PyRCS Documentation

Release 0.2.10

Qian Fu

Mar 23, 2021

CONTENTS

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: 0.2.10
```

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)
<class 'dict'>
>>> print(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.

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)
<class 'dict'>
>>> print(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.

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)
<class '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.

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)
<class 'dict'>
>>> print(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.

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

The keys of `em_amm` include:

- `'ELR'`
- `'Line'`
- `'Sub-Line'`
- `'AAM'`
- `'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['AAM']`: a pandas.DataFrame of the mileage file data;
- `em_amm['Notes']`: additional information/notes (if any).

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_railway_station_data_by_initial()*:

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

>>> type(stn_data_a)
<class 'dict'>
>>> print(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.

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

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

(continues on next page)

(continued from previous page)

```
>>> type(stn_data)
<class 'dict'>
>>> print(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.

(The end of the quick start)

For more details and examples, check *Sub-packages and modules*.

CHAPTER

THREE

SUB-PACKAGES AND MODULES

3.1 Sub-packages

3.1.1 line_data

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

elr_mileage

Collect Engineer's Line References (ELRs) codes.

Class

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

ELRMileages

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

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

Example:

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

>>> em = ELRMileages()

>>> print(em.Name)
```

(continues on next page)

(continued from previous page)

```
ELRs and mileages

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

Methods

ELRMileages.cdd_em(*sub_dir[, mkdir])	Change directory to package data directory and sub-directories (and/or a file).
ELRMileages.collect_elr_by_initial(initial)	Collect Engineer's Line References (ELRs) for the given initial letter from source web page.
ELRMileages.collect_mileage_file(elr[, ...])	Collect mileage file for the given ELR from source web page.
ELRMileages.fetch_elr([update, pickle_it, ...])	Fetch ELRs and mileages from local backup.
ELRMileages.fetch_mileage_file(elr[, ...])	Fetch mileage file for the given ELR from local backup.
ELRMileages.get_conn_mileages(start_elr, end_elr)	Get a connection point between two ELR-and-mileage pairs.
ELRMileages.parse_mileage_col(mileage)	Parse column of mileage data.
ELRMileages.parse_mileage_data(mileage_data)	Parse scraped data of mileage file.
ELRMileages.parse_multi_measures(mileage_data)	Process data of mileage file with multiple measures.
ELRMileages.parse_node_col(node)	Parse column of node data.
ELRMileages.search_conn(start_elr, start_em, ...)	Search for connection between two ELR-and-mileage pairs.

elr_mileage.ELRMileages.cdd_em

ELRMileages.**cdd_em**(**sub_dir*, *mkdir=False*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\line-data\elrs-and-mileages"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **mkdir** (*bool*) – whether to create a directory, defaults to `False`
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `ELRMileages`

Return type str

elr_mileage.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')

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

elr_mileage.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)
<class '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'].head())
   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
```

(continues on next page)

(continued from previous page)

```
[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'].head())
   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]
```

elr_mileage.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()
```

(continues on next page)

(continued from previous page)

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

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

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

[5 rows x 5 columns]
```

elr_mileage.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')

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

(continues on next page)

(continued from previous page)

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

elr_mileage.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 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 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')
>>> (start_dest_mileage_,
...  conn_elr_, conn_orig_mileage_, conn_dest_mileage_,
...  end_orig_mileage_) = conn

>>> print(start_dest_mileage_)
5.1606
>>> print(conn_elr_)
```

(continues on next page)

(continued from previous page)

```
NOL
>>> print(conn_orig_mileage_)
5.1606
>>> print(conn_dest_mileage_)
0.0638
>>> print(end_orig_mileage_)
123.1320
```

Example 2:

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

>>> em = ELRMileages()

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

elr_mileage.ELRMileages.parse_mileage_col

static ELRMileages.**parse_mileage_col**(*mileage*)

Parse column of mileage data.

