs) codes.
elec	Collect codes of British railway overhead electrification installations.
loc_id	Collect CRS, NLC, TIPLOC and STANOX codes.
lor_code	Collect Line of Route (LOR/PRIDE) codes.
line_name	Collect British railway line names.
trk_diagr	Collect British railway track diagrams.

elr_mileage

Collect Engineer's Line References (ELRs) codes.

Class

ELRMileages([data_dir, update, verbose])	A class for collecting Engineer's Line References (ELRs) codes.

ELRMileages

class elr_mileage.**ELRMileages**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting Engineer's Line References (ELRs) codes.

Parameters

- **data_dir** (*str or None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to True

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type data
- **HomeURL** (*str*) – URL of the main homepage
- **SourceURL** (*str*) – URL of the data web page
- **LUDKey** (*str*) – key of the last updated date
- **LUD** (*str*) – last updated date
- **Catalogue** (*dict*) – catalogue of the data
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory

Example:

```
>>> from pyrcs.line_data import ELRMileages

>>> em = ELRMileages()

>>> print(em.Name)
ELRs and mileages

>>> print(em.SourceURL)
http://www.railwaycodes.org.uk/elrs/elr0.shtm
```

Methods

collect_elr_by_initial(initial[, update, ...])	Collect Engineer's Line References (ELRs) for the given initial letter from source web page.
collect_mileage_file(elr[, parsed, ...])	Collect mileage file for the given ELR from source web page.
fetch_elr([update, pickle_it, data_dir, verbose])	Fetch ELRs and mileages from local backup.
fetch_mileage_file(elr[, update, pickle_it, ...])	Fetch mileage file for the given ELR from local backup.
get_conn_mileages(start_elr, end_elr[, ...])	Get a connection point between two ELR-and-mileage pairs.
search_conn(start_elr, start_em, end_elr, end_em)	Search for connection between two ELR-and-mileage pairs.

ELRMileages.collect_elr_by_initial

ELRMileages.**collect_elr_by_initial**(*initial*, *update=False*, *verbose=False*)

Collect Engineer's Line References (ELRs) for the given initial letter from source web page.

Parameters

- **initial** (*str*) – initial letter of an ELR, e.g. `'a'`, `'z'`
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of ELRs whose names start with the given `initial` and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.line_data import ELRMileages

>>> em = ELRMileages()

>>> # elrs_a = em.collect_elr_by_initial(initial='a', update=True, verbose=True)
>>> elrs_a = em.collect_elr_by_initial(initial='a')

>>> type(elrs_a)
<class 'dict'>
>>> print(list(elrs_a.keys()))
['A', 'Last updated date']

>>> print(elrs_a['A'].head())
   ELR  ...         Notes
```

(continues on next page)

(continued from previous page)

```
0  AAL  ...      Now NAJ3
1  AAM  ...  Formerly AML
2  AAV  ...
3  ABB  ...       Now AHB
4  ABB  ...
[5 rows x 5 columns]
```

ELRMileages.collect_mileage_file

ELRMileages.**collect_mileage_file**(*elr*, *parsed=True*, *confirmation_required=True*, *pickle_it=False*, *verbose=False*)

Collect mileage file for the given ELR from source web page.

Parameters

- **elr** (*str*) – ELR, e.g. `'CJD'`, `'MLA'`, `'FED'`
- **parsed** (*bool*) – whether to parse the scraped mileage data
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns mileage file for the given `elr`

Return type dict

Note:

- In some cases, mileages are unknown hence left blank, e.g. ANI2, Orton Junction with ROB (~3.05)
- Mileages in parentheses are not on that ELR, but are included for reference, e.g. ANL, (8.67) NORTHOLT [London Underground]
- As with the main ELR list, mileages preceded by a tilde (~) are approximate.

Examples:

```
>>> from pyrcs.line_data import ELRMileages

>>> em = ELRMileages()

>>> mileage_dat = em.collect_mileage_file(elr='CJD')
To collect mileage file for "CJD"? [No]|Yes: yes
>>> type(mileage_dat)
```

(continues on next page)

(continued from previous page)

```
dict
>>> print(list(mileage_dat.keys()))
['ELR', 'Line', 'Sub-Line', 'Mileage', 'Notes']

>>> mileage_dat = em.collect_mileage_file(elr='GAM')
To collect mileage file of "GAM"? [No]|Yes: yes
>>> print(mileage_dat['Mileage'])
   Mileage Mileage_Note Miles_Chains  ... Link_1 Link_1_ELR Link_1_Mile_Chain
0   8.1518                     8.69  ...   None
1  10.0264                    10.12  ...   None
[2 rows x 8 columns]

>>> mileage_dat = em.collect_mileage_file(elr='SLD')
To collect mileage file of "SLD"? [No]|Yes: yes
>>> print(mileage_dat['Mileage'])
   Mileage Mileage_Note Miles_Chains  ... Link_1 Link_1_ELR Link_1_Mile_Chain
0  30.1694                    30.77  ...   None
1  32.1210                    32.55  ...   None
[2 rows x 8 columns]

>>> mileage_dat = em.collect_mileage_file(elr='ELR')
To collect mileage file of "ELR"? [No]|Yes: yes
>>> print(mileage_dat['Mileage'].head())
    Mileage Mileage_Note  ... Link_1_ELR Link_1_Mile_Chain
0  122.0044               ...       GRS3
1  122.0682               ...                        0.00
2  122.0726               ...        SPI             0.00
3  122.0836               ...
4  124.0792               ...
[5 rows x 8 columns]
```

ELRMileages.fetch_elr

ELRMileages.**fetch_elr**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch ELRs and mileages from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of all available ELRs and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.line_data import ELRMileages

>>> em = ELRMileages()

>>> elrs_dat = em.fetch_elr()

>>> type(elrs_dat)
<class 'dict'>
>>> print(list(elrs_dat.keys()))
['ELRs', 'Last updated date']

>>> print(elrs_dat['ELRs'])
   ELR  ...         Notes
0  AAL  ...      Now NAJ3
1  AAM  ...  Formerly AML
2  AAV  ...
3  ABB  ...       Now AHB
4  ABB  ...
[5 rows x 5 columns]
```

ELRMileages.fetch_mileage_file

ELRMileages.**fetch_mileage_file**(*elr*, *update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch mileage file for the given ELR from local backup.

Parameters

- **elr** (*str*) – elr: ELR, e.g. `'CJD'`, `'MLA'`, `'FED'`
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns mileage file (codes), line name and, if any, additional information/notes

Return type dict

Example:

```
>>> from pyrcs.line_data import ELRMileages

>>> em = ELRMileages()

>>> mileage_dat = em.fetch_mileage_file('MLA')
```

(continues on next page)

(continued from previous page)

```
>>> type(mileage_dat)
dict
>>> list(mileage_dat.keys())
['ELR', 'Line', 'Sub-Line', 'Mileage', 'Notes']

>>> type(mileage_dat['Mileage'])
dict
>>> list(mileage_dat['Mileage'].keys())
['Current measure', 'Original measure']
>>> print(mileage_dat['Mileage']['Current measure'])
  Mileage Mileage_Note Miles_Chains  ...       Link_1 Link_1_ELR Link_1_Mile_
↪Chain
0  0.0000                     0.00  ...  MRL2 (4.44)       MRL2             4.
↪44
1  0.0572                     0.26  ...         None
2  0.1540                     0.70  ...         None
3  0.1606                     0.73  ...         None
[4 rows x 8 columns]
```

ELRMileages.get_conn_mileages

ELRMileages.**get_conn_mileages**(*start_elr*, *end_elr*, *update=False*, *pickle_mileage_file=False*, *data_dir=None*, *verbose=False*)

Get a connection point between two ELR-and-mileage pairs.

Namely, find the end and start mileages for the start and end ELRs, respectively.

Note: This function may not be able find the connection for every pair of ELRs. See the *Example 2* below.

Parameters

- **start_elr** (*str*) – start ELR
- **end_elr** (*str*) – end ELR
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_mileage_file** (*bool*) – whether to replace the current mileage file, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns connection ELR and mileages between the given `start_elr` and `end_elr`

Return type tuple

Example 1:

```
>>> from pyrcs.line_data import ELRMileages

>>> em = ELRMileages()

>>> conn = em.get_conn_mileages('NAY', 'LTN2')
>>> (s_dest_mlg, c_elr, c_orig_mlg, c_dest_mlg, e_orig_mlg) = conn

>>> print(s_dest_mlg)
5.1606
>>> print(c_elr)
NOL
>>> print(c_orig_mlg)
5.1606
>>> print(c_dest_mlg)
0.0638
>>> print(e_orig_mlg)
123.1320
```

Example 2:

```
>>> from pyrcs.line_data import ELRMileages

>>> em = ELRMileages()

>>> conn = em.get_conn_mileages('MAC3', 'DBP1')
>>> print(conn)
('', '', '', '', '')
```

ELRMileages.search_conn

static ELRMileages.**search_conn**(*start_elr*, *start_em*, *end_elr*, *end_em*)

Search for connection between two ELR-and-mileage pairs.

Parameters

- **start_elr** (*str*) – start ELR
- **start_em** (*pandas.DataFrame*) – mileage file of the start ELR
- **end_elr** (*str*) – end ELR
- **end_em** (*pandas.DataFrame*) – mileage file of the end ELR

Returns connection, in the form (<end mileage of the start ELR>, <start mileage of the end ELR>)

Return type tuple

Example:

```
>>> from pyrcs.line_data import ELRMileages

>>> em = ELRMileages()

>>> s_elr = 'AAM'
>>> s_m_file = em.collect_mileage_file(s_elr, confirmation_required=False)
>>> s_m_data = s_m_file['Mileage']

>>> e_elr = 'ANZ'
>>> e_m_file = em.collect_mileage_file(e_elr, confirmation_required=False)
>>> e_m_data = e_m_file['Mileage']

>>> s_dest_mileage, e_orig_mileage = em.search_conn(s_elr, s_m_data, e_elr, e_m_
↪data)

>>> print(s_dest_mileage)
0.0396
>>> print(e_orig_mileage)
84.1364
```

elec

Collect codes of British railway overhead electrification installations.

Class

Electrification([data_dir, update, verbose])	A class for collecting section codes for OLE installations.

Electrification

class elec.**Electrification**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting section codes for OLE installations.

Parameters

- **data_dir** (*str or None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to True

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type data

- **HomeURL** (*str*) – URL of the main homepage
- **SourceURL** (*str*) – URL of the data web page
- **LUDKey** (*str*) – key of the last updated date
- **LUD** (*str*) – last updated date
- **Catalogue** (*dict*) – catalogue of the data
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory
- **NationalNetworkKey** (*str*) – key of the dict-type data of national network
- **NationalNetworkPickle** (*str*) – name of the pickle file of national network data
- **IndependentLinesKey** (*str*) – key of the dict-type data of independent lines
- **IndependentLinesPickle** (*str*) – name of the pickle file of independent lines data
- **OhnsKey** (*str*) – key of the dict-type data of OHNS
- **OhnsPickle** (*str*) – name of the pickle file of OHNS data
- **TariffZonesKey** (*str*) – key of the dict-type data of tariff zones
- **TariffZonesPickle** (*str*) – name of the pickle file of tariff zones data

Example:

```
>>> from pyrcs.line_data import Electrification

>>> elec = Electrification()

>>> print(elec.Name)
Electrification masts and related features

>>> print(elec.SourceURL)
http://www.railwaycodes.org.uk/electrification/mast_prefix0.shtm
```

Methods

collect_etz_codes([confirmation_requi ...])	Collect OLE section codes for national network energy tariff zones from source web page.
collect_indep_lines_codes([...])	Collect OLE section codes for independent lines from source web page.
collect_national_network_codes([...]	Collect OLE section codes for national network from source web page.
collect_ohns_codes([confirmation_requ ...])	Collect codes for overhead line electrification neutral sections (OHNS) from source web page.

continues on next page

Table 6 – continued from previous page	
fetch_elec_codes([update, pickle_it, ...])	Fetch OLE section codes in electrification catalogue.
fetch_etz_codes([update, pickle_it, ...])	Fetch OLE section codes for national network energy tariff zones from source web page.
fetch_indep_lines_codes([update, pickle_it, ...])	Fetch OLE section codes for independent lines from local backup.
fetch_national_network_codes([updat ...])	Fetch OLE section codes for national network from local backup.
fetch_ohns_codes([update, pickle_it, ...])	Fetch codes for overhead line electrification neutral sections (OHNS) from local backup.
get_indep_line_names([verbose])	Get names of independent lines.

Electrification.collect_etz_codes

Electrification.**collect_etz_codes**(*confirmation_required=True*, *verbose=False*)

Collect OLE section codes for national network energy tariff zones from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to require users to confirm and proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Returns OLE section codes for national network energy tariff zones

Return type dict or None

Example:

```
>>> from pyrcs.line_data import Electrification

>>> elec = Electrification()

>>> etz_ole_dat = elec.collect_etz_codes(confirmation_required=False)

>>> type(etz_ole_dat)
dict
>>> list(etz_ole_dat.keys())
['National network energy tariff zones', 'Last updated date']

>>> type(etz_ole_dat['National network energy tariff zones'])
dict
```

(continues on next page)

(continued from previous page)

```
>>> list(etz_ole_dat['National network energy tariff zones'].keys())
['Railtrack', 'Notes', 'Network Rail']
```

Electrification.collect_indep_lines_codes

Electrification.**collect_indep_lines_codes**(*confirmation_required=True*, *verbose=False*)

Collect OLE section codes for independent lines from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to require users to confirm and proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Returns OLE section codes for independent lines

Return type dict or None

Example:

```
>>> from pyrcs.line_data import Electrification

>>> elec = Electrification()

>>> il_ole_dat = elec.collect_indep_lines_codes(confirmation_required=False)

>>> type(il_ole_dat)
dict
>>> list(il_ole_dat.keys())
['Independent lines', 'Last updated date']

>>> type(il_ole_dat['Independent lines'])
dict
>>> list(il_ole_dat['Independent lines'].keys())[-5:]
['Seaton Tramway',
 'Sheffield Supertram',
 'Snaefell Mountain Railway',
 'Summerlee, Museum of Scottish Industrial Life Tramway',
 'Tyne & Wear Metro']
```

Electrification.collect_national_network_codes

Electrification.**collect_national_network_codes**(*confirmation_required=True*, *verbose=False*)

Collect OLE section codes for national network from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to require users to confirm and proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Returns OLE section codes for National network

Return type dict or None

Example:

```
>>> from pyrcs.line_data import Electrification

>>> elec = Electrification()

>>> nn_dat = elec.collect_national_network_codes(confirmation_required=False)

>>> type(nn_dat)
dict
>>> list(nn_dat.keys())
['National network', 'Last updated date']

>>> type(nn_dat['National network'])
dict
>>> list(nn_dat['National network'].keys())
['Traditional numbering system distance and sequence',
 'New numbering system km and decimal',
 'Codes not certain confirmation is welcome',
 'Suspicious data',
 'An odd one to complete the record',
 'LBSC/Southern Railway overhead system',
 'Codes not known']
```

Electrification.collect_ohns_codes

Electrification.**collect_ohns_codes**(*confirmation_required=True*, *verbose=False*)

Collect codes for overhead line electrification neutral sections (OHNS) from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to require users to confirm and proceed, defaults to True

- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns OHNS codes

Return type dict or None

Example:

```
>>> from pyrcs.line_data import Electrification

>>> elec = Electrification()

>>> ohns_dat = elec.collect_ohns_codes(confirmation_required=False)

>>> type(ohns_dat)
dict
>>> list(ohns_dat.keys())
['National network neutral sections', 'Last updated date']

>>> print(ohns_dat['National network neutral sections'].head())
    ELR          OHNS Name  Mileage    Tracks Dates
0  ARG1         Rutherglen   0m 3ch
1  ARG2   Finnieston East  4m 23ch      Down
2  ARG2   Finnieston West  4m 57ch        Up
3  AYR1  Shields Junction  0m 68ch    Up Ayr
4  AYR1  Shields Junction  0m 69ch  Down Ayr
```

Electrification.fetch_elec_codes

Electrification.**fetch_elec_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch OLE section codes in electrification catalogue.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns section codes for overhead line electrification (OLE) installations