Parameters **mileage** (*pandas.Series*) – column of mileage data

Returns parsed mileages

Return type pandas.DataFrame

elr_mileage.ELRMileages.parse_mileage_data

ELRMileages.**parse_mileage_data**(*mileage_data*)

Parse scraped data of mileage file.

Parameters **mileage_data** (*pandas.DataFrame*) – preprocessed data of mileage file scraped from source web page

Returns parsed data of mileage file

Return type pandas.DataFrame

elr_mileage.ELRMileages.parse_multi_measures

static ELRMileages.**parse_multi_measures**(*mileage_data*)

Process data of mileage file with multiple measures.

Parameters **mileage_data** – scraped raw mileage file from source web page

Type pandas.DataFrame

elr_mileage.ELRMileages.parse_node_col

static ELRMileages.**parse_node_col**(*node*)

Parse column of node data.

Parameters **node** (*pandas.Series*) – column of node data

Returns parsed nodes

Return type pandas.DataFrame

elr_mileage.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()

>>> start_elr_ = 'AAM'
>>> start_mileage_file_ = em.collect_mileage_file(
...     start_elr_, confirmation_required=False)
>>> start_mileage_data_ = start_mileage_file_['Mileage']

>>> end_elr_ = 'ANZ'
>>> end_mileage_file_ = em.collect_mileage_file(
...     end_elr_, confirmation_required=False)
```

(continues on next page)

(continued from previous page)

```
>>> end_mileage_data_ = end_mileage_file_['Mileage']

>>> start_dest_mileage_, end_orig_mileage_ = em.search_conn(
...     start_elr_, start_mileage_data_, end_elr_, end_mileage_data_)

>>> print(start_dest_mileage_)
0.0396
>>> print(end_orig_mileage_)
84.1364
```

elec

Collect codes of British railway overhead electrification installations.

Class

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

Electrification

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

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

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

Electrification.cdd_elec(*sub_dir, **kwargs)	Change directory to package data directory and sub-directories (and/or a file).
Electrification.collect_etz_codes([...])	Collect OLE section codes for national network energy tariff zones from source web page.
Electrification.collect_indep_lines_codes([...])	Collect OLE section codes for independent lines from source web page.
Electrification.collect_national_network_codes([...])	Collect OLE section codes for national network from source web page.
Electrification.collect_ohns_codes([...])	Collect codes for overhead line electrification neutral sections (OHNS) from source web page.
Electrification.fetch_elec_codes([update, ...])	Fetch OLE section codes in electrification catalogue.
Electrification.fetch_etz_codes([update, ...])	Fetch OLE section codes for national network energy tariff zones from source web page.
Electrification.fetch_indep_lines_codes([...])	Fetch OLE section codes for independent lines from local backup.
Electrification.fetch_national_network_codes([...])	Fetch OLE section codes for national network from local backup.

continues on next page

Table 4 – continued from previous page	
Electrification.fetch_ohns_codes([update, ...])	Fetch codes for overhead line electrification neutral sections (OHNS) from local backup.
Electrification.get_indep_line_names()	Get names of independent lines.

elec.Electrification.cdd_elec

Electrification.**cdd_elec**(**sub_dir*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\line-data\electrification"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `Electrification`

Return type str

elec.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)
```

(continues on next page)

(continued from previous page)

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

elec.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)
<class 'dict'>
>>> print(list(il_ole_dat.keys()))
['Independent lines', 'Last updated date']
```

elec.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)
<class 'dict'>
>>> print(list(nn_dat.keys()))
['National network', 'Last updated date']
```

elec.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)
<class 'dict'>
>>> print(list(ohns_dat.keys()))
['National network neutral sections', 'Last updated date']
```

elec.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()

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

elec.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()

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

elec.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()

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

elec.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()

>>> type(nn_ole_dat)
<class 'dict'>
>>> print(list(nn_ole_dat.keys()))
['National network', 'Last updated date']
```

elec.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()

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

elec.Electrification.get_indep_line_names

Electrification.**get_indep_line_names**()

Get names of independent lines.

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()

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

LocationIdentifiers

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

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

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

LocationIdentifiers.amendment_to_loc_names()	Create a replacement dictionary for location name amendments.
LocationIdentifiers.cdd_lc(*sub_dir, **kwargs)	Change directory to package data directory and sub-directories (and/or a file).

continues on next page

Table 6 – continued from previous page	
LocationIdentifiers.collect_explanatory_note([...])	Collect note about CRS code from source web page.
LocationIdentifiers.collect_loc_codes_by_initial(initial)	Collect CRS, NLC, TIPLOC, STANME and STANOX codes for a given `initial` letter.
LocationIdentifiers.collect_other_systems_codes([...])	Collect data of other systems' codes from source web page.
LocationIdentifiers.fetch_explanatory_note([...])	Fetch multiple station codes explanatory note from local backup.
LocationIdentifiers.fetch_location_codes([...])	Fetch CRS, NLC, TIPLOC, STANME and STANOX codes from local backup.
LocationIdentifiers.fetch_other_systems_codes([...])	Fetch data of other systems' codes from local backup.
LocationIdentifiers.make_loc_id_dict(keys[, ...])	Make a dict/dataframe for location code data for the given `keys`.
LocationIdentifiers.parse_note_page(note_url)	Parse addition note page.

loc_id.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(list(loc_name_amendment_dict.keys()))
['Location']
```

loc_id.LocationIdentifiers.cdd_lc

LocationIdentifiers.**cdd_lc**(**sub_dir*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\line-data\crs-nlc-tiploc-stanox"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `LocationIdentifiers`

Return type str

loc_id.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, None

Example:

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

>>> lid = LocationIdentifiers()

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

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

loc_id.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)
<class 'dict'>
>>> print(list(location_codes_a.keys()))
['A', 'Additional notes', 'Last updated date']
```

loc_id.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, None

Example:

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

>>> lid = LocationIdentifiers()

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

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

loc_id.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=False, pickle_it=False, data_dir=None, verbose=True)

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

loc_id.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()

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

loc_id.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()

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

loc_id.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, list*) – one or a sublist of ['CRS', 'NLC', 'TIPLOC', 'STANOX', 'STANME']
- **initials** (*str, list, 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, pandas.DataFrame, 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

>>> keys_ = ['STANOX', 'TIPLOC']
>>> initial_ = 'a'

>>> stanox_dictionary = lid.make_loc_id_dict(keys_, initial_)

>>> 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

>>> keys_ = ['STANOX', 'TIPLOC']
>>> initial_ = 'b'

>>> stanox_dictionary = lid.make_loc_id_dict(
...     keys_, initial_, as_dict=True, main_key='Data')

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

loc_id.LocationIdentifiers.parse_note_page

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

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'`

Returns parsed texts

Return type list

Example:

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

>>> lid = LocationIdentifiers()

>>> url = lid.HomeURL + '/crs/CRS2.shtm'

>>> parsed_note_ = lid.parse_note_page(url, parser='lxml')

>>> print(parsed_note_[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 PRIDE/LOR codes.

Class

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

LOR

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

A class for collecting PRIDE/LOR 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

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

LOR.cdd_lor(*sub_dir, **kwargs)	Change directory to package data directory and sub-directories (and/or a file).
LOR.collect_elr_lor_converter([. . .])	Collect ELR/LOR converter from source web page.
LOR.collect_lor_codes_by_prefix(prefix[, . . .])	Collect PRIDE/LOR codes by a given prefix.
LOR.fetch_elr_lor_converter([update, . . .])	Fetch ELR/LOR converter from local backup.
LOR.fetch_lor_codes([update, pickle_it, . . .])	Fetch PRIDE/LOR codes from local backup.

continues on next page

Table 8 – continued from previous page	
LOR.get_keys_to_prefixes([prefixes_only, ...])	Get key to PRIDE/LOR code prefixes.
LOR.get_lor_page_urls([update, verbose])	Get URLs to PRIDE/LOR codes with different prefixes.
LOR.update_catalogue([...])	Update catalogue data including keys to prefixes and LOR page URLs.