Return type dict

Example:

```
>>> from pyrcs.line_data import Electrification

>>> elec = Electrification()

>>> # electrification_codes = elec.fetch_elec_codes(update=True, verbose=True)
>>> electrification_codes = elec.fetch_elec_codes()

>>> type(electrification_codes)
dict
>>> list(electrification_codes.keys())
['Electrification', 'Last updated date']

>>> type(electrification_codes['Electrification'])
dict
>>> list(electrification_codes['Electrification'].keys())
['National network energy tariff zones',
 'Independent lines',
 'National network',
 'National network neutral sections']
```

Electrification.fetch_etz_codes

Electrification.**fetch_etz_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch OLE section codes for national network energy tariff zones from source web page.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns OLE section codes for national network energy tariff zones

Return type dict

Example:

```
>>> from pyrcs.line_data import Electrification

>>> elec = Electrification()

>>> # etz_ole_dat = elec.fetch_etz_codes(update=True, verbose=True)
>>> etz_ole_dat = elec.fetch_etz_codes()

>>> type(etz_ole_dat)
```

(continues on next page)

(continued from previous page)

```
dict
>>> list(etz_ole_dat.keys())
['National network energy tariff zones', 'Last updated date']

>>> type(etz_ole_dat['National network energy tariff zones'])
dict
>>> list(etz_ole_dat['National network energy tariff zones'].keys())
['Railtrack', 'Notes', 'Network Rail']
```

Electrification.fetch_indep_lines_codes

Electrification.**fetch_indep_lines_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch OLE section codes for independent lines from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns OLE section codes for independent lines

Return type dict

Example:

```
>>> from pyrcs.line_data import Electrification

>>> elec = Electrification()

>>> # il_ole_dat = elec.fetch_indep_lines_codes(update=True, verbose=True)
>>> il_ole_dat = elec.fetch_indep_lines_codes()

>>> type(il_ole_dat)
dict
>>> list(il_ole_dat.keys())
['Independent lines', 'Last updated date']

>>> type(il_ole_dat['Independent lines'])
dict
>>> list(il_ole_dat['Independent lines'].keys())[-5:]
['Seaton Tramway',
 'Sheffield Supertram',
 'Snaefell Mountain Railway',
```

(continues on next page)

(continued from previous page)

```
 'Summerlee, Museum of Scottish Industrial Life Tramway',
 'Tyne & Wear Metro']
```

Electrification.fetch_national_network_codes

Electrification.**fetch_national_network_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch OLE section codes for national network from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns OLE section codes for National network

Return type dict or None

Example:

```
>>> from pyrcs.line_data import Electrification

>>> elec = Electrification()

>>> # nn_ole_dat = elec.fetch_national_network_codes(update=True, verbose=True)
>>> nn_dat = elec.fetch_national_network_codes()

>>> type(nn_dat)
dict
>>> list(nn_dat.keys())
['National network', 'Last updated date']

>>> type(nn_dat['National network'])
dict
>>> list(nn_dat['National network'].keys())
['Traditional numbering system distance and sequence',
 'New numbering system km and decimal',
 'Codes not certain confirmation is welcome',
 'Suspicious data',
 'An odd one to complete the record',
 'LBSC/Southern Railway overhead system',
 'Codes not known']
```

Electrification.fetch_ohns_codes

Electrification.**fetch_ohns_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch codes for overhead line electrification neutral sections (OHNS) from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns OHNS codes

Return type dict

Example:

```
>>> from pyrcs.line_data import Electrification

>>> elec = Electrification()

>>> # ohns_dat = elec.fetch_ohns_codes(update=True, verbose=True)
>>> ohns_dat = elec.fetch_ohns_codes()

>>> type(ohns_dat)
dict
>>> list(ohns_dat.keys())
['National network neutral sections', 'Last updated date']

>>> print(ohns_dat['National network neutral sections'].head())
    ELR         OHNS Name  Mileage    Tracks Dates
0  ARG1        Rutherglen   0m 3ch
1  ARG2   Finnieston East  4m 23ch      Down
2  ARG2   Finnieston West  4m 57ch        Up
3  AYR1  Shields Junction  0m 68ch    Up Ayr
4  AYR1  Shields Junction  0m 69ch  Down Ayr
```

Electrification.get_indep_line_names

Electrification.**get_indep_line_names**(*verbose=False*)

Get names of independent lines.

Parameters **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns a list of independent line names

Return type list

Example:

```
>>> from pyrcs.line_data import Electrification

>>> elec = Electrification()

>>> l_names = elec.get_indep_line_names()

>>> l_names[:5]
['Beamish Tramway',
 'Birkenhead Tramway',
 'Black Country Living Museum',
 'Blackpool Tramway',
 'Brighton and Rottingdean Seashore Electric Railway']
```

loc_id

Collect CRS, NLC, TIPLOC and STANOX codes.

Class

LocationIdentifiers([data_dir, update, verbose])	A class for collecting location identifiers (including other systems station).

LocationIdentifiers

class loc_id.**LocationIdentifiers**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting location identifiers (including other systems station).

Parameters

- **data_dir** (*str or None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to True

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type data
- **HomeURL** (*str*) – URL of the main homepage
- **SourceURL** (*str*) – URL of the data web page
- **LUDKey** (*str*) – key of the last updated date
- **LUD** (*str*) – last updated date
- **Catalogue** (*dict*) – catalogue of the data
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory
- **OtherSystemsKey** (*str*) – key of the dict-type data of other systems
- **OtherSystemsPickle** (*str*) – name of the pickle file of other systems data
- **AddNotesKey** (*str*) – key of the dict-type data of additional notes
- **MscENKey** (*str*) – key of the dict-type data of multiple station codes explanatory note
- **MscENPickle** (*str*) – name of the pickle file of multiple station codes explanatory note

Example:

```
>>> from pyrcs.line_data import LocationIdentifiers

>>> lid = LocationIdentifiers()

>>> print(lid.Name)
CRS, NLC, TIPLOC and STANOX codes

>>> print(lid.SourceURL)
http://www.railwaycodes.org.uk/crs/crs0.shtm
```

Methods

amendment_to_loc_names()	Create a replacement dictionary for location name amendments.
collect_explanatory_note([...])	Collect note about CRS code from source web page.
collect_loc_codes_by_initial(initial[...])	Collect CRS, NLC, TIPLOC, STANME and STANOX codes for a given `initial` letter.
collect_other_systems_codes([...])	Collect data of other systems' codes from source web page.

continues on next page

Table 8 – continued from previous page	
fetch_explanatory_note([update, pickle_it, ...])	Fetch multiple station codes explanatory note from local backup.
fetch_location_codes([update, pickle_it, ...])	Fetch CRS, NLC, TIPLOC, STANME and STANOX codes from local backup.
fetch_other_systems_codes([update, ...])	Fetch data of other systems' codes from local backup.
make_loc_id_dict(keys[, initials, ...])	Make a dict/dataframe for location code data for the given `keys`.
parse_note_page(note_url[, parser, verbose])	Parse addition note page.

LocationIdentifiers.amendment_to_loc_names

static LocationIdentifiers.**amendment_to_loc_names**()

Create a replacement dictionary for location name amendments.

Returns dictionary of regular-expression amendments to location names

Return type dict

Example:

```
>>> from pyrcs.line_data import LocationIdentifiers

>>> lid = LocationIdentifiers()

>>> loc_name_amendment_dict = lid.amendment_to_loc_names()

>>> print(loc_name_amendment_dict.keys())
dict_keys(['Location'])
```

LocationIdentifiers.collect_explanatory_note

LocationIdentifiers.**collect_explanatory_note**(*confirmation_required=True*, *verbose=False*)

Collect note about CRS code from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of multiple station codes explanatory note

Return type dict or None

Example:

```
>>> from pyrcs.line_data import LocationIdentifiers

>>> lid = LocationIdentifiers()

>>> exp_note = lid.collect_explanatory_note(confirmation_required=False)

>>> type(exp_note)
dict
>>> list(exp_note.keys())
['Multiple station codes explanatory note', 'Notes', 'Last updated date']

>>> print(exp_note['Multiple station codes explanatory note'].head())
                Location  CRS CRS_alt1 CRS_alt2
0    Glasgow Queen Street  GLQ      GQL
1         Glasgow Central  GLC      GCL
2                Heworth  HEW      HEZ
3    Highbury & Islington  HHY      HII      XHZ
4  Lichfield Trent Valley  LTV      LIF
```

LocationIdentifiers.collect_loc_codes_by_initial

LocationIdentifiers.**collect_loc_codes_by_initial**(*initial*, *update=False*, *verbose=False*)

Collect CRS, NLC, TIPLOC, STANME and STANOX codes for a given `initial` letter.

Parameters

- **initial** (*str*) – initial letter of station/junction name or certain word for specifying URL
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of location codes for the given `initial` letter; and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.line_data import LocationIdentifiers

>>> lid = LocationIdentifiers()

>>> location_codes_a = lid.collect_loc_codes_by_initial(initial='a')

>>> type(location_codes_a)
dict
```

(continues on next page)

(continued from previous page)

```
>>> list(location_codes_a.keys())
['A', 'Additional notes', 'Last updated date']

>>> print(location_codes_a['A'].head())
                              Location CRS  ... STANME_Note STANOX_Note
0                                Aachen      ...
1                   Abbeyhill Junction      ...
2                Abbeyhill Signal E811      ...
3           Abbeyhill Turnback Sidings      ...
4  Abbey Level Crossing (Staffordshire)      ...
[5 rows x 12 columns]
```

LocationIdentifiers.collect_other_systems_codes

LocationIdentifiers.**collect_other_systems_codes**(*confirmation_required=True*, *verbose=False*)

Collect data of other systems' codes from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to require users to confirm and proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns codes of other systems

Return type dict or None

Example:

```
>>> from pyrcs.line_data import LocationIdentifiers

>>> lid = LocationIdentifiers()

>>> os_codes = lid.collect_other_systems_codes(confirmation_required=False)

>>> type(os_codes)
dict
>>> list(os_codes.keys())
['Other systems', 'Last updated date']

>>> type(os_codes['Other systems'])
dict
>>> list(os_codes['Other systems'].keys())
['Córas Iompair Éireann (Republic of Ireland)',
 'Crossrail',
 'Croydon Tramlink',
 'Docklands Light Railway',
 'Manchester Metrolink',
```

(continues on next page)

(continued from previous page)

```
 'Translink (Northern Ireland)',
 'Tyne & Wear Metro']
```

LocationIdentifiers.fetch_explanatory_note

LocationIdentifiers.**fetch_explanatory_note**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch multiple station codes explanatory note from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of multiple station codes explanatory note

Return type dict

Example:

```
>>> from pyrcs.line_data import LocationIdentifiers

>>> lid = LocationIdentifiers()

>>> # exp_note = lid.fetch_explanatory_note(update=True, verbose=True)
>>> exp_note = lid.fetch_explanatory_note()

>>> type(exp_note)
dict
>>> list(exp_note.keys())
['Multiple station codes explanatory note', 'Notes', 'Last updated date']

>>> print(exp_note['Multiple station codes explanatory note'].head())
                Location  CRS CRS_alt1 CRS_alt2
0    Glasgow Queen Street  GLQ      GQL
1         Glasgow Central  GLC      GCL
2                Heworth  HEW      HEZ
3    Highbury & Islington  HHY      HII      XHZ
4  Lichfield Trent Valley  LTV      LIF
```

LocationIdentifiers.fetch_location_codes

LocationIdentifiers.**fetch_location_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch CRS, NLC, TIPLOC, STANME and STANOX codes from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of location codes and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.line_data import LocationIdentifiers

>>> lid = LocationIdentifiers()

>>> # loc_codes = lid.fetch_location_codes(update=True, verbose=True)
>>> loc_codes = lid.fetch_location_codes()

>>> type(loc_codes)
dict
>>> list(loc_codes.keys())
['Location codes', 'Other systems', 'Additional notes', 'Last updated date']

>>> print(loc_codes['Location codes'].head())
                              Location CRS  ... STANME_Note STANOX_Note
0                                   Aachen      ...
1                       Abbeyhill Junction      ...
2                    Abbeyhill Signal E811      ...
3              Abbeyhill Turnback Sidings      ...
4  Abbey Level Crossing (Staffordshire)      ...
[5 rows x 12 columns]
```

LocationIdentifiers.fetch_other_systems_codes

LocationIdentifiers.**fetch_other_systems_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch data of other systems' codes from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns codes of other systems

Return type dict

Example:

```
>>> from pyrcs.line_data import LocationIdentifiers

>>> lid = LocationIdentifiers()

>>> # os_codes = lid.fetch_other_systems_codes(update=True, verbose=True)
>>> os_codes = lid.fetch_other_systems_codes()

>>> type(os_codes)
dict
>>> list(os_codes.keys())
['Other systems', 'Last updated date']

>>> type(os_codes['Other systems'])
dict
>>> list(os_codes['Other systems'].keys())
['Córas Iompair Éireann (Republic of Ireland)',
 'Crossrail',
 'Croydon Tramlink',
 'Docklands Light Railway',
 'Manchester Metrolink',
 'Translink (Northern Ireland)',
 'Tyne & Wear Metro']
```

LocationIdentifiers.make_loc_id_dict

LocationIdentifiers.**make_loc_id_dict**(*keys*, *initials=None*, *drop_duplicates=False*, *as_dict=False*, *main_key=None*, *save_it=False*, *data_dir=None*, *update=False*, *verbose=False*)

Make a dict/dataframe for location code data for the given `keys`.

Parameters

- **keys** (*str or list*) – one or a sublist of ['CRS', 'NLC', 'TIPLOC', 'STANOX', 'STANME']
- **initials** (*str or list or None*) – one or a sequence of initials for which the location codes are used, defaults to `None`
- **drop_duplicates** (*bool*) – whether to drop duplicates, defaults to `False`
- **as_dict** (*bool*) – whether to return a dictionary, defaults to `False`
- **main_key** (*str or None*) – key of the returned dictionary if `as_dict` is `True`, defaults to `None`
- **save_it** (*bool*) – whether to save the location codes dictionary, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns dictionary or a data frame for location code data for the given `keys`

Return type dict or pandas.DataFrame or None

Examples:

```
>>> from pyrcs.line_data import LocationIdentifiers

>>> lid = LocationIdentifiers()

>>> key = 'STANOX'
>>> stanox_dictionary = lid.make_loc_id_dict(key)

>>> print(stanox_dictionary.head())
                         Location
STANOX
00005                       Aachen
04309          Abbeyhill Junction
04311        Abbeyhill Signal E811
04308   Abbeyhill Turnback Sidings
88601                  Abbey Wood

>>> ks = ['STANOX', 'TIPLOC']
```

(continues on next page)

(continued from previous page)

```
>>> ini = 'a'

>>> stanox_dictionary = lid.make_loc_id_dict(ks, ini)

>>> print(stanox_dictionary.head())
                                Location
STANOX TIPLOC
00005  AACHEN                     Aachen
04309  ABHLJN           Abbeyhill Junction
04311  ABHL811       Abbeyhill Signal E811
04308  ABHLTB   Abbeyhill Turnback Sidings
88601  ABWD                    Abbey Wood

>>> ks = ['STANOX', 'TIPLOC']
>>> ini = 'b'

>>> stanox_dictionary = lid.make_loc_id_dict(ks, ini, as_dict=True, main_key=
↪'Data')

>>> type(stanox_dictionary)
dict
>>> list(stanox_dictionary['Data'].keys())[:5]
[('55115', ''),
 ('23490', 'BABWTHL'),
 ('38306', 'BACHE'),
 ('66021', 'BADESCL'),
 ('81003', 'BADMTN')]
```

LocationIdentifiers.parse_note_page

static LocationIdentifiers.**parse_note_page**(*note_url*, *parser='lxml'*, *verbose=False*)

Parse addition note page.