lor_code.LOR.cdd_lor

LOR.**cdd_lor**(**sub_dir*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\line-data\lor-codes"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `LOR`

Return type str

lor_code.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_converter_ = lor.collect_elr_lor_converter()
```

(continues on next page)

(continued from previous page)

```
To collect data of ELR/LOR converter? [No]|Yes: yes

>>> type(elr_lor_converter_)
<class 'dict'>
>>> print(elr_lor_converter_['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_code.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()

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

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

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

(continues on next page)

(continued from previous page)

```
>>> lor_codes_ea = lor.collect_lor_codes_by_prefix(prefix='EA')
>>> ea_dat = lor_codes_ea['EA']
>>> type(ea_dat)
<class 'dict'>
>>> 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  ...  Replaced by new EA1013 from 19 April 2014

[5 rows x 5 columns]
```

lor_code.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_converter_ = lor.fetch_elr_lor_converter()

>>> type(elr_lor_converter_)
<class 'dict'>
>>> for col in elr_lor_converter_['ELR/LOR converter'].columns:
...     print(col)
ELR
Miles from
Miles to
```

(continues on next page)

(continued from previous page)

```
LOR code
ELR_URL
LOR_URL
```

lor_code.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()

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

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

lor_code.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, dict

Examples:

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

>>> lor = LOR()

>>> keys_to_prefixes_ = lor.get_keys_to_prefixes()

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

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

>>> type(keys_to_prefixes_)
<class 'dict'>
>>> print(keys_to_prefixes_['Key to prefixes'].head())
  Prefixes                                  Name
0       CY                                 Wales
1       EA         South Eastern: East Anglia area
2       GW  Great Western (later known as Western)
3       LN                  London & North Eastern
4       MD       North West: former Midlands lines
```

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

lor_code.LOR.update_catalogue

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

Update catalogue data including keys to prefixes and LOR page URLs.

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`

line_name

Collect British railway line names.

Class

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

LineNames

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

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`

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

LineNames.cdd_ln(*sub_dir, **kwargs)	Change directory to package data directory and sub-directories (and/or a file).
LineNames.collect_line_names([...])	Collect data of railway line names from source web page.
LineNames.fetch_line_names([update, ...])	Fetch data of railway line names from local backup.

line_name.LineNames.cdd_ln

LineNames.**cdd_ln**(**sub_dir*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\line-data\line-names"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `LineNames`

Return type str

line_name.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)
<class 'dict'>
>>> print(list(line_names_dat.keys()))
['Line names', 'Last updated date']
```

line_name.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()

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

trk_diagr

Collect British railway track diagrams.

Class

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

TrackDiagrams

class trk_diagr.**TrackDiagrams**(*data_dir=None*, *update=False*)

A class for collecting British railway track diagrams.

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`

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

TrackDiagrams.cdd_td(*sub_dir, **kwargs)	Change directory to package data directory and sub-directories (and/or a file).
TrackDiagrams.collect_sample_catalogue([...])	Collect catalogue of sample railway track diagrams from source web page.
TrackDiagrams.fetch_sample_catalogue([...])	Fetch catalogue of sample railway track diagrams from local backup.

trk_diagr.TrackDiagrams.cdd_td

TrackDiagrams.**cdd_td**(**sub_dir*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\line-data\track-diagrams"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `LOR`

Return type str

trk_diagr.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)
<class 'dict'>
>>> print(list(track_diagrams_catalog.keys()))
['Track diagrams', 'Last updated date']
```

trk_diagr.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()

>>> track_diagrams_catalog = td.fetch_sample_catalogue()

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

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

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 PRIDE/LOR codes.
line_name	Collect British railway line names.
trk_diagr	Collect British railway track diagrams.

3.1.2 other_assets

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

sig_box

Collect signal box prefix codes.