Parameters

- **note_url** (*str*) – URL link of the target web page
- **parser** (*str*) – the parser to use for bs4.BeautifulSoup, defaults to `'lxml'`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns parsed texts

Return type list

Example:

```
>>> from pyrcs.line_data import LocationIdentifiers

>>> lid = LocationIdentifiers()
```

(continues on next page)

(continued from previous page)

```
>>> url = lid.Catalogue[lid.MscENKey]
>>> parsed_note_dat = lid.parse_note_page(url, parser='lxml')

>>> print(parsed_note_dat[3].head())
                Location  CRS CRS_alt1 CRS_alt2
0    Glasgow Queen Street  GLQ      GQL
1         Glasgow Central  GLC      GCL
2                Heworth  HEW      HEZ
3   Highbury & Islington  HHY      HII      XHZ
4  Lichfield Trent Valley  LTV      LIF
```

lor_code

Collect Line of Route (LOR/PRIDE) codes.

Class

LOR([data_dir, update, verbose])	A class for collecting Line of Route (LOR/PRIDE) codes.

LOR

class lor_code.**LOR**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting Line of Route (LOR/PRIDE) codes.

- PRIDE: Possession Resource Information Database
- LOR: Line Of Route

Parameters

- **data_dir** (*str or None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to True

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type LOR data
- **PKey** (*str*) – key of the dict-type prefixes
- **ELCKey** (*str*) – key of the dict-type ELR/LOR converter data
- **HomeURL** (*str*) – URL of the main homepage

- **SourceURL** (*str*) – URL of the data web page
- **LUDKey** (*str*) – key of the last updated date
- **LUD** (*str*) – last updated date
- **Catalogue** (*dict*) – catalogue of the data
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory

Example:

```
>>> from pyrcs.line_data import LOR

>>> lor = LOR()

>>> print(lor.Name)
Possession Resource Information Database (PRIDE)/Line Of Route (LOR) codes

>>> print(lor.SourceURL)
http://www.railwaycodes.org.uk/pride/pride0.shtm
```

Methods

collect_elr_lor_converter([...])	Collect ELR/LOR converter from source web page.
collect_lor_codes_by_prefix(prefix[, ...])	Collect PRIDE/LOR codes by a given prefix.
fetch_elr_lor_converter([update, pickle_it, ...])	Fetch ELR/LOR converter from local backup.
fetch_lor_codes([update, pickle_it, ...])	Fetch PRIDE/LOR codes from local backup.
get_keys_to_prefixes([prefixes_only, ...])	Get key to PRIDE/LOR code prefixes.
get_lor_page_urls([update, verbose])	Get URLs to PRIDE/LOR codes with different prefixes.

LOR.collect_elr_lor_converter

LOR.**collect_elr_lor_converter**(*confirmation_required=True*, *verbose=False*)

Collect ELR/LOR converter from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to require users to confirm and proceed, defaults to `True`
- **verbose** (*bool*) – whether to print relevant information in console as the

function runs, defaults to `False`

Returns data of ELR/LOR converter

Return type dict or None

Example:

```
>>> from pyrcs.line_data import LOR

>>> lor = LOR()

>>> elr_lor_conv = lor.collect_elr_lor_converter()
To collect data of ELR/LOR converter? [No]|Yes: yes

>>> type(elr_lor_conv)
dict
>>> list(elr_lor_conv.keys())
['ELR/LOR converter', 'Last updated date']

>>> print(elr_lor_conv['ELR/LOR converter'].head())
    ELR  ...                                            LOR_URL
0   AAV  ...  http://www.railwaycodes.org.uk/pride/pridesw.s...
1   ABD  ...  http://www.railwaycodes.org.uk/pride/pridegw.s...
2   ABE  ...  http://www.railwaycodes.org.uk/pride/prideln.s...
3  ABE1  ...  http://www.railwaycodes.org.uk/pride/prideln.s...
4  ABE2  ...  http://www.railwaycodes.org.uk/pride/prideln.s...
[5 rows x 6 columns]
```

LOR.collect_lor_codes_by_prefix

LOR.**collect_lor_codes_by_prefix**(*prefix*, *update=False*, *verbose=False*)

Collect PRIDE/LOR codes by a given prefix.

Parameters

- **prefix** (*str*) – prefix of LOR codes
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns LOR codes for the given `prefix`

Return type dict or None

Examples:

```
>>> from pyrcs.line_data import LOR

>>> lor = LOR()
```

(continues on next page)

(continued from previous page)

```
>>> lor_codes_cy = lor.collect_lor_codes_by_prefix(prefix='CY')

>>> type(lor_codes_cy)
dict
>>> list(lor_codes_cy.keys())
['CY', 'Notes', 'Last updated date']
>>> type(lor_codes_cy['CY'])
pandas.core.frame.DataFrame

>>> lor_codes_nw = lor.collect_lor_codes_by_prefix(prefix='NW')
>>> list(lor_codes_nw.keys())
['NW/NZ', 'Notes', 'Last updated date']

>>> lor_codes_ea = lor.collect_lor_codes_by_prefix(prefix='EA')
>>> ea_dat = lor_codes_ea['EA']
>>> type(ea_dat)
dict
>>> list(ea_dat.keys())
['Current system', 'Original system']
>>> print(ea_dat['Current system']['EA'].head())
     Code  ... Line Name Note
0  EA1000  ...           None
1  EA1010  ...           None
2  EA1011  ...           None
3  EA1012  ...           None
4  EA1013  ...           None
[5 rows x 5 columns]
```

LOR.fetch_elr_lor_converter

LOR.**fetch_elr_lor_converter**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch ELR/LOR converter from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of ELR/LOR converter

Return type dict

Example:

```
>>> from pyrcs.line_data import LOR

>>> lor = LOR()

>>> # elr_lor_conv = lor.fetch_elr_lor_converter(update=True, verbose=True)
>>> elr_lor_conv = lor.fetch_elr_lor_converter()

>>> type(elr_lor_conv)
dict
>>> list(elr_lor_conv.keys())
['ELR/LOR converter', 'Last updated date']

>>> print(elr_lor_conv['ELR/LOR converter'].head())
    ELR  ...                                            LOR_URL
0   AAV  ...  http://www.railwaycodes.org.uk/pride/pridesw.s...
1   ABD  ...  http://www.railwaycodes.org.uk/pride/pridegw.s...
2   ABE  ...  http://www.railwaycodes.org.uk/pride/prideln.s...
3  ABE1  ...  http://www.railwaycodes.org.uk/pride/prideln.s...
4  ABE2  ...  http://www.railwaycodes.org.uk/pride/prideln.s...
[5 rows x 6 columns]
```

LOR.fetch_lor_codes

LOR.**fetch_lor_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch PRIDE/LOR codes from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns LOR codes

Return type dict

Example:

```
>>> from pyrcs.line_data import LOR

>>> lor = LOR()

>>> # lor_codes_dat = lor.fetch_lor_codes(update=True, verbose=True)
>>> lor_codes_dat = lor.fetch_lor_codes()

>>> type(lor_codes_dat)
```

(continues on next page)

(continued from previous page)

```
dict
>>> type(lor_codes_dat['LOR'])
dict
>>> list(lor_codes_dat['LOR'].keys())
['CY', 'EA', 'GW', 'LN', 'MD', 'NW/NZ', 'SC', 'SO', 'SW', 'XR']

>>> type(lor_codes_dat['LOR']['CY'])
dict
```

LOR.get_keys_to_prefixes

LOR.**get_keys_to_prefixes**(*prefixes_only=True*, *update=False*, *verbose=False*)

Get key to PRIDE/LOR code prefixes.

Parameters

- **prefixes_only** (*bool*) – whether to get only prefixes, defaults to `True`
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns keys to LOR code prefixes

Return type list or dict

Examples:

```
>>> from pyrcs.line_data import LOR

>>> lor = LOR()

>>> # keys_to_pfx = lor.get_keys_to_prefixes(update=True, verbose=True)
>>> keys_to_pfx = lor.get_keys_to_prefixes()

>>> print(keys_to_pfx)
['CY', 'EA', 'GW', 'LN', 'MD', 'NW', 'NZ', 'SC', 'SO', 'SW', 'XR']

>>> keys_to_pfx = lor.get_keys_to_prefixes(prefixes_only=False)

>>> type(keys_to_pfx)
dict
>>> list(keys_to_pfx.keys())
['Key to prefixes', 'Last updated date']

>>> print(keys_to_pfx['Key to prefixes'].head())
  Prefixes                                  Name
0       CY                                 Wales
1       EA         South Eastern: East Anglia area
2       GW  Great Western (later known as Western)
```

(continues on next page)

(continued from previous page)

```
3       LN                  London & North Eastern
4       MD       North West: former Midlands lines
```

LOR.get_lor_page_urls

LOR.**get_lor_page_urls**(*update=False*, *verbose=False*)

Get URLs to PRIDE/LOR codes with different prefixes.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns a list of URLs of web pages hosting LOR codes for each prefix

Return type list

Example:

```
>>> from pyrcs.line_data import LOR

>>> lor = LOR()

>>> # lor_page_urls_ = lor.get_lor_page_urls(update=True, verbose=True)
>>> lor_page_urls_ = lor.get_lor_page_urls()

>>> lor_page_urls_[:2]
['http://www.railwaycodes.org.uk/pride/pridecy.shtm',
 'http://www.railwaycodes.org.uk/pride/prideea.shtm']
```

line_name

Collect British railway line names.

Class

LineNames([data_dir, update, verbose])	A class for collecting British railway line names.

LineNames

class line_name.**LineNames**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting British railway line names.

Parameters

- **data_dir** (*str or None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to True

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type data
- **HomeURL** (*str*) – URL of the main homepage
- **SourceURL** (*str*) – URL of the data web page
- **LUDKey** (*str*) – key of the last updated date
- **LUD** (*str*) – last updated date
- **Catalogue** (*dict*) – catalogue of the data
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory

Example:

```
>>> from pyrcs.line_data import LineNames

>>> ln = LineNames()

>>> print(ln.Name)
Railway line names

>>> print(ln.SourceURL)
http://www.railwaycodes.org.uk/misc/line_names.shtm
```

Methods

collect_line_names([confirmation_requ ...])	Collect data of railway line names from source web page.
fetch_line_names([update, pickle_it, ...])	Fetch data of railway line names from local backup.

LineNames.collect_line_names

LineNames.**collect_line_names**(*confirmation_required=True*, *verbose=False*)

Collect data of railway line names from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to require users to confirm and proceed, defaults to True
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns railway line names and routes data and date of when the data was last updated

Return type dict or None

Example:

```
>>> from pyrcs.line_data import LineNames

>>> ln = LineNames()

>>> line_names_dat = ln.collect_line_names(confirmation_required=False)

>>> type(line_names_dat)
dict
>>> list(line_names_dat.keys())
['Line names', 'Last updated date']

>>> print(line_names_dat['Line names'].head())
             Line name  ... Route_note
0           Abbey Line  ...       None
1        Airedale Line  ...       None
2          Argyle Line  ...       None
3     Arun Valley Line  ...       None
4  Atlantic Coast Line  ...       None
[5 rows x 3 columns]
```

LineNames.fetch_line_names

LineNames.**fetch_line_names**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch data of railway line names from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to False
- **data_dir** (*str or None*) – name of package data folder, defaults to None

- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns railway line names and routes data and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.line_data import LineNames

>>> ln = LineNames()

>>> # line_names_dat = ln.fetch_line_names(update=True, verbose=True)
>>> line_names_dat = ln.fetch_line_names()

>>> type(line_names_dat)
dict
>>> list(line_names_dat.keys())
['Line names', 'Last updated date']

>>> print(line_names_dat['Line names'].head())
             Line name  ... Route_note
0           Abbey Line  ...       None
1        Airedale Line  ...       None
2          Argyle Line  ...       None
3     Arun Valley Line  ...       None
4  Atlantic Coast Line  ...       None
[5 rows x 3 columns]
```

trk_diagr

Collect British railway track diagrams.

Class

TrackDiagrams([data_dir, verbose])	A class for collecting British railway track diagrams.

TrackDiagrams

class trk_diagr.**TrackDiagrams**(*data_dir=None*, *verbose=True*)

A class for collecting British railway track diagrams.

Parameters

- **data_dir** (*str or None*) – name of data directory, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `True`

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type data
- **HomeURL** (*str*) – URL of the main homepage
- **SourceURL** (*str*) – URL of the data web page
- **LUDKey** (*str*) – key of the last updated date
- **LUD** (*str*) – last updated date
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory

Example:

```
>>> from pyrcs.line_data import TrackDiagrams

>>> td = TrackDiagrams()

>>> print(td.Name)
Railway track diagrams (some samples)

>>> print(td.SourceURL)
http://www.railwaycodes.org.uk/track/diagrams0.shtm
```

Methods

collect_sample_catalogue([...])	Collect catalogue of sample railway track diagrams from source web page.
fetch_sample_catalogue([update, pickle_it, ...])	Fetch catalogue of sample railway track diagrams from local backup.
get_track_diagrams_items([update, verbose])	Get catalogue of track diagrams.

TrackDiagrams.collect_sample_catalogue

TrackDiagrams.**collect_sample_catalogue**(*confirmation_required=True*, *verbose=False*)

Collect catalogue of sample railway track diagrams from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to require users to confirm and proceed, defaults to True
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns catalogue of sample railway track diagrams and date of when the data

was last updated

Return type dict, None

Example:

```
>>> from pyrcs.line_data import TrackDiagrams

>>> td = TrackDiagrams()

>>> track_diagrams_catalog = td.collect_sample_catalogue()
To collect the catalogue of sample track diagrams? [No]|Yes: yes

>>> type(track_diagrams_catalog)
dict
>>> list(track_diagrams_catalog.keys())
['Track diagrams', 'Last updated date']

>>> td_dat = track_diagrams_catalog['Track diagrams']

>>> type(td_dat)
dict
>>> list(td_dat.keys())
['Main line diagrams', 'Tram systems', 'London Underground', 'Miscellaneous']
```

TrackDiagrams.fetch_sample_catalogue

TrackDiagrams.**fetch_sample_catalogue**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch catalogue of sample railway track diagrams from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns catalogue of sample railway track diagrams and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.line_data import TrackDiagrams

>>> td = TrackDiagrams()
```

(continues on next page)

(continued from previous page)

```
>>> # trk_diagr_cat = td.fetch_sample_catalogue(update=True, verbose=True)
>>> trk_diagr_cat = td.fetch_sample_catalogue()

>>> type(trk_diagr_cat)
dict
>>> list(trk_diagr_cat.keys())
['Track diagrams', 'Last updated date']

>>> td_dat = trk_diagr_cat['Track diagrams']

>>> type(td_dat)
dict
>>> list(td_dat.keys())
['Main line diagrams', 'Tram systems', 'London Underground', 'Miscellaneous']

>>> print(td_dat['Main line diagrams'][1])
                            Description                            FileURL
0  South Central area (1985) 10.4Mb file  http://www.railwaycodes.org.uk/...
1   South Eastern area (1976) 5.4Mb file  http://www.railwaycodes.org.uk/...
```

TrackDiagrams.get_track_diagrams_items

TrackDiagrams.**get_track_diagrams_items**(*update=False*, *verbose=False*)

Get catalogue of track diagrams.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns catalogue of railway station data

Return type dict

Example:

```
>>> from pyrcs.line_data import TrackDiagrams

>>> td = TrackDiagrams()

>>> # trk_diagr_items = td.get_track_diagrams_items(update=True, verbose=True)
>>> trk_diagr_items = td.get_track_diagrams_items()

>>> type(trk_diagr_items)
dict
>>> print(trk_diagr_items.keys())
dict_keys(['Track diagrams'])
```

3.1.2 other_assets

A collection of modules for collecting other assets. See also *pyrcs.collector.OtherAssets*.

Submodules

sig_box	Collect signal box prefix codes.
tunnel	Collect codes of railway tunnel lengths.
viaduct	Collect codes of railway viaducts.
station	Collect railway station data.
depot	Collect depots codes.
feature	Collect codes of infrastructure features.

sig_box

Collect signal box prefix codes.

Class

SignalBoxes([data_dir, update, verbose])	A class for collecting signal box prefix codes.

SignalBoxes

class sig_box.**SignalBoxes**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting signal box prefix codes.