Class

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

SignalBoxes

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

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

Example:

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

(continues on next page)

(continued from previous page)

```
>>> sb = SignalBoxes()

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

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

Methods

SignalBoxes.cdd_sigbox(*sub_dir, **kwargs)	Change directory to package data directory and sub-directories (and/or a file).
SignalBoxes.collect_non_national_rail_codes([...])	Collect signal box prefix codes of non-national rail from source web page.
SignalBoxes.collect_signal_box_prefix_codes(initial)	Collect signal box prefix codes for the given `initial` from source web page.
SignalBoxes.fetch_non_national_rail_codes([...])	Fetch signal box prefix codes of non-national rail from local backup.
SignalBoxes.fetch_signal_box_prefix_codes([...])	Fetch signal box prefix codes from local backup.

sig_box.SignalBoxes.cdd_sigbox

SignalBoxes.**cdd_sigbox**(**sub_dir*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\other-assets\signal-boxes"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `SignalBoxes`

Return type str

sig_box.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()

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

>>> type(non_national_rail_codes_dat)
<class 'dict'>
>>> print(list(non_national_rail_codes_dat.keys()))
['Non-National Rail', 'Last updated date']
```

sig_box.SignalBoxes.collect_signal_box_prefix_codes

SignalBoxes.**collect_signal_box_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()

>>> signal_boxes_a = sb.collect_signal_box_prefix_codes(initial='a')

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

>>> signal_boxes_a_codes = signal_boxes_a['A']
>>> type(signal_boxes_a_codes)
<class '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]
```

sig_box.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()
```

(continues on next page)

(continued from previous page)

```
>>> non_national_rail_codes_dat = sb.fetch_non_national_rail_codes()

>>> non_national_rail_codes = non_national_rail_codes_dat['Non-National Rail']
>>> type(non_national_rail_codes)
<class 'dict'>
>>> print(list(non_national_rail_codes.keys())[:5])
['Croydon Tramlink signals',
 'Docklands Light Railway signals',
 'Edinburgh Tramway signals',
 'Glasgow Subway signals',
 'London Underground signals']

>>> croydon_tl_signals = non_national_rail_codes['Croydon Tramlink signals']
>>> type(croydon_tl_signals)
<class 'list'>
>>> print(croydon_tl_signals[0])
None
>>> print(croydon_tl_signals[1])
Croydon Tramlink signal codes can be found on the ...
```

sig_box.SignalBoxes.fetch_signal_box_prefix_codes

SignalBoxes.**fetch_signal_box_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()

>>> signal_box_prefix_codes_dat = sb.fetch_signal_box_prefix_codes()

>>> type(signal_box_prefix_codes_dat)
```

(continues on next page)

(continued from previous page)

```
<class 'dict'>
>>> print(list(signal_box_prefix_codes_dat.keys()))
['Signal boxes', 'Last updated date']

>>> signal_box_prefix_codes_ = signal_box_prefix_codes_dat['Signal boxes']
>>> type(signal_box_prefix_codes_)
<class 'pandas.core.frame.DataFrame'>
>>> print(signal_box_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])	A class for collecting railway tunnel lengths.

Tunnels

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

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`

Example:

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

>>> tunnels = Tunnels()

>>> print(tunnels.Name)
Railway tunnel lengths
```

(continues on next page)

(continued from previous page)

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

Methods

Tunnels.cdd_tunnels(*sub_dir, **kwargs)	Change directory to package data directory and sub-directories (and/or a file).
Tunnels.collect_tunnel_lengths_by_page(page_no)	Collect data of railway tunnel lengths for a page number from source web page.
Tunnels.fetch_tunnel_lengths([update, . . .])	Fetch data of railway tunnel lengths from local backup.
Tunnels.parse_length(x)	Parse data in `'Length'` column, i.e. convert miles/yards to metres.

tunnel.Tunnels.cdd_tunnels

Tunnels.**cdd_tunnels**(**sub_dir*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\other-assets\tunnels"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `Tunnels`

Return type str

tunnel.Tunnels.collect_tunnel_lengths_by_page

Tunnels.**collect_tunnel_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

>>> tunnels = Tunnels()

>>> tunnel_len_1 = tunnels.collect_tunnel_lengths_by_page(page_no=1)
>>> type(tunnel_len_1)
<class 'dict'>
>>> print(list(tunnel_len_1.keys()))
['Page 1 (A-F)', 'Last updated date']

>>> tunnel_len_4 = tunnels.collect_tunnel_lengths_by_page(page_no=4)
>>> type(tunnel_len_4)
<class 'dict'>
>>> print(list(tunnel_len_4.keys()))
['Page 4 (others)', 'Last updated date']
```