Parameters

- **data_dir** (*str*, *None*) – name of data directory, defaults to `None`
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `True`

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type data
- **HomeURL** (*str*) – URL of the main homepage
- **LUDKey** (*str*) – key of the last updated date
- **LUD** (*str*) – last updated date

- **Catalogue** (*dict*) – catalogue of the data
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory
- **NonNationalRailKey** (*str*) – key of the dict-type data of non-national rail
- **NonNationalRailPickle** (*str*) – name of the pickle file of non-national rail data
- **IrelandKey** (*str*) – key of the dict-type data of Ireland
- **IrelandPickle** (*str*) – name of the pickle file of Ireland data
- **WRMASDKey** (*str*) – key of the dict-type data of WR MAS dates
- **WRMASDPickle** (*str*) – name of the pickle file of WR MAS dates data
- **MSBKey** (*str*) – key of the dict-type data of signal box bell codes
- **MSBPickle** (*str*) – name of the pickle file of signal box bell codes

Example:

```
>>> from pyrcs.other_assets import SignalBoxes

>>> sb = SignalBoxes()

>>> print(sb.Name)
Signal box prefix codes

>>> print(sb.SourceURL)
http://www.railwaycodes.org.uk/signal/signal_boxes0.shtm
```

Methods

collect_non_national_rail_codes([...]	Collect signal box prefix codes of non-national rail from source web page.
collect_prefix_codes(initial[, update, verbose])	Collect signal box prefix codes for the given `initial` from source web page.
fetch_non_national_rail_codes([upda ...])	Fetch signal box prefix codes of non-national rail from local backup.
fetch_prefix_codes([update, pickle_it, ...])	Fetch signal box prefix codes from local backup.

SignalBoxes.collect_non_national_rail_codes

SignalBoxes.**collect_non_national_rail_codes**(*confirmation_required=True*, *verbose=False*)

Collect signal box prefix codes of non-national rail from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to require users to confirm and proceed, defaults to True
- **verbose** (*bool*, *int*) – whether to print relevant information in console as the function runs, defaults to False

Returns signal box prefix codes of non-national rail

Return type dict, None

Example:

```
>>> from pyrcs.other_assets import SignalBoxes

>>> sb = SignalBoxes()

>>> nnr_codes_dat = sb.collect_non_national_rail_codes()
To collect signal box data of non-national rail? [No]|Yes: yes

>>> type(nnr_codes_dat)
dict
>>> list(nnr_codes_dat.keys())
['Non-National Rail', 'Last updated date']

>>> nnr_codes = nnr_codes_dat['Non-National Rail']

>>> type(nnr_codes)
dict
>>> list(nnr_codes.keys())
['Croydon Tramlink signals',
 'Docklands Light Railway signals',
 'Edinburgh Tramway signals',
 'Glasgow Subway signals',
 'London Underground signals',
 'Luas signals',
 'Manchester Metrolink signals',
 'Midland Metro signals',
 'Nottingham Tram signals',
 'Sheffield Supertram signals',
 'Tyne & Wear Metro signals',
 'Heritage, minor and miniature railways and other "special" signals']
```

SignalBoxes.collect_prefix_codes

SignalBoxes.**collect_prefix_codes**(*initial*, *update=False*, *verbose=False*)

Collect signal box prefix codes for the given `initial` from source web page.

Parameters

- **initial** (*str*) – initial letter of signal box name (for specifying a target URL)
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool* , *int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of signal box prefix codes for the given `initial` and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import SignalBoxes

>>> sb = SignalBoxes()

>>> # sb_a = sb.collect_prefix_codes(initial='a', update=True, verbose=True)
>>> sb_a = sb.collect_prefix_codes(initial='a')

>>> type(sb_a)
dict
>>> list(sb_a.keys())
['A', 'Last updated date']

>>> signal_boxes_a_codes = sb_a['A']

>>> type(signal_boxes_a_codes)
pandas.core.frame.DataFrame
>>> print(signal_boxes_a_codes.head())
  Code               Signal Box  ...             Closed        Control to
0   AF  Abbey Foregate Junction  ...
1   AJ           Abbey Junction  ...  16 February 1992     Nuneaton (NN)
2    R           Abbey Junction  ...  16 February 1992     Nuneaton (NN)
3   AW               Abbey Wood  ...      13 July 1975      Dartford (D)
4   AE         Abbey Works East  ...   1 November 1987  Port Talbot (PT)
[5 rows x 8 columns]
```

SignalBoxes.fetch_non_national_rail_codes

SignalBoxes.**fetch_non_national_rail_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch signal box prefix codes of non-national rail from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str*, *None*) – name of package data folder, defaults to `None`
- **verbose** (*bool*, *int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns signal box prefix codes of non-national rail

Return type dict

Example:

```
>>> from pyrcs.other_assets import SignalBoxes

>>> sb = SignalBoxes()

>>> # nnr_codes = sb.fetch_non_national_rail_codes(update=True, verbose=True)
>>> nnr_codes = sb.fetch_non_national_rail_codes()

>>> type(nnr_codes)
dict
>>> list(nnr_codes.keys())
['Non-National Rail', 'Last updated date']

>>> nnr_codes_ = nnr_codes['Non-National Rail']
>>> type(nnr_codes_)
dict
>>> list(nnr_codes_.keys())
['Croydon Tramlink signals',
 'Docklands Light Railway signals',
 'Edinburgh Tramway signals',
 'Glasgow Subway signals',
 'London Underground signals',
 'Luas signals',
 'Manchester Metrolink signals',
 'Midland Metro signals',
 'Nottingham Tram signals',
 'Sheffield Supertram signals',
 'Tyne & Wear Metro signals',
 'Heritage, minor and miniature railways and other "special" signals']

>>> lu_signals = nnr_codes_['London Underground signals']
```

(continues on next page)

(continued from previous page)

```
>>> type(lu_signals)
list
>>> print(lu_signals[0].head())
  Code  ... Became or taken over by (where known)
0  BMX  ...                                      -
1    A  ...                                      -
2    S  ...                                      -
3    X  ...                                      -
4    R  ...                                      -
[5 rows x 5 columns]
```

SignalBoxes.fetch_prefix_codes

SignalBoxes.**fetch_prefix_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch signal box prefix codes from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str* , *None*) – name of package data folder, defaults to `None`
- **verbose** (*bool* , *int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of location codes and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import SignalBoxes

>>> sb = SignalBoxes()

>>> # sb_prefix_codes_dat = sb.fetch_prefix_codes(update=True, verbose=True)
>>> sb_prefix_codes_dat = sb.fetch_prefix_codes()

>>> type(sb_prefix_codes_dat)
dict
>>> list(sb_prefix_codes_dat.keys())
['Signal boxes', 'Last updated date']

>>> sb_prefix_codes = sb_prefix_codes_dat['Signal boxes']

>>> type(sb_prefix_codes)
pandas.core.frame.DataFrame
```

(continues on next page)

(continued from previous page)

```
>>> print(sb_prefix_codes.head())
  Code               Signal Box  ...            Closed        Control to
0   AF  Abbey Foregate Junction  ...
1   AJ           Abbey Junction  ...  16 February 1992     Nuneaton (NN)
2    R           Abbey Junction  ...  16 February 1992     Nuneaton (NN)
3   AW               Abbey Wood  ...      13 July 1975      Dartford (D)
4   AE         Abbey Works East  ...   1 November 1987  Port Talbot (PT)
[5 rows x 8 columns]
```

tunnel

Collect codes of railway tunnel lengths.

Class

Tunnels([data_dir, update, verbose])	A class for collecting railway tunnel lengths.

Tunnels

class tunnel.**Tunnels**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting railway tunnel lengths.

Parameters

- **data_dir** (*str*, *None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to True

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type data
- **HomeURL** (*str*) – URL of the main homepage
- **SourceURL** (*str*) – URL of the data web page
- **LUDKey** (*str*) – key of the last updated date
- **LUD** (*str*) – last updated date
- **Catalogue** (*dict*) – catalogue of the data
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory

- **P1Key** (*str*) – key of the dict-type data of Page 1
- **P2Key** (*str*) – key of the dict-type data of Page 2
- **P3Key** (*str*) – key of the dict-type data of Page 3
- **P4Key** (*str*) – key of the dict-type data of Page 4

Example:

```
>>> from pyrcs.other_assets import Tunnels

>>> tunl = Tunnels()

>>> print(tunl.Name)
Railway tunnel lengths

>>> print(tunl.SourceURL)
http://www.railwaycodes.org.uk/tunnels/tunnels0.shtm
```

Methods

collect_lengths_by_page(page_no[, update, ...])	Collect data of railway tunnel lengths for a page number from source web page.
fetch_tunnel_lengths([update, pickle_it, ...])	Fetch data of railway tunnel lengths from local backup.
parse_length(x)	Parse data in `'Length'` column, i.e. convert miles/yards to metres.

Tunnels.collect_lengths_by_page

Tunnels.**collect_lengths_by_page**(*page_no*, *update=False*, *verbose=False*)

Collect data of railway tunnel lengths for a page number from source web page.

Parameters

- **page_no** (*int*, *str*) – page number; valid values include `1`, `2`, `3` and `4`
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool*, *int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns tunnel lengths data of the given `page_no` and date of when the data was last updated

Return type dict

Examples:

```
>>> from pyrcs.other_assets import Tunnels

>>> tunl = Tunnels()

>>> tunl_len_1 = tunl.collect_lengths_by_page(page_no=1)
>>> type(tunl_len_1)
dict
>>> list(tunl_len_1.keys())
['Page 1 (A-F)', 'Last updated date']

>>> tunl_len_4 = tunl.collect_lengths_by_page(page_no=4)
>>> type(tunl_len_4)
dict
>>> list(tunl_len_4.keys())
['Page 4 (others)', 'Last updated date']
```

Tunnels.fetch_tunnel_lengths

Tunnels.**fetch_tunnel_lengths**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch data of railway tunnel lengths from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str*, *None*) – name of package data folder, defaults to `None`
- **verbose** (*bool*, *int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns railway tunnel lengths data (including the name, length, owner and relative location) and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Tunnels

>>> tunl = Tunnels()

>>> # tunl_len_data = tunl.fetch_tunnel_lengths(update=True, verbose=True)
>>> tunl_len_data = tunl.fetch_tunnel_lengths()

>>> type(tunl_len_data)
dict
>>> list(tunl_len_data.keys())
['Tunnels', 'Last updated date']
```

(continues on next page)

(continued from previous page)

```
>>> tunl_len_dat = tunl_len_data['Tunnels']
>>> type(tunl_len_dat)
dict
>>> list(tunl_len_dat.keys())
['Page 1 (A-F)', 'Page 2 (G-P)', 'Page 3 (Q-Z)', 'Page 4 (others)']

>>> page_1 = tunl_len_dat['Page 1 (A-F)']
>>> print(page_1.head())
            Name  Other names, remarks  ...   Length_metres Length_notes
0    Abbotscliffe                       ...       1775.7648          NaN
1      Abercanaid           see Merthyr  ...             NaN  Unavailable
2     Aberchalder         see Loch Oich  ...             NaN  Unavailable
3  Aberdovey No 1  also called Frongoch  ...        182.8800          NaN
4  Aberdovey No 2    also called Morfor  ...        200.2536          NaN
[5 rows x 12 columns]
```

Tunnels.parse_length

static Tunnels.**parse_length**(*x*)

Parse data in `'Length'` column, i.e. convert miles/yards to metres.

Parameters **x** (*str*, *None*) – raw length data

Returns parsed length data and, if any, additional information associated with it

Return type tuple

Examples:

```
>>> from pyrcs.other_assets import Tunnels

>>> tunl = Tunnels()

>>> tunl.parse_length('')
(nan, 'Unavailable')

>>> tunl.parse_length('1m 182y')
(1775.7648, None)

>>> tunl.parse_length('formerly 0m236y')
(215.7984, 'Formerly')

>>> tunl.parse_length('0.325km (0m 356y)')
(325.5264, '0.325km')

>>> tunl.parse_length("0m 48yd- (['0m 58yd'])")
(48.4632, '43.89-53.04 metres')
```

viaduct

Collect codes of railway viaducts.

Class

Viaducts([data_dir, update, verbose])	A class for collecting railway viaducts.

Viaducts

class viaduct.**Viaducts**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting railway viaducts.

Parameters

- **data_dir** (*str , None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to True

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type data
- **HomeURL** (*str*) – URL of the main homepage
- **SourceURL** (*str*) – URL of the data web page
- **LUDKey** (*str*) – key of the last updated date
- **LUD** (*str*) – last updated date
- **Catalogue** (*dict*) – catalogue of the data
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory
- **P1Key** (*str*) – key of the dict-type data of Page 1
- **P2Key** (*str*) – key of the dict-type data of Page 2
- **P3Key** (*str*) – key of the dict-type data of Page 3
- **P4Key** (*str*) – key of the dict-type data of Page 4
- **P5Key** (*str*) – key of the dict-type data of Page 5
- **P6Key** (*str*) – key of the dict-type data of Page 6

Example:

```
>>> from pyrcs.other_assets import Viaducts

>>> vdct = Viaducts()

>>> print(vdct.Name)
Railway viaducts

>>> print(vdct.SourceURL)
http://www.railwaycodes.org.uk/viaducts/viaducts0.shtm
```

Methods

collect_viaduct_codes_by_page(page_ . . .])	Collect data of railway viaducts for a given page number from source web page.
fetch_viaduct_codes([update, pickle_it, . . .])	Fetch data of railway viaducts from local backup.

Viaducts.collect_viaduct_codes_by_page

Viaducts.**collect_viaduct_codes_by_page**(*page_no*, *update=False*, *verbose=False*)

Collect data of railway viaducts for a given page number from source web page.

Parameters

- **page_no** (*int* , *str*) – page number; valid values include 1, 2, 3, 4, 5, and 6
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns railway viaducts data of the given page_no and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Viaducts

>>> vdct = Viaducts()

>>> # vd1 = vdct.collect_viaduct_codes_by_page(1, update=True, verbose=True)
>>> vd1 = vdct.collect_viaduct_codes_by_page(page_no=1)

>>> type(vd1)
dict
>>> list(vd1.keys())
```

(continues on next page)

(continued from previous page)

```
['Page 1 (A-C)', 'Last updated date']

>>> viaducts_1 = vd1['Page 1 (A-C)']
>>> print(viaducts_1.head())
       Name  ... Spans
0  7 Arches  ...     7
1   36 Arch  ...    36
2   42 Arch  ...
3     A6120  ...
4      A698  ...
[5 rows x 7 columns]
```

Viaducts.fetch_viaduct_codes

Viaducts.**fetch_viaduct_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch data of railway viaducts from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str*, *None*) – name of package data folder, defaults to `None`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns railway viaducts data and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Viaducts

>>> vdct = Viaducts()

>>> # viaducts_codes = vdct.fetch_viaduct_codes(update=True, verbose=True)
>>> viaducts_codes = vdct.fetch_viaduct_codes()

>>> type(viaducts_codes)
dict
>>> list(viaducts_codes.keys())
['Viaducts', 'Last updated date']

>>> viaducts_dat = viaducts_codes['Viaducts']
>>> type(viaducts_dat)
dict
```

(continues on next page)

(continued from previous page)

```
>>> list(viaducts_dat.keys())
['Page 1 (A-C)',
 'Page 2 (D-G)',
 'Page 3 (H-K)',
 'Page 4 (L-P)',
 'Page 5 (Q-S)',
 'Page 6 (T-Z)']

>>> viaducts_dat_6 = viaducts_dat['Page 6 (T-Z)']
>>> print(viaducts_dat_6.head())
         Name  ... Spans
0        Taff  ...
1        Taff  ...
2  Taff River  ...
3  Taffs Well  ...
4        Tame  ...     4
[5 rows x 7 columns]
```

station

Collect railway station data.

Class

Stations([data_dir, verbose])	A class for collecting railway station data.

Stations

class station.**Stations**(*data_dir=None*, *verbose=True*)

A class for collecting railway station data.

Parameters

- **data_dir** (*str*, *None*) – name of data directory, defaults to None
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to True

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type data
- **HomeURL** (*str*) – URL of the main homepage
- **SourceURL** (*str*) – URL of the data web page
- **LUDKey** (*str*) – key of the last updated date

- **LUD** (*str*) – last updated date
- **Catalogue** (*dict*) – catalogue of the data
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory
- **StnKey** (*str*) – key of the dict-type data of railway stations
- **StnPickle** (*str*) – name of the pickle file of railway station data
- **BilingualKey** (*str*) – key of the dict-type data of bilingual names
- **SpStnNameSignKey** (*str*) – key of the dict-type data of sponsored station name signs
- **NSFOKey** (*str*) – key of the dict-type data of stations not served by SFO
- **IntlKey** (*str*) – key of the dict-type data of UK international railway stations
- **TriviaKey** (*str*) – key of the dict-type data of UK railway station trivia
- **ARKey** (*str*) – key of the dict-type data of UK railway station access rights
- **BarrierErrKey** (*str*) – key of the dict-type data of railway station barrier error codes

Example:

```
>>> from pyrcs.other_assets import Stations

>>> stn = Stations()

>>> print(stn.Name)
Railway station data

>>> print(stn.SourceURL)
http://www.railwaycodes.org.uk/stations/station0.shtm
```

Methods

collect_station_data_by_initial(initial, ...])	Collect railway station data for the given initial letter.
extended_info(info_dat, name)	Get extended information of the owners/operators.
fetch_station_data([update, pickle_it, ...])	Fetch railway station data from local backup.
get_station_data_catalogue([update, verbose])	Get catalogue of railway station data.

Stations.collect_station_data_by_initial

Stations.**collect_station_data_by_initial**(*initial*, *update=False*, *verbose=False*)

Collect railway station data for the given `initial` letter.

Parameters

- **initial** (*str*) – initial letter of station data (including the station name, ELR, mileage, status, owner, operator, degrees of longitude and latitude, and grid reference) for specifying URL
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool*, *int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns railway station data for the given `initial` letter and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Stations

>>> stn = Stations()

>>> # sa = stn.collect_station_data_by_initial('a', update=True, verbose=True)
>>> sa = stn.collect_station_data_by_initial(initial='a')

>>> type(sa)
dict
>>> list(sa.keys())
['A', 'Last updated date']

>>> print(sa['A'].head())
           Station   ELR  ... Prev_Operator_6 Prev_Operator_Period_6
0       Abbey Wood   NKL  ...            None                    None
1       Abbey Wood  XRS3  ...            None                    None
2             Aber   CAR  ...            None                    None
3  Abercynon North   ABD  ...            None                    None
4                    ABD  ...            None                    None
[5 rows x 28 columns]
```

Stations.extended_info

Stations.**extended_info**(*info_dat*, *name*)

Get extended information of the owners/operators.

Parameters

- **info_dat** (*pandas.Series*) – raw data of owners/operators
- **name** (*str*) – original column name of the owners/operators data

Returns extended information of the owners/operators

Return type pandas.DataFrame

Stations.fetch_station_data

Stations.**fetch_station_data**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch railway station data from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to False
- **data_dir** (*str*, *None*) – name of package data folder, defaults to None
- **verbose** (*bool*, *int*) – whether to print relevant information in console as the function runs, defaults to False

Returns railway station data (including the station name, ELR, mileage, status, owner, operator, degrees of longitude and latitude, and grid reference) and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Stations

>>> stn = Stations()

>>> # rail_stn_data = stn.fetch_station_data(update=True, verbose=True)
>>> rail_stn_data = stn.fetch_station_data()

>>> type(rail_stn_data)
dict
>>> list(rail_stn_data.keys())
['Railway station data', 'Last updated date']

>>> rail_stn_dat = rail_stn_data['Railway station data']
```

(continues on next page)

(continued from previous page)

```
>>> type(rail_stn_dat)
pandas.core.frame.DataFrame
>>> print(rail_stn_dat.head())
         Station   ELR  ... Prev_Operator_6 Prev_Operator_Period_6
2606              MRL1  ...            None                    None
723                TAT  ...            None                    None
89                 ABD  ...            None                    None
90                 CAM  ...            None                    None
85    Abbey Wood   NKL  ...            None                    None
[5 rows x 32 columns]
```

Stations.get_station_data_catalogue

Stations.**get_station_data_catalogue**(*update=False*, *verbose=False*)

Get catalogue of railway station data.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Returns catalogue of railway station data

Return type dict

Example:

```
>>> from pyrcs.other_assets import Stations

>>> stn = Stations()

>>> # stn_data_cat = stn.get_station_data_catalogue(update=True, verbose=True)
>>> stn_data_cat = stn.get_station_data_catalogue()

>>> type(stn_data_cat)
dict
>>> list(stn_data_cat.keys())
['Railway station data',
 'Sponsored signs',
 'International',
 'Trivia',
 'Access rights',
 'Barrier error codes']
```

depot

Collect depots codes.

Class

Depots([data_dir, update, verbose])	A class for collecting depot codes.

Depots

class depot.**Depots**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting depot codes.

Parameters

- **data_dir** (*str or None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type data
- **HomeURL** (*str*) – URL of the main homepage
- **SourceURL** (*str*) – URL of the data web page
- **LUDKey** (*str*) – key of the last updated date
- **LUD** (*str*) – last updated date
- **Catalogue** (*dict*) – catalogue of the data
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory
- **TCTKey** (*str*) – key of the dict-type data of two character TOPS codes
- **TCTPickle** (*str*) – name of the pickle file of two character TOPS codes
- **FDPTKey** (*str*) – key of the dict-type data of four digit pre-TOPS codes
- **FDPTPickle** (*str*) – name of the pickle file of four digit pre-TOPS codes
- **S1950Key** (*str*) – key of the dict-type data of 1950 system (pre-TOPS) codes
- **S1950Pickle** (*str*) – name of the pickle file of 1950 system (pre-TOPS) codes
- **GWRKey** (*str*) – key of the dict-type data of GWR codes
- **GWRPickle** (*str*) – name of the pickle file of GWR codes

Example:

```
>>> from pyrcs.other_assets import Depots

>>> depots = Depots()

>>> print(depots.Name)
Depot codes

>>> print(depots.SourceURL)
http://www.railwaycodes.org.uk/depots/depots0.shtm
```

Methods

collect_1950_system_codes([...])	Collect 1950 system (pre-TOPS) codes from source web page.
collect_four_digit_pre_tops_codes([	Collect four-digit pre-TOPS codes from source web page.
collect_gwr_codes([confirmation_requi ...])	Collect Great Western Railway (GWR) depot codes from source web page.
collect_two_char_tops_codes([...])	Collect two-character TOPS codes from source web page.
fetch_1950_system_codes([update, pickle_it, ...])	Fetch 1950 system (pre-TOPS) codes from local backup.
fetch_depot_codes([update, pickle_it, ...])	Fetch depots codes from local backup.
fetch_four_digit_pre_tops_codes([up ...])	Fetch four-digit pre-TOPS codes from local backup.
fetch_gwr_codes([update, pickle_it, ...])	Fetch Great Western Railway (GWR) depot codes from local backup.
fetch_two_char_tops_codes([update, ...])	Fetch two-character TOPS codes from local backup.

Depots.collect_1950_system_codes

Depots.**collect_1950_system_codes**(*confirmation_required=True*, *verbose=False*)

Collect 1950 system (pre-TOPS) codes from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool*, *int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of 1950 system (pre-TOPS) codes and date of when the data was last updated

Return type dict or None

Example:

```
>>> from pyrcs.other_assets import Depots

>>> depots = Depots()

>>> system_1950_codes_dat = depots.collect_1950_system_codes()
To collect data of 1950 system (pre-TOPS) codes? [No]|Yes: yes

>>> type(system_1950_codes_dat)
dict
>>> list(system_1950_codes_dat.keys())
['1950 system (pre-TOPS) codes', 'Last updated date']

>>> print(system_1950_codes_dat['1950 system (pre-TOPS) codes'].head())
  Code click to sort  ...                                              Notes
0                 1A  ...               From 1950. Became WN from 6 May 1973
1                 1B  ...                        From 1950. To 3 January 1966
2                 1C  ...               From 1950. Became WJ from 6 May 1973
3                 1D  ...  Previously 13B to 9 June 1950. Became 1J from ...
4                 1D  ...  Previously 14F to 31 August 1963. Became ME fr...
[5 rows x 3 columns]
```

Depots.collect_four_digit_pre_tops_codes

Depots.**collect_four_digit_pre_tops_codes**(*confirmation_required=True*, *verbose=False*)

Collect four-digit pre-TOPS codes from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool*, *int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of two-character TOPS codes and date of when the data was last updated

Return type dict or None

Example:

```
>>> from pyrcs.other_assets import Depots

>>> depots = Depots()

>>> fdpt_codes = depots.collect_four_digit_pre_tops_codes()
```

(continues on next page)

(continued from previous page)

```
To collect data of four digit pre-TOPS codes? [No]|Yes: yes

>>> type(fdpt_codes)
dict
>>> list(fdpt_codes.keys())
['Four digit pre-TOPS codes', 'Last updated date']

>>> print(fdpt_codes['Four digit pre-TOPS codes'].head())
   Code             Depot name          Region
0  2000             Accrington  London Midland
1  2001   Derby Litchurch Lane      Main Works
2  2003              Blackburn  London Midland
3  2004  Bolton Trinity Street  London Midland
4  2006                Burnley  London Midland
```

Depots.collect_gwr_codes

Depots.**collect_gwr_codes**(*confirmation_required=True*, *verbose=False*)

Collect Great Western Railway (GWR) depot codes from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool*, *int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of GWR depot codes and date of when the data was last updated

Return type dict or None

Example:

```
>>> from pyrcs.other_assets import Depots

>>> depots = Depots()

>>> gwr_codes_dat = depots.collect_gwr_codes()
To collect data of GWR codes? [No]|Yes: yes

>>> type(gwr_codes_dat)
dict
>>> list(gwr_codes_dat.keys())
['GWR codes', 'Last updated date']

>>> type(gwr_codes_dat['GWR codes'])
dict
>>> list(gwr_codes_dat['GWR codes'].keys())
['Alphabetical codes', 'Numerical codes']

>>> print(gwr_codes_dat['GWR codes']['Alphabetical codes'].head())
```

(continues on next page)

(continued from previous page)

```
    Code   Depot name
0  ABEEG     Aberbeeg
1    ABG     Aberbeeg
2    AYN    Abercynon
3   ABDR     Aberdare
4    ABH  Aberystwyth
```

Depots.collect_two_char_tops_codes

Depots.**collect_two_char_tops_codes**(*confirmation_required=True*, *verbose=False*)

Collect two-character TOPS codes from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool*, *int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of two-character TOPS codes and date of when the data was last updated

Return type dict or None

Example:

```
>>> from pyrcs.other_assets import Depots

>>> depots = Depots()

>>> tct_codes = depots.collect_two_char_tops_codes()
To collect data of two character TOPS codes? [No]|Yes: yes

>>> type(tct_codes)
dict
>>> list(tct_codes.keys())
['Two character TOPS codes', 'Last updated date']

>>> print(tct_codes['Two character TOPS codes'].head())
  Code click to sort  ...                Notes
0                 AB  ...          Closed 1987
1                 AB  ...
2                 AC  ...  Became WH from 1994
3                 AC  ...
4                 AD  ...
[5 rows x 5 columns]
```

Depots.fetch_1950_system_codes

Depots.**fetch_1950_system_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch 1950 system (pre-TOPS) codes from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to False
- **data_dir** (*str or None*) – name of package data folder, defaults to None
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns data of 1950 system (pre-TOPS) codes and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Depots

>>> depots = Depots()

>>> # s1950_codes = depots.fetch_1950_system_codes(update=True, verbose=True)
>>> s1950_codes = depots.fetch_1950_system_codes()

>>> system_1950_codes = s1950_codes['1950 system (pre-TOPS) codes']

>>> type(system_1950_codes)
pandas.core.frame.DataFrame
>>> print(system_1950_codes.head())
  Code click to sort  ...                                              Notes
0                 1A  ...                 From 1950. Became WN from 6 May 1973
1                 1B  ...                         From 1950. To 3 January 1966
2                 1C  ...                 From 1950. Became WJ from 6 May 1973
3                 1D  ...  Previously 13B to 9 June 1950. Became 1J from ...
4                 1D  ...  Previously 14F to 31 August 1963. Became ME fr...
[5 rows x 3 columns]
```

Depots.fetch_depot_codes

Depots.**fetch_depot_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch depots codes from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to False
- **data_dir** (*str or None*) – name of package data folder, defaults to None
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns data of depot codes and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Depots

>>> depots = Depots()

>>> # depot_codes_dat = depots.fetch_depot_codes(update=True, verbose=True)
>>> depot_codes_dat = depots.fetch_depot_codes()

>>> type(depot_codes_dat)
dict
>>> list(depot_codes_dat.keys())
['Depots', 'Last updated date']

>>> type(depot_codes_dat['Depots'])
dict
>>> list(depot_codes_dat['Depots'].keys())
['1950 system (pre-TOPS) codes',
 'Four digit pre-TOPS codes',
 'GWR codes',
 'Two character TOPS codes']

>>> print(depot_codes_dat['Depots']['Four digit pre-TOPS codes'].head())
   Code             Depot name          Region
0  2000             Accrington  London Midland
1  2001   Derby Litchurch Lane      Main Works
2  2003              Blackburn  London Midland
3  2004  Bolton Trinity Street  London Midland
4  2006                Burnley  London Midland
```

Depots.fetch_four_digit_pre_tops_codes

Depots.**fetch_four_digit_pre_tops_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch four-digit pre-TOPS codes from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of two-character TOPS codes and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Depots

>>> depots = Depots()

>>> # fdpt = depots.fetch_four_digit_pre_tops_codes(update=True, verbose=True)
>>> fdpt = depots.fetch_four_digit_pre_tops_codes()

>>> type(fdpt)
dict
>>> list(fdpt.keys())
['Four digit pre-TOPS codes', 'Last updated date']

>>> print(fdpt['Four digit pre-TOPS codes'].head())
   Code             Depot name          Region
0  2000             Accrington  London Midland
1  2001   Derby Litchurch Lane      Main Works
2  2003              Blackburn  London Midland
3  2004  Bolton Trinity Street  London Midland
4  2006                Burnley  London Midland
```

Depots.fetch_gwr_codes

Depots.**fetch_gwr_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch Great Western Railway (GWR) depot codes from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to False
- **data_dir** (*str or None*) – name of package data folder, defaults to None
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns data of GWR depot codes and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Depots

>>> depots = Depots()

>>> # gwr_codes_dat = depots.fetch_gwr_codes(update=True, verbose=True)
>>> gwr_codes_dat = depots.fetch_gwr_codes()

>>> gwr_codes = gwr_codes_dat['GWR codes']
>>> type(gwr_codes)
dict
>>> list(gwr_codes.keys())
['Alphabetical codes', 'Numerical codes']

>>> gwr_codes_alpha = gwr_codes['Alphabetical codes']
>>> type(gwr_codes_alpha)
pandas.core.frame.DataFrame
>>> print(gwr_codes_alpha.head())
    Code   Depot name
0  ABEEG     Aberbeeg
1    ABG     Aberbeeg
2    AYN    Abercynon
3   ABDR     Aberdare
4    ABH  Aberystwyth
```

Depots.fetch_two_char_tops_codes

Depots.**fetch_two_char_tops_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch two-character TOPS codes from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to False
- **data_dir** (*str or None*) – name of package data folder, defaults to None
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns data of two-character TOPS codes and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Depots

>>> depots = Depots()

>>> # tct_codes = depots.fetch_two_char_tops_codes(update=True, verbose=True)
>>> tct_codes = depots.fetch_two_char_tops_codes()

>>> type(tct_codes)
dict
>>> list(tct_codes.keys())
['Two character TOPS codes', 'Last updated date']

>>> print(tct_codes['Two character TOPS codes'].head())
  Code click to sort  ...                Notes
0                 AB  ...          Closed 1987
1                 AB  ...
2                 AC  ...  Became WH from 1994
3                 AC  ...
4                 AD  ...
[5 rows x 5 columns]
```

feature

Collect codes of infrastructure features.

This category includes:

- OLE neutral sections
- HABD and WILD
- Water troughs
- Telegraph codes
- Driver/guard buzzer codes

Class

Features([data_dir, update, verbose])	A class for collecting codes of infrastructure features.