tunnel.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
```

(continues on next page)

(continued from previous page)

```
>>> tunnels = Tunnels()

>>> tunnel_lengths_data = tunnels.fetch_tunnel_lengths()

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

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

tunnel.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

>>> tunnels = Tunnels()

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

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

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

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

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

viaduct

Collect codes of railway viaducts.

Class

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

Viaducts

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

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

Example:

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

>>> viaducts = Viaducts()

>>> print(viaducts.Name)
Railway viaducts

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

Methods

Viaducts.cdd_viaducts(*sub_dir, **kwargs)	Change directory to package data directory and sub-directories (and/or a file).
Viaducts.collect_railway_viaducts_by_page(page_no)	Collect data of railway viaducts for a given page number from source web page.
Viaducts.fetch_railway_viaducts([update, ...])	Fetch data of railway viaducts from local backup.

viaduct.Viaducts.cdd_viaducts

Viaducts.**cdd_viaducts**(**sub_dir*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\other-assets\viaducts"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `Viaducts`

Return type str

viaduct.Viaducts.collect_railway_viaducts_by_page

Viaducts.**collect_railway_viaducts_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

>>> viaducts = Viaducts()

>>> viaducts_1 = viaducts.collect_railway_viaducts_by_page(page_no=1)

>>> type(viaducts_1)
<class 'dict'>
>>> print(list(viaducts_1.keys()))
['Page 1 (A-C)', 'Last updated date']
```

viaduct.Viaducts.fetch_railway_viaducts

Viaducts.**fetch_railway_viaducts**(*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

>>> viaducts = Viaducts()

>>> viaducts_data = viaducts.fetch_railway_viaducts()

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

>>> viaducts_dat = viaducts_data['Viaducts']
>>> type(viaducts_dat)
<class 'dict'>
>>> print(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)']
```

station

Collect railway station data.

Class

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

Stations

class station.**Stations**(*data_dir=None*, *update=False*)

A class for collecting railway station data.

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`

Example:

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

>>> stn = Stations()

>>> print(stn.Name)
Stations

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

Methods

Stations.cdd_stn(*sub_dir, **kwargs)	Change directory to package data directory and sub-directories (and/or a file).
Stations.collect_railway_station_data_by_initial(initial)	Collect railway station data for the given `initial` letter.
Stations.fetch_railway_station_data([...])	Fetch railway station data from local backup.
Stations.parse_current_operator(x)	Parse 'Operator' column

station.Stations.cdd_stn

Stations.**cdd_stn**(**sub_dir*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\other-assets\stations"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `Stations`

Return type str

station.Stations.collect_railway_station_data_by_initial

Stations.**collect_railway_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()

>>> stn_data_a = stn.collect_railway_station_data_by_initial(initial='a')

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

station.Stations.fetch_railway_station_data

Stations.**fetch_railway_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 (incl. 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()

>>> stn_data = stn.fetch_railway_station_data()

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

>>> stn_dat = stn_data['Railway station data']
>>> type(stn_dat)
<class 'pandas.core.frame.DataFrame'>
>>> print(stn_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]
```

station.Stations.parse_current_operator

static Stations.**parse_current_operator**(*x*)
Parse 'Operator' column

depot

Collect depots codes.

Class

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

Depots

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

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

Depots.cdd_depots(*sub_dir, **kwargs)	Change directory to package data directory and sub-directories (and/or a file).

continues on next page

Table 23 – continued from previous page	
Depots.collect_1950_system_codes([...])	Collect 1950 system (pre-TOPS) codes from source web page.
Depots.collect_four_digit_pre_tops_codes([...])	Collect four-digit pre-TOPS codes from source web page.
Depots.collect_gwr_codes([...])	Collect Great Western Railway (GWR) depot codes from source web page.
Depots.collect_two_char_tops_codes([...])	Collect two-character TOPS codes from source web page.
Depots.fetch_1950_system_codes([update, ...])	Fetch 1950 system (pre-TOPS) codes from local backup.
Depots.fetch_depot_codes([update, ...])	Fetch depots codes from local backup.
Depots.fetch_four_digit_pre_tops_codes([...])	Fetch four-digit pre-TOPS codes from local backup.
Depots.fetch_gwr_codes([update, pickle_it, ...])	Fetch Great Western Railway (GWR) depot codes from local backup.
Depots.fetch_two_char_tops_codes([update, ...])	Fetch two-character TOPS codes from local backup.

depot.Depots.cdd_depots

Depots.**cdd_depots**(**sub_dir*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\other-assets\depots"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `Depots`

Return type str

depot.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)
<class 'dict'>
>>> print(list(system_1950_codes_dat.keys()))
['1950 system (pre-TOPS) codes', 'Last updated date']
```

depot.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()

>>> four_digit_pre_tops_codes_dat = depots.collect_four_digit_pre_tops_codes()
To collect data of four digit pre-TOPS codes? [No]|Yes: yes

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

>>> type(four_digit_pre_tops_codes_dat['Four digit pre-TOPS codes'])
<class 'dict'>
```

depot.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)
<class 'dict'>
>>> print(list(gwr_codes_dat.keys()))
['GWR codes', 'Last updated date']
```

depot.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()

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

>>> type(two_char_tops_codes_dat)
<class 'dict'>
>>> print(list(two_char_tops_codes_dat.keys()))
['Two character TOPS codes', 'Last updated date']
```

depot.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()

>>> system_1950_codes_dat = depots.fetch_1950_system_codes()

>>> system_1950_codes = system_1950_codes_dat['1950 system (pre-TOPS) codes']

>>> type(system_1950_codes)
<class 'pandas.core.frame.DataFrame'>
>>> print(system_1950_codes.head())
  Code             Depot                                              Notes
0   1A         Willesden              From 1950. Became WN from 6 May 1973
1   1B            Camden                      From 1950. To 3 January 1966
2   1C           Watford              From 1950. Became WJ from 6 May 1973
3   1D  Devons Road, Bow  Previously 13B to 9 June 1950. Became 1J from ...
4   1D        Marylebone  Previously 14F to 31 August 1963. Became ME fr...
```

depot.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()

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

depot.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()

>>> four_digit_pre_tops_codes_dat = depots.fetch_four_digit_pre_tops_codes()

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

>>> four_digit_pre_tops_codes =             ...     four_digit_pre_tops_codes_
↪dat['Four digit pre-TOPS codes']

>>> print(list(four_digit_pre_tops_codes.keys()))
['Main Works',
 'London Midland Region',
 'Western Region',
 'Southern Region',
 'Eastern Region',
 'Scottish Region']
```

depot.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()

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

>>> gwr_codes_alpha = gwr_codes['Alphabetical codes']
>>> type(gwr_codes_alpha)
<class '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
```

depot.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()

>>> two_char_tops_codes_dat = depots.fetch_two_char_tops_codes()