Features

class feature.**Features**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting codes of infrastructure features.

Parameters

- **data_dir** (*str or None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to True

Variables

- **Name** (*str*) – name of the data
- **Key** (*str*) – key of the dict-type data
- **HomeURL** (*str*) – URL of the main homepage
- **LUDKey** (*str*) – key of the last updated date
- **Catalogue** (*dict*) – catalogue of the data
- **DataDir** (*str*) – path to the data directory
- **CurrentDataDir** (*str*) – path to the current data directory
- **HabdWildKey** (*str*) – key of the dict-type data of HABD and WILD
- **HabdWildPickle** (*str*) – name of the pickle file of HABD and WILD

- **OLENeutralNetworkKey** (*str*) – key of the dict-type data of OLE neutral sections
- **WaterTroughsKey** (*str*) – key of the dict-type data of water troughs
- **WaterTroughsPickle** (*str*) – name of the pickle file of water troughs
- **TelegraphKey** (*str*) – key of the dict-type data of telegraphic codes
- **TelegraphPickle** (*str*) – name of the pickle file of telegraphic codes
- **BuzzerKey** (*str*) – key of the dict-type data of buzzer codes
- **BuzzerPickle** (*str*) – name of the pickle file of buzzer codes

Example:

```
>>> from pyrcs.other_assets import Features

>>> features = Features()

>>> print(features.Name)
Infrastructure features
```

Methods

collect_buzzer_codes([...])	Collect buzzer codes from source web page.
collect_habds_and_wilds([...])	Collect codes of HABDs and WILDs from source web page.
collect_telegraph_codes([...])	Collect telegraph code words from source web page.
collect_water_troughs([...])	Collect codes of water troughs from source web page.
fetch_buzzer_codes([update, pickle_it, ...])	Fetch buzzer codes from local backup.
fetch_features_codes([update, pickle_it, ...])	Fetch features codes from local backup.
fetch_habds_and_wilds([update, pickle_it, ...])	Fetch codes of HABDs and WILDs from local backup.
fetch_telegraph_codes([update, pickle_it, ...])	Fetch telegraph code words from local backup.
fetch_water_troughs([update, pickle_it, ...])	Fetch codes of water troughs from local backup.

Features.collect_buzzer_codes

Features.**collect_buzzer_codes**(*confirmation_required=True*, *verbose=False*)

Collect buzzer codes from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Returns data of buzzer codes, and date of when the data was last updated

Return type dict or None

Example:

```
>>> from pyrcs.other_assets import Features

>>> features = Features()

>>> buz_codes_dat = features.collect_buzzer_codes()
To collect data of buzzer codes? [No]|Yes:  yes

>>> type(buz_codes_dat)
dict
>>> list(buz_codes_dat.keys())
['Buzzer codes', 'Last updated date']

>>> buz_codes = buz_codes_dat['Buzzer codes']
>>> print(buz_codes.head())
  Code number of buzzes or groups separated by pauses            Meaning
0                                                   1               Stop
1                                                 1-2        Close doors
2                                                   2     Ready to start
3                                                 2-2   Do not open doors
4                                                   3           Set back
```

Features.collect_habds_and_wilds

Features.**collect_habds_and_wilds**(*confirmation_required=True*, *verbose=False*)

Collect codes of HABDs and WILDs from source web page.

- HABDs - Hot axle box detectors
- WILDs - Wheel impact load detectors

Parameters

- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True

- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of HABDs and WILDs, and date of when the data was last updated

Return type dict or None

Example:

```
>>> from pyrcs.other_assets import Features

>>> features = Features()

>>> hw_codes_dat = features.collect_habds_and_wilds()
# To collect data of HABD and WILD? [No]|Yes: yes

>>> type(hw_codes_dat)
dict
>>> list(hw_codes_dat.keys())
['HABD and WILD', 'Last updated date']

>>> hw_codes = hw_codes_dat['HABD and WILD']

>>> type(hw_codes)
dict
>>> list(hw_codes.keys())
['HABD', 'WILD']

>>> habd = hw_codes['HABD']
>>> print(habd.head())
    ELR  ...                                              Notes
0  BAG2  ...
1  BAG2  ...  installed 29 September 1997, later adjusted to...
2  BAG2  ...                              previously at 74m 51ch
3  BAG2  ...                           removed 29 September 1997
4  BAG2  ...            present in 1969, later moved to 89m 0ch
[5 rows x 5 columns]

>>> wild = hw_codes['WILD']
>>> print(wild.head())
    ELR  ...                                 Notes
0  AYR3  ...
1  BAG2  ...
2  BML1  ...
3  BML1  ...
4  CGJ3  ...  moved to 183m 68ch 8 September 2018
[5 rows x 5 columns]
```

Features.collect_telegraph_codes

Features.**collect_telegraph_codes**(*confirmation_required=True*, *verbose=False*)

Collect telegraph code words from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of telegraph code words, and date of when the data was last updated

Return type dict or None

Example:

```
>>> from pyrcs.other_assets import Features

>>> features = Features()

>>> tel_codes_dat = features.collect_telegraph_codes()
To collect data of telegraphic codes? [No]|Yes: yes

>>> type(tel_codes_dat)
dict
>>> list(tel_codes_dat.keys())
['Telegraphic codes', 'Last updated date']

>>> tel_codes = tel_codes_dat['Telegraphic codes']

>>> type(tel_codes)
dict
>>> list(tel_codes.keys())
['Official codes', 'Unofficial codes']

>>> print(tel_codes['Official codes'].head())
     Code  ...                In use
0  ACACIA  ...     'companies', 1939
1     ACK  ...             BR, 1980s
2    ADEX  ...  GWR, 1939 BR, 1980s
3    AJAX  ...             BR, 1980s
4   ALERT  ...             BR, 1980s
[5 rows x 3 columns]
```

Features.collect_water_troughs

Features.**collect_water_troughs**(*confirmation_required=True*, *verbose=False*)

Collect codes of water troughs from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Returns data of water troughs, and date of when the data was last updated

Return type dict or None

Example:

```
>>> from pyrcs.other_assets import Features

>>> features = Features()

>>> wt_codes_dat = features.collect_water_troughs()
To collect data of water troughs? [No]|Yes: yes

>>> type(wt_codes_dat)
dict
>>> list(wt_codes_dat.keys())
['Water troughs', 'Last updated date']

>>> wt_codes = wt_codes_dat['Water troughs']
>>> print(wt_codes.head())
    ELR  Trough Name  ...                                        Notes
0   BEI    Eckington  ...                           Installed 1904
1   BHL  Aldermaston  ...                        Installed by 1904
2  CGJ2        Moore  ...                          Installed 1860s
3  CGJ6     Lea Road  ...  Installed 1885, taken out of use 8 May 1967
4  CGJ6        Brock  ...                          Installed 1860s
[5 rows x 5 columns]
```

Features.fetch_buzzer_codes

Features.**fetch_buzzer_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch buzzer codes from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to False

- **data_dir** (*str or None*) – name of package data folder, defaults to None
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns data of buzzer codes, and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Features

>>> features = Features()

>>> # buz_codes_dat = features.fetch_buzzer_codes(verbose=True, update=True)
>>> buz_codes_dat = features.fetch_buzzer_codes()

>>> type(buz_codes_dat)
dict
>>> list(buz_codes_dat.keys())
['Buzzer codes', 'Last updated date']

>>> buz_codes = buz_codes_dat['Buzzer codes']

>>> type(buz_codes)
pandas.core.frame.DataFrame
>>> print(buz_codes.head())
 Code (number of buzzes or groups separated by pauses)             Meaning
0                                                  1                Stop
1                                                1-2         Close doors
2                                                  2       Ready to start
3                                                2-2    Do not open doors
4                                                  3             Set back
```

Features.fetch_features_codes

Features.**fetch_features_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch features codes from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to False
- **data_dir** (*str or None*) – name of package data folder, defaults to None
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns data of features codes and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Features

>>> features = Features()

>>> # feat_codes_dat = features.fetch_features_codes(update=True, verbose=True)
>>> feat_codes_dat = features.fetch_features_codes()

>>> type(feat_codes_dat)
dict
>>> list(feat_codes_dat.keys())
['Features', 'Last updated date']

>>> feat_codes = feat_codes_dat['Features']

>>> type(feat_codes)
dict
>>> list(feat_codes.keys())
['National network neutral sections',
 'Buzzer codes',
 'HABD and WILD',
 'Telegraphic codes',
 'Water troughs']

 >>> print(feat_codes['National network neutral sections'].head())
    ELR         OHNS Name  Mileage    Tracks Dates
0  ARG1        Rutherglen   0m 3ch
1  ARG2   Finnieston East  4m 23ch      Down
2  ARG2   Finnieston West  4m 57ch        Up
3  AYR1  Shields Junction  0m 68ch    Up Ayr
4  AYR1  Shields Junction  0m 69ch  Down Ayr
```

Features.fetch_habds_and_wilds

Features.**fetch_habds_and_wilds**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch codes of HABDs and WILDs from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data_dir** (*str or None*) – name of package data folder, defaults to `None`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns data of hot axle box detectors (HABDs) and wheel impact load detectors (WILDs), and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Features

>>> features = Features()

>>> # hw_codes_dat = features.fetch_habds_and_wilds(update=True, verbose=True)
>>> hw_codes_dat = features.fetch_habds_and_wilds()

>>> hw_codes = hw_codes_dat['HABD and WILD']
>>> type(hw_codes)
dict
>>> list(hw_codes.keys())
['HABD', 'WILD']

>>> habd = hw_codes['HABD']
>>> print(habd.head())
    ELR  ...                                              Notes
0  BAG2  ...
1  BAG2  ...  installed 29 September 1997, later adjusted to...
2  BAG2  ...                              previously at 74m 51ch
3  BAG2  ...                           removed 29 September 1997
4  BAG2  ...            present in 1969, later moved to 89m 0ch
[5 rows x 5 columns]

>>> wild = hw_codes['WILD']
>>> print(wild.head())
    ELR  ...                                  Notes
0  AYR3  ...
1  BAG2  ...
2  BML1  ...
3  BML1  ...
4  CGJ3  ...  moved to 183m 68ch 8 September 2018
[5 rows x 5 columns]
```

Features.fetch_telegraph_codes

Features.**fetch_telegraph_codes**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch telegraph code words from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`

- **data_dir** (*str or None*) – name of package data folder, defaults to None
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns data of telegraph code words, and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Features

>>> features = Features()

>>> # tel_codes_dat = features.fetch_telegraph_codes(update=True, verbose=True)
>>> tel_codes_dat = features.fetch_telegraph_codes()

>>> tel_codes = tel_codes_dat['Telegraphic codes']
>>> type(tel_codes)
dict
>>> list(tel_codes.keys())
['Official codes', 'Unofficial codes']

>>> official_codes = tel_codes['Official codes']
>>> type(official_codes)
pandas.core.frame.DataFrame
>>> print(official_codes.head())
     Code  ...               In use
0  ACACIA  ...    'companies', 1939
1     ACK  ...            BR, 1980s
2    ADEX  ...  GWR, 1939 BR, 1980s
3    AJAX  ...            BR, 1980s
4   ALERT  ...            BR, 1980s
[5 rows x 3 columns]
```

Features.fetch_water_troughs

Features.**fetch_water_troughs**(*update=False*, *pickle_it=False*, *data_dir=None*, *verbose=False*)

Fetch codes of water troughs from local backup.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **pickle_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to False
- **data_dir** (*str or None*) – name of package data folder, defaults to None
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to False

Returns data of water troughs, and date of when the data was last updated

Return type dict

Example:

```
>>> from pyrcs.other_assets import Features

>>> features = Features()

>>> # wt_codes_dat = features.fetch_water_troughs(update=True, verbose=True)
>>> wt_codes_dat = features.fetch_water_troughs()

>>> type(wt_codes_dat)
dict
>>> list(wt_codes_dat.keys())
['Water troughs', 'Last updated date']

>>> wt_codes = wt_codes_dat['Water troughs']
>>> print(wt_codes.head())
    ELR  Trough Name  ...                                       Notes
0   BEI    Eckington  ...                                Installed 1904
1   BHL  Aldermaston  ...                              Installed by 1904
2  CGJ2        Moore  ...                                Installed 1860s
3  CGJ6     Lea Road  ...  Installed 1885, taken out of use 8 May 1967
4  CGJ6        Brock  ...                                Installed 1860s
[5 rows x 5 columns]
```

3.2 Modules

collector	Collect data of railway codes.
updater	Update package data.
utils	Utilities - Helper functions.

3.2.1 collector

Collect data of railway codes.

The current release includes only:

- line data
- other assets

LineData([update, verbose])	A class representation of all modules of the subpackage *pyrcs.line_data* for collecting line data.

continues on next page

Table 29 – continued from previous page	
OtherAssets([update, verbose])	A class representation of all modules of the subpackage *pyrcs.other_assets* for collecting other assets.

LineData

class pyrcs.collector.**LineData**(*update=False*, *verbose=True*)

A class representation of all modules of the subpackage *pyrcs.line_data* for collecting line data.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `True`

Examples:

```
>>> from pyrcs import LineData

>>> ld = LineData()

>>> # To get data of location codes
>>> location_codes_data = ld.LocationIdentifiers.fetch_location_codes()

>>> type(location_codes_data)
<class 'dict'>
>>> print(list(location_codes_data.keys()))
['Location codes', 'Other systems', 'Additional notes', 'Last updated date']
>>> location_codes_dat = location_codes_data['Location codes']
>>> print(location_codes_dat.head())
                              Location CRS  ... STANME_Note STANOX_Note
0                               Aachen      ...
1                   Abbeyhill Junction      ...
2                 Abbeyhill Signal E811      ...
3            Abbeyhill Turnback Sidings      ...
4  Abbey Level Crossing (Staffordshire)      ...

[5 rows x 12 columns]

>>> # To get data of line names
>>> line_names_data = ld.LineNames.fetch_line_names()

>>> type(line_names_data)
<class 'dict'>
>>> print(list(line_names_data.keys()))
['Line names', 'Last updated date']
>>> line_names_dat = line_names_data['Line names']
>>> print(line_names_dat.head())
            Line name  ... Route_note
```

(continues on next page)

(continued from previous page)

```
0           Abbey Line  ...       None
1        Airedale Line  ...       None
2          Argyle Line  ...       None
3     Arun Valley Line  ...       None
4  Atlantic Coast Line  ...       None

[5 rows x 3 columns]
```

Methods

update([confirmation_required, verbose, ...])	Update local backup of the line data.

LineData.update

LineData.**update**(*confirmation_required=True*, *verbose=False*, *time_gap=2*, *init_update=False*)

Update local backup of the line data.

Parameters

- **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
- **time_gap** (*int*) – time gap (in seconds) between the updating of different classes
- **init_update** (*bool*) – whether to update the data for instantiation of each subclass, defaults to False

Example:

```
>>> from pyrcs import LineData

>>> ld = LineData()
>>> ld.update(verbose=True)
```

OtherAssets

class pyrcs.collector.**OtherAssets**(*update=False*, *verbose=True*)

A class representation of all modules of the subpackage *pyrcs.other_assets* for collecting other assets.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `True`

Examples:

```
>>> from pyrcs import OtherAssets

>>> oa = OtherAssets()

>>> # To get data of railway stations
>>> railway_station_data = oa.Stations.fetch_station_data()

>>> type(railway_station_data)
<class 'dict'>
>>> print(list(railway_station_data.keys()))
['Railway station data', 'Last updated date']
>>> railway_station_dat = railway_station_data['Railway station data']
>>> print(railway_station_dat.head())
      Station   ELR   Mileage  ... Prev_Date_6 Prev_Operator_7  Prev_Date_7
0  Abbey Wood   NKL  11m 43ch  ...         NaN             NaN          NaN
1  Abbey Wood  XRS3  24.458km  ...         NaN             NaN          NaN
2        Aber   CAR   8m 69ch  ...         NaN             NaN          NaN
3   Abercynon   CAM  16m 28ch  ...         NaN             NaN          NaN
4   Abercynon   ABD  16m 28ch  ...         NaN             NaN          NaN

[5 rows x 25 columns]

>>> # To get data of signal boxes
>>> signal_boxes_data = oa.SignalBoxes.fetch_prefix_codes()
>>> type(signal_boxes_data)
<class 'dict'>
>>> print(list(signal_boxes_data.keys()))
['Signal boxes', 'Last updated date']
>>> signal_boxes_dat = signal_boxes_data['Signal boxes']
>>> print(signal_boxes_dat.head())
  Code               Signal Box  ...            Closed        Control to
0   AF  Abbey Foregate Junction  ...
1   AJ           Abbey Junction  ...  16 February 1992     Nuneaton (NN)
2    R           Abbey Junction  ...  16 February 1992     Nuneaton (NN)
3   AW               Abbey Wood  ...      13 July 1975      Dartford (D)
4   AE         Abbey Works East  ...   1 November 1987  Port Talbot (PT)

[5 rows x 8 columns]
```

Methods

update([confirmation_required, verbose, ...])	Update local backup of the other assets data.

OtherAssets.update

OtherAssets.**update**(*confirmation_required=True*, *verbose=False*, *time_gap=2*, *init_update=False*)

Update local backup of the other assets data.

Parameters

- **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`
- **time_gap** (*int*) – time gap (in seconds) between the updating of different classes
- **init_update** (*bool*) – whether to update the data for instantiation of each subclass, defaults to `False`

Example:

```
>>> from pyrcs.collector import OtherAssets

>>> oa = OtherAssets()
>>> oa.update(verbose=True)
```

3.2.2 updater

Update package data.

Local backup

update_backup_data([verbose, time_gap])	Update data of the package's local backup.

update_backup_data

pyrcs.updater.**update_backup_data**(*verbose=False*, *time_gap=2*)

Update data of the package's local backup.

Parameters

- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`
- **time_gap** (*int*) – time gap (in seconds) between the updating of different classes

Example:

```
>>> from pyrcs.updater import update_backup_data

>>> update_backup_data(verbose=True)
```

3.2.3 utils

Utilities - Helper functions.

Specification of resource homepage

homepage_url()	Specify the homepage URL of the data source.

homepage_url

pyrcs.utils.**homepage_url**()

Specify the homepage URL of the data source.

Returns URL of the data source homepage

Return type str

Data converters

mile_chain_to_nr_mileage(miles_chains)	Convert mileage data in the form '<miles>.<chains>' to Network Rail mileage.
nr_mileage_to_mile_chain(str_mileage)	Convert Network Rail mileage to the form '<miles>.<chains>'.
nr_mileage_str_to_num(str_mileage)	Convert string-type Network Rail mileage to numerical-type one.
nr_mileage_num_to_str(num_mileage)	Convert numerical-type Network Rail mileage to string-type one.

continues on next page

Table 34 – continued from previous page

nr_mileage_to_yards(nr_mileage)	Convert Network Rail mileages to yards.
yards_to_nr_mileage(yards)	Convert yards to Network Rail mileages.
shift_num_nr_mileage(nr_mileage, shift_yards)	Shift Network Rail mileage by given yards.
year_to_financial_year(date)	Convert calendar year of a given date to Network Rail financial year.

mile_chain_to_nr_mileage

pyrcs.utils.**mile_chain_to_nr_mileage**(*miles_chains*)

Convert mileage data in the form '<miles>.<chains>' to Network Rail mileage.

Parameters **miles_chains** (*str or numpy.nan or None*) – mileage data presented in the form '<miles>.<chains>'

Returns Network Rail mileage in the form '<miles>.<yards>'

Return type str

Examples:

```
>>> from pyrcs.utils import mile_chain_to_nr_mileage

>>> miles_chains_dat = '0.18'  # AAM 0.18 Tewkesbury Junction with ANZ (84.62)
>>> mileage_data = mile_chain_to_nr_mileage(miles_chains_dat)
>>> print(mileage_data)
0.0396

>>> miles_chains_dat = None  # or np.nan, or ''
>>> mileage_data = mile_chain_to_nr_mileage(miles_chains_dat)
>>> print(mileage_data)
```

nr_mileage_to_mile_chain

pyrcs.utils.**nr_mileage_to_mile_chain**(*str_mileage*)

Convert Network Rail mileage to the form '<miles>.<chains>'.

Parameters **str_mileage** (*str or numpy.nan or None*) – Network Rail mileage data presented in the form '<miles>.<yards>'

Returns '<miles>.<chains>'

Return type str

Examples:

```
>>> from pyrcs.utils import nr_mileage_to_mile_chain

>>> str_mileage_dat = '0.0396'
>>> miles_chains_dat = nr_mileage_to_mile_chain(str_mileage_dat)
```

(continues on next page)

(continued from previous page)

```
>>> print(miles_chains_dat)
0.18

>>> str_mileage_dat = None  # or np.nan, or ''
>>> miles_chains_dat = nr_mileage_to_mile_chain(str_mileage_dat)
>>> print(miles_chains_dat)
```

nr_mileage_str_to_num

pyrcs.utils.**nr_mileage_str_to_num**(*str_mileage*)

Convert string-type Network Rail mileage to numerical-type one.

Parameters **str_mileage** (*str*) – string-type Network Rail mileage in the form '<miles>.<yards>'

Returns numerical-type Network Rail mileage

Return type float

Examples:

```
>>> from pyrcs.utils import nr_mileage_str_to_num

>>> str_mileage_dat = '0.0396'
>>> num_mileage_dat = nr_mileage_str_to_num(str_mileage_dat)
>>> print(num_mileage_dat)
0.0396

>>> str_mileage_dat = ''
>>> num_mileage_dat = nr_mileage_str_to_num(str_mileage_dat)
>>> print(num_mileage_dat)
nan
```

nr_mileage_num_to_str

pyrcs.utils.**nr_mileage_num_to_str**(*num_mileage*)

Convert numerical-type Network Rail mileage to string-type one.

Parameters **num_mileage** (*float*) – numerical-type Network Rail mileage

Returns string-type Network Rail mileage in the form '<miles>.<yards>'

Return type str

Examples:

```
>>> import numpy as np_
>>> from pyrcs.utils import nr_mileage_num_to_str

>>> num_mileage_dat = 0.0396
```

(continues on next page)

(continued from previous page)

```
>>> str_mileage_dat = nr_mileage_num_to_str(num_mileage_dat)
>>> print(str_mileage_dat)
0.0396
>>> type(str_mileage_dat)
<class 'str'>

>>> num_mileage_dat = np_.nan
>>> str_mileage_dat = nr_mileage_num_to_str(num_mileage_dat)
>>> print(str_mileage_dat)

>>> type(str_mileage_dat)
<class 'str'>
```

nr_mileage_to_yards

pyrcs.utils.**nr_mileage_to_yards**(*nr_mileage*)

Convert Network Rail mileages to yards.

Parameters **nr_mileage** (*float or str*) – Network Rail mileage

Returns yards

Return type int

Examples:

```
>>> from pyrcs.utils import nr_mileage_to_yards

>>> nr_mileage_dat = '0.0396'
>>> yards_dat = nr_mileage_to_yards(nr_mileage_dat)
>>> print(yards_dat)
396

>>> nr_mileage_dat = 0.0396
>>> yards_dat = nr_mileage_to_yards(nr_mileage_dat)
>>> print(yards_dat)
396
```

yards_to_nr_mileage

pyrcs.utils.**yards_to_nr_mileage**(*yards*)

Convert yards to Network Rail mileages.

Parameters **yards** (*int or float or numpy.nan or None*) – yards

Returns Network Rail mileage in the form '<miles>.<yards>'

Return type str

Examples:

```
>>> from pyrcs.utils import yards_to_nr_mileage

>>> yards_dat = 396
>>> mileage_dat = yards_to_nr_mileage(yards_dat)
>>> print(mileage_dat)
0.0396
>>> type(mileage_dat)
<class 'str'>

>>> yards_dat = 396.0
>>> mileage_dat = yards_to_nr_mileage(yards_dat)
>>> print(mileage_dat)
0.0396
>>> type(mileage_dat)
<class 'str'>

>>> yards_dat = None
>>> mileage_dat = yards_to_nr_mileage(yards_dat)
>>> print(mileage_dat)

>>> type(mileage_dat)
<class 'str'>
```

shift_num_nr_mileage

pyrcs.utils.**shift_num_nr_mileage**(*nr_mileage*, *shift_yards*)

Shift Network Rail mileage by given yards.

Parameters

- **nr_mileage** (*float or int or str*) – Network Rail mileage
- **shift_yards** (*int or float*) – yards by which the given nr_mileage is shifted

Returns shifted numerical Network Rail mileage

Return type float

Examples:

```
>>> from pyrcs.utils import shift_num_nr_mileage

>>> num_mileage_dat = shift_num_nr_mileage(nr_mileage='0.0396', shift_yards=220)
>>> print(num_mileage_dat)
0.0616

>>> shift_num_nr_mileage(nr_mileage='0.0396', shift_yards=220.99)
>>> print(num_mileage_dat)
0.0617

>>> shift_num_nr_mileage(nr_mileage=10, shift_yards=220)
```

(continues on next page)

(continued from previous page)

```
>>> print(num_mileage_dat)
10.022
```

year_to_financial_year

pyrcs.utils.**year_to_financial_year**(*date*)

Convert calendar year of a given date to Network Rail financial year.

Parameters **date** (*datetime.datetime*) – date

Returns Network Rail financial year of the given date

Return type int

Example:

```
>>> import datetime
>>> from pyrcs.utils import year_to_financial_year

>>> financial_year = year_to_financial_year(datetime.datetime.now())
>>> print(financial_year)
2020
```

Data parsers

parse_tr(header, trs)	Parse a list of parsed HTML <tr> elements.
parse_table(source[, parser])	Parse HTML <tr> elements for creating a data frame.
parse_location_name(location_name)	Parse location name (and its associated note).
parse_date(str_date[, as_date_type])	Parse a date.

parse_tr

pyrcs.utils.**parse_tr**(*header*, *trs*)

Parse a list of parsed HTML <tr> elements.

See also [PT-1].

Parameters

- **header** (*list*) – list of column names of a requested table
- **trs** (*bs4.ResultSet*) – contents under <tr> tags (bs4.Tag) of a web page

Returns list of lists with each comprising a row of the requested table

Return type list

Example:

```
>>> import bs4
>>> import requests
>>> from pyrcs.utils import fake_requests_headers, parse_tr

>>> source = requests.get('http://www.railwaycodes.org.uk/elrs/elra.shtm',
...                       headers=fake_requests_headers())
>>> parsed_text = bs4.BeautifulSoup(source.text, 'lxml')
>>> header_ = []
>>> for th in parsed_text.find_all('th'):
...     header_.append(th.text)
>>> trs_dat = parsed_text.find_all('tr')

>>> tables_list = parse_tr(header_, trs_dat)  # returns a list of lists
>>> type(tables_list)
<class 'list'>
>>> print(tables_list[-1])
['AYT', 'Aberystwyth Branch', '0.00 - 41.15', 'Pencader Junction', ' ']
```

parse_table

pyrcs.utils.**parse_table**(*source*, *parser='lxml'*)

Parse HTML <tr> elements for creating a data frame.

Parameters

- **source** (*requests.Response*) – response object to connecting a URL to request a table
- **parser** (*str*) – `'lxml'` (default), `'html5lib'` or `'html.parser'`

Returns a list of lists each comprising a row of the requested table (see also *parse_tr()*) and a list of column names of the requested table

Return type tuple

Examples:

```
>>> from pyrcs.utils import fake_requests_headers, parse_table

>>> source_ = requests.get('http://www.railwaycodes.org.uk/elrs/elra.shtm',
...                        headers=fake_requests_headers())

>>> parsed_contents = parse_table(source_, parser='lxml')
>>> type(parsed_contents)
<class 'tuple'>
>>> type(parsed_contents[0])
<class 'list'>
>>> type(parsed_contents[1])
<class 'list'>
```

parse_location_name

pyrcs.utils.**parse_location_name**(*location_name*)

Parse location name (and its associated note).

Parameters **location_name** (*str or None*) – location name (in raw data)

Returns location name and, if any, note

Return type tuple

Examples:

```
>>> from pyrcs.utils import parse_location_name

>>> location_dat = 'Abbey Wood'
>>> dat_and_note = parse_location_name(location_dat)
>>> print(dat_and_note)
('Abbey Wood', '')

>>> location_dat = None
>>> dat_and_note = parse_location_name(location_dat)
>>> print(dat_and_note)
('', '')

>>> location_dat = 'Abercynon (formerly Abercynon South)'
>>> dat_and_note = parse_location_name(location_dat)
>>> print(dat_and_note)
('Abercynon', 'formerly Abercynon South')

>>> location_dat = 'Allerton (reopened as Liverpool South Parkway)'
>>> dat_and_note = parse_location_name(location_dat)
>>> print(dat_and_note)
('Allerton', 'reopened as Liverpool South Parkway')

>>> location_dat = 'Ashford International [domestic portion]'
>>> dat_and_note = parse_location_name(location_dat)
>>> print(dat_and_note)
('Ashford International', 'domestic portion')
```

parse_date

pyrcs.utils.**parse_date**(*str_date*, *as_date_type=False*)

Parse a date.

Parameters

- **str_date** (*str*) – string-type date
- **as_date_type** (*bool*) – whether to return the date as datetime.date, defaults to False

Returns parsed date as a string or datetime.date

Return type str or datetime.date

Examples:

```
>>> from pyrcs.utils import parse_date

>>> str_date_dat = '2020-01-01'

>>> parsed_date_dat = parse_date(str_date_dat, as_date_type=True)
>>> print(parsed_date_dat)
2020-01-01
>>> type(parsed_date_dat)
<class 'datetime.date'>
```

Retrieval of useful information

get_site_map([update, ...])	Fetch the site map from the package data.
get_last_updated_date(url[, parsed, ...])	Get last update date.
get_catalogue(page_url[, update, ...])	Get the catalogue for a class.
get_category_menu(menu_url[, update, ...])	Get a menu of the available classes.

get_site_map

pyrcs.utils.**get_site_map**(*update=False*, *confirmation_required=True*, *verbose=False*)

Fetch the site map from the package data.

Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns dictionary of site map data

Return type dict or None

Examples:

```
>>> from pyrcs.utils import get_site_map

>>> site_map_dat = get_site_map()

>>> type(site_map_dat)
```

(continues on next page)

(continued from previous page)

```
<class 'dict'>
>>> print(list(site_map_dat.keys()))
['Home', 'Line data', 'Other assets', '"Legal/financial" lists', 'Miscellaneous']
>>> print(site_map_dat['Home'])
http://www.railwaycodes.org.uk/index.shtml

>>> site_map_dat = get_site_map(update=True, verbose=2)
```

get_last_updated_date

pyrcs.utils.**get_last_updated_date**(*url*, *parsed=True*, *as_date_type=False*, *verbose=False*)

Get last update date.

Parameters

- **url** (*str*) – URL link of a requested web page
- **parsed** (*bool*) – whether to reformat the date, defaults to `True`
- **as_date_type** (*bool*) – whether to return the date as datetime.date, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns date of when the specified web page was last updated

Return type str or datetime.date or None

Examples:

```
>>> from pyrcs.utils import get_last_updated_date

>>> last_upd_date = get_last_updated_date(
...     url='http://www.railwaycodes.org.uk/crs/CRSa.shtm', parsed=True,
...     as_date_type=False)
>>> type(last_upd_date)
<class 'str'>

>>> last_upd_date = get_last_updated_date(
...     url='http://www.railwaycodes.org.uk/crs/CRSa.shtm', parsed=True,
...     as_date_type=True)
>>> type(last_upd_date)
<class 'datetime.date'>

>>> last_upd_date = get_last_updated_date(
...     url='http://www.railwaycodes.org.uk/linedatamenu.shtm')
>>> print(last_upd_date)
None
```

get_catalogue

pyrcs.utils.**get_catalogue**(*page_url*, *update=False*, *confirmation_required=True*, *json_it=True*, *verbose=False*)

Get the catalogue for a class.

Parameters

- **page_url** (*str*) – URL of the main page of a code category
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **json_it** (*bool*) – whether to save the catalogue as a .json file, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns catalogue in the form {'<title>': '<URL>'}

Return type dict or None

Examples:

```
>>> from pyrcs.utils import get_catalogue

>>> url = 'http://www.railwaycodes.org.uk/elrs/elr0.shtm'
>>> catalog = get_catalogue(url)
>>> type(catalog)
<class 'dict'>
>>> print(list(catalog.keys())[:5])
['Introduction', 'A', 'B', 'C', 'D']

>>> url = 'http://www.railwaycodes.org.uk/linedatamenu.shtm'
>>> catalog = get_catalogue(url)
>>> print(list(catalog.keys())[:5])
['Line data']

>>> line_data_catalog = catalog['Line data']
>>> type(line_data_catalog)
<class 'dict'>
```

get_category_menu

pyrcs.utils.**get_category_menu**(*menu_url*, *update=False*, *confirmation_required=True*, *json_it=True*, *verbose=False*)

Get a menu of the available classes.

Parameters

- **menu_url** (*str*) – URL of the menu page
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **json_it** (*bool*) – whether to save the catalogue as a .json file, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns a category menu

Return type dict or None

Example:

```
>>> from pyrcs.utils import get_category_menu

>>> url = 'http://www.railwaycodes.org.uk/linedatamenu.shtm'
>>> menu = get_category_menu(url)

>>> type(menu)
<class 'dict'>
>>> print(list(menu.keys()))
['Line data']
```

Rectification of location names

fetch_loc_names_repl_dict([k, regex, ...])	Create a dictionary for rectifying location names.
update_loc_names_repl_dict(new_items, regex)	Update the location-names replacement dictionary in the package data.

fetch_loc_names_repl_dict

pyrcs.utils.**fetch_loc_names_repl_dict**(*k=None*, *regex=False*, *as_dataframe=False*)

Create a dictionary for rectifying location names.

Parameters

- **k** (*str or int or float or bool or None*) – key of the created dictionary, defaults to `None`
- **regex** (*bool*) – whether to create a dictionary for replacement based on regular expressions, defaults to `False`
- **as_dataframe** (*bool*) – whether to return the created dictionary as a pandas.DataFrame, defaults to `False`

Returns dictionary for rectifying location names

Return type dict or pandas.DataFrame

Examples:

```
>>> from pyrcs.utils import fetch_loc_names_repl_dict

>>> repl_dict = fetch_loc_names_repl_dict()
>>> type(repl_dict)
<class 'dict'>
>>> print(list(repl_dict.keys())[:5])
['"Tyndrum Upper" (Upper Tyndrum)',
 'AISH EMERGENCY CROSSOVER',
 'ATLBRJN',
 'Aberdeen Craiginches',
 'Aberdeen Craiginches T.C.']

>>> repl_dict = fetch_loc_names_repl_dict(regex=True, as_dataframe=True)
>>> type(repl_dict)
<class 'pandas.core.frame.DataFrame'>
>>> print(repl_dict.head())
                              new_value
re.compile(' \(DC lines\)')   [DC lines]
re.compile(' And | \+ ')              &
re.compile('-By-')                  -by-
re.compile('-In-')                  -in-
re.compile('-En-Le-')            -en-le-
```

update_loc_names_repl_dict

pyrcs.utils.**update_loc_names_repl_dict**(*new_items*, *regex*, *verbose=False*)

Update the location-names replacement dictionary in the package data.

Parameters

- **new_items** (*dict*) – new items to replace
- **regex** (*bool*) – whether this update is for regular-expression dictionary
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Example:

```
>>> from pyrcs.utils import update_loc_names_repl_dict
```

```
>>> new_items_ = {}
>>> update_loc_names_repl_dict(new_items_, regex=False)
```

Data fixers

fix_num_stanox(stanox_code)	Fix 'STANOX' if it is loaded as numbers.
fix_nr_mileage_str(nr_mileage)	Fix Network Rail mileage.

fix_num_stanox

pyrcs.utils.**fix_num_stanox**(*stanox_code*)

Fix 'STANOX' if it is loaded as numbers.

Parameters **stanox_code** (*str or int*) – STANOX code

Returns standard STANOX code

Return type str

Examples:

```
>>> from pyrcs.utils import fix_num_stanox

>>> stanox = 65630
>>> stanox_ = fix_num_stanox(stanox)
>>> type(stanox_)
<class 'str'>

>>> stanox = 2071
>>> stanox_ = fix_num_stanox(stanox)
>>> print(stanox_)
02071
```

fix_nr_mileage_str

pyrcs.utils.**fix_nr_mileage_str**(*nr_mileage*)

Fix Network Rail mileage.

Parameters **nr_mileage** (*str or float*) – NR mileage

Returns conventional NR mileage code

Return type str

Examples:

```
>>> from pyrcs.utils import fix_nr_mileage_str

>>> mileage = 29.011
>>> mileage_ = fix_nr_mileage_str(mileage)
>>> print(mileage_)
29.0110

>>> mileage = '.1100'
>>> mileage_ = fix_nr_mileage_str(mileage)
>>> print(mileage_)
0.1100
```

Miscellaneous utilities

print_connection_error([verbose])	Print a message about unsuccessful attempts to establish a connection to the Internet.
print_conn_err([update, verbose])	Print a message about unsuccessful attempts to establish a connection to the Internet for an instance of a class.
is_str_float(str_val)	Check if a string-type variable can express a float-type value.
is_internet_connected()	Check the Internet connection.

print_connection_error

pyrcs.utils.**print_connection_error**(*verbose=False*)

Print a message about unsuccessful attempts to establish a connection to the Internet.

Parameters **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

print_conn_err

pyrcs.utils.**print_conn_err**(*update=False*, *verbose=False*)

Print a message about unsuccessful attempts to establish a connection to the Internet for an instance of a class.

Parameters

- **update** (*bool*) – defaults to `False` (mostly complies with `update` in a parent function that uses this function)
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

is_str_float

pyrcs.utils.**is_str_float**(*str_val*)

Check if a string-type variable can express a float-type value.

Parameters **str_val** (*str*) – a string-type variable

Returns whether `str_val` can express a float value

Return type bool

Examples:

```
>>> from pyrcs.utils import is_str_float

>>> is_str_float('')
False

>>> is_str_float('a')
False

>>> is_str_float('1')
True

>>> is_str_float('1.1')
True
```

is_internet_connected

pyrcs.utils.**is_internet_connected**()

Check the Internet connection.

Returns whether the machine is currently connected to the Internet

Return type bool

Examples:

```
>>> from pyrcs.utils import is_internet_connected

>>> is_internet_connected()
```

CHAPTER

FOUR

LICENSE

PyRCS is licensed under GNU General Public License v3 (GPLv3).

CHAPTER

FIVE

USE OF DATA

For the use of the data collected from this package, please refer to this link: http://www.railwaycodes.org.uk/misc/contributing.shtm

CHAPTER

SIX

ACKNOWLEDGEMENT

The development of the PyRCS is mainly built on data from the *Railway Codes* website. The author of the package would like to thank the website editor and all contributors to the data resources.

PYTHON MODULE INDEX

p

pyrcs, 10
pyrcs.collector, 90
pyrcs.line_data, 11
pyrcs.line_data.elec, 19
pyrcs.line_data.elr_mileage, 11
pyrcs.line_data.line_name, 45
pyrcs.line_data.loc_id, 29
pyrcs.line_data.lor_code, 39
pyrcs.line_data.trk_diagr, 48
pyrcs.other_assets, 51
pyrcs.other_assets.depot, 69
pyrcs.other_assets.feature, 79
pyrcs.other_assets.sig_box, 52
pyrcs.other_assets.station, 65
pyrcs.other_assets.tunnel, 58
pyrcs.other_assets.viaduct, 61
pyrcs.updater, 94
pyrcs.utils, 95

INDEX

A

amendment_to_loc_names() (*loc_id.LocationIdentifiers static method*), 31

C

collect_1950_system_codes() (*depot.Depots method*), 71
collect_buzzer_codes() (*feature.Features method*), 82
collect_elr_by_initial() (*elr_mileage.ELRMileages method*), 13
collect_elr_lor_converter() (*lor_code.LOR method*), 40
collect_etz_codes() (*elec.Electrification method*), 21
collect_explanatory_note() (*loc_id.LocationIdentifiers method*), 31
collect_four_digit_pre_tops_codes() (*depot.Depots method*), 72
collect_gwr_codes() (*depot.Depots method*), 73
collect_habds_and_wilds() (*feature.Features method*), 82
collect_indep_lines_codes() (*elec.Electrification method*), 22
collect_lengths_by_page() (*tunnel.Tunnels method*), 59
collect_line_names() (*line_name.LineNames method*), 47
collect_loc_codes_by_initial() (*loc_id.LocationIdentifiers method*), 32
collect_lor_codes_by_prefix() (*lor_code.LOR method*), 41
collect_mileage_file() (*elr_mileage.ELRMileages method*), 14
collect_national_network_codes() (*elec.Electrification method*), 23
collect_non_national_rail_codes() (*sig_box.SignalBoxes method*), 54
collect_ohns_codes() (*elec.Electrification method*), 23
collect_other_systems_codes() (*loc_id.LocationIdentifiers method*), 33
collect_prefix_codes() (*sig_box.SignalBoxes method*), 55
collect_sample_catalogue() (*trk_diagr.TrackDiagrams method*), 49
collect_station_data_by_initial() (*station.Stations method*), 67
collect_telegraph_codes() (*feature.Features method*), 84
collect_two_char_tops_codes() (*depot.Depots method*), 74
collect_viaduct_codes_by_page() (*viaduct.Viaducts method*), 63
collect_water_troughs() (*feature.Features method*), 85

D

Depots (*class in depot*), 70

E

Electrification (*class in elec*), 19
ELRMileages (*class in elr_mileage*), 12
extended_info() (*station.Stations method*), 68

F

Features (*class in feature*), 80
fetch_1950_system_codes() (*depot.Depots method*), 75
fetch_buzzer_codes() (*feature.Features method*), 85
fetch_depot_codes() (*depot.Depots method*), 76
fetch_elec_codes() (*elec.Electrification method*), 24
fetch_elr() (*elr_mileage.ELRMileages method*), 15
fetch_elr_lor_converter() (*lor_code.LOR method*), 42
fetch_etz_codes() (*elec.Electrification method*), 25
fetch_explanatory_note() (*loc_id.LocationIdentifiers method*), 34
fetch_features_codes() (*feature.Features method*), 86
fetch_four_digit_pre_tops_codes() (*depot.Depots method*), 77
fetch_gwr_codes() (*depot.Depots method*), 78
fetch_habds_and_wilds() (*feature.Features method*), 87
fetch_indep_lines_codes() (*elec.Electrification method*), 26
fetch_line_names() (*line_name.LineNames method*), 47
fetch_loc_names_repl_dict() (*in module pyrcs.utils*), 107
fetch_location_codes() (*loc_id.LocationIdentifiers method*), 35
fetch_lor_codes() (*lor_code.LOR method*), 43
fetch_mileage_file() (*elr_mileage.ELRMileages method*), 16
fetch_national_network_codes() (*elec.Electrification method*), 27
fetch_non_national_rail_codes() (*sig_box.SignalBoxes method*), 56
fetch_ohns_codes() (*elec.Electrification method*), 28
fetch_other_systems_codes() (*loc_id.LocationIdentifiers method*), 36
fetch_prefix_codes() (*sig_box.SignalBoxes method*), 57
fetch_sample_catalogue() (*trk_diagr.TrackDiagrams method*), 50
fetch_station_data() (*station.Stations method*), 68
fetch_telegraph_codes() (*feature.Features method*), 88
fetch_tunnel_lengths() (*tunnel.Tunnels method*), 60
fetch_two_char_tops_codes() (*depot.Depots method*), 79
fetch_viaduct_codes() (*viaduct.Viaducts method*), 64
fetch_water_troughs() (*feature.Features method*), 89
fix_nr_mileage_str() (*in module pyrcs.utils*), 109

fix_num_stanox() (*in module pyrcs.utils*), 108

G

get_catalogue() (*in module pyrcs.utils*), 105
get_category_menu() (*in module pyrcs.utils*), 106
get_conn_mileages() (*elr_mileage.ELRMileages method*), 17
get_indep_line_names() (*elec.Electrification method*), 29
get_keys_to_prefixes() (*lor_code.LOR method*), 44
get_last_updated_date() (*in module pyrcs.utils*), 104
get_lor_page_urls() (*lor_code.LOR method*), 45
get_site_map() (*in module pyrcs.utils*), 103
get_station_data_catalogue() (*station.Stations method*), 69
get_track_diagrams_items() (*trk_diagr.TrackDiagrams method*), 51

H

homepage_url() (*in module pyrcs.utils*), 95

I

is_internet_connected() (*in module pyrcs.utils*), 110
is_str_float() (*in module pyrcs.utils*), 110

L

LineData (*class in pyrcs.collector*), 91
LineNames (*class in line_name*), 46
LocationIdentifiers (*class in loc_id*), 29
LOR (*class in lor_code*), 39

M

make_loc_id_dict() (*loc_id.LocationIdentifiers method*), 37
mile_chain_to_nr_mileage() (*in module pyrcs.utils*), 96
module
    pyrcs, 10
    pyrcs.collector, 90
    pyrcs.line_data, 11
    pyrcs.line_data.elec, 19
    pyrcs.line_data.elr_mileage, 11
    pyrcs.line_data.line_name, 45
    pyrcs.line_data.loc_id, 29
    pyrcs.line_data.lor_code, 39
    pyrcs.line_data.trk_diagr, 48
    pyrcs.other_assets, 51
    pyrcs.other_assets.depot, 69
    pyrcs.other_assets.feature, 79
    pyrcs.other_assets.sig_box, 52
    pyrcs.other_assets.station, 65
    pyrcs.other_assets.tunnel, 58
    pyrcs.other_assets.viaduct, 61
    pyrcs.updater, 94
    pyrcs.utils, 95

N

nr_mileage_num_to_str() (*in module pyrcs.utils*), 97
nr_mileage_str_to_num() (*in module pyrcs.utils*), 97
nr_mileage_to_mile_chain() (*in module pyrcs.utils*), 96
nr_mileage_to_yards() (*in module pyrcs.utils*), 98

O

OtherAssets (*class in pyrcs.collector*), 93

P

parse_date() (*in module pyrcs.utils*), 102
parse_length() (*tunnel.Tunnels static method*), 61
parse_location_name() (*in module pyrcs.utils*), 102
parse_note_page() (*loc_id.LocationIdentifiers static method*), 38
parse_table() (*in module pyrcs.utils*), 101
parse_tr() (*in module pyrcs.utils*), 100
print_conn_err() (*in module pyrcs.utils*), 110
print_connection_error() (*in module pyrcs.utils*), 109
pyrcs
    module, 10
pyrcs.collector
    module, 90
pyrcs.line_data
    module, 11
pyrcs.line_data.elec
    module, 19
pyrcs.line_data.elr_mileage
    module, 11
pyrcs.line_data.line_name
    module, 45
pyrcs.line_data.loc_id
    module, 29
pyrcs.line_data.lor_code
    module, 39
pyrcs.line_data.trk_diagr
    module, 48
pyrcs.other_assets
    module, 51
pyrcs.other_assets.depot
    module, 69
pyrcs.other_assets.feature
    module, 79
pyrcs.other_assets.sig_box
    module, 52
pyrcs.other_assets.station
    module, 65
pyrcs.other_assets.tunnel
    module, 58
pyrcs.other_assets.viaduct
    module, 61
pyrcs.updater
    module, 94
pyrcs.utils
    module, 95

S

search_conn() (*elr_mileage.ELRMileages static method*), 18
shift_num_nr_mileage() (*in module pyrcs.utils*), 99
SignalBoxes (*class in sig_box*), 52
Stations (*class in station*), 65

T

TrackDiagrams (*class in trk_diagr*), 48

Tunnels (*class in tunnel*), 58

U

update() (*pyrcs.collector.LineData method*), 92
update() (*pyrcs.collector.OtherAssets method*), 94
update_backup_data() (*in module pyrcs.updater*), 95
update_loc_names_repl_dict() (*in module pyrcs.utils*), 108

V

Viaducts (*class in viaduct*), 62

Y

yards_to_nr_mileage() (*in module pyrcs.utils*), 98
year_to_financial_year() (*in module pyrcs.utils*), 100