>>> type(two_char_tops_codes_dat)
<class 'dict'>
>>> print(list(two_char_tops_codes_dat.keys()))
['Two character TOPS codes', 'Last updated date']
```

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])	A class for collecting codes of infrastructure features.

Features

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

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

Example:

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

>>> features = Features()

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

Methods

Features.cdd_features(*sub_dir, **kwargs)	Change directory to package data directory and sub-directories (and/or a file).
Features.collect_buzzer_codes([...])	Collect buzzer codes from source web page.
Features.collect_habds_and_wilds([...])	Collect codes of HABDs and WILDs from source web page.
Features.collect_telegraph_codes([...])	Collect telegraph code words from source web page.
Features.collect_water_troughs([...])	Collect codes of water troughs from source web page.

continues on next page

Table 25 – continued from previous page	
Features.fetch_buzzer_codes([update, ...])	Fetch buzzer codes from local backup.
Features.fetch_features_codes([update, ...])	Fetch features codes from local backup.
Features.fetch_habds_and_wilds([update, ...])	Fetch codes of HABDs and WILDs from local backup.
Features.fetch_telegraph_codes([update, ...])	Fetch telegraph code words from local backup.
Features.fetch_water_troughs([update, ...])	Fetch codes of water troughs from local backup.
Features.parse_vulgar_fraction_in_length(x)	Parse 'VULGAR FRACTION' for 'Length' of water trough locations.

feature.Features.cdd_features

Features.**cdd_features**(**sub_dir*, ***kwargs*)

Change directory to package data directory and sub-directories (and/or a file).

The directory for this module: `"\dat\other-assets\features"`.

Parameters

- **sub_dir** (*str*) – sub-directory or sub-directories (and/or a file)
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns path to the backup data directory for `Features`

Return type str

feature.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()

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

>>> type(buzzer_codes_dat)
<class 'dict'>
>>> print(list(buzzer_codes_dat.keys()))
['Buzzer codes', 'Last updated date']
```

feature.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()

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

>>> type(habds_and_wilds_codes_dat)
<class 'dict'>
>>> print(list(habds_and_wilds_codes_dat.keys()))
['HABD and WILD', 'Last updated date']
```

feature.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()

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

>>> type(telegraph_codes_dat)
<class 'dict'>
>>> print(list(telegraph_codes_dat.keys()))
['Telegraphic codes', 'Last updated date']
```

feature.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()
```

(continues on next page)

(continued from previous page)

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

>>> type(water_troughs_dat)
<class 'dict'>
>>> print(water_troughs_dat)
['Water troughs', 'Last updated date']
```

feature.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()

>>> buzzer_codes_dat = features.fetch_buzzer_codes()

>>> buzzer_codes = buzzer_codes_dat['Buzzer codes']
>>> type(buzzer_codes)
<class 'pandas.core.frame.DataFrame'>
>>> print(buzzer_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
```

feature.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()

>>> features_codes_dat = features.fetch_features_codes()

>>> type(features_codes_dat)
<class 'dict'>
>>> print(list(features_codes_dat.keys()))
['Features', 'Last updated date']
```

feature.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()

>>> habds_and_wilds_codes_dat = features.fetch_habds_and_wilds()

>>> habds_and_wilds_codes = habds_and_wilds_codes_dat['HABD and WILD']
>>> type(habds_and_wilds_codes)
<class 'dict'>
>>> print(list(habds_and_wilds_codes.keys()))
['HABD', 'WILD']

>>> habd = habds_and_wilds_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]
```

feature.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()

>>> telegraph_codes_dat = features.fetch_telegraph_codes()

>>> telegraph_codes = telegraph_codes_dat['Telegraphic codes']
>>> type(telegraph_codes)
<class 'dict'>
>>> print(list(telegraph_codes.keys()))
['Official codes', 'Unofficial codes']

>>> official_codes = telegraph_codes['Official codes']
>>> type(official_codes)
<class 'pandas.core.frame.DataFrame'>
>>> print(official_codes.head())
    Code  ...               In use
0    ACK  ...            BR, 1980s
1   ADEX  ...  GWR, 1939 BR, 1980s
2   AJAX  ...            BR, 1980s
3  ALERT  ...            BR, 1980s
4  AMBER  ...            BR, 1980s

[5 rows x 3 columns]
```

feature.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()
```

(continues on next page)

(continued from previous page)

```
>>> water_troughs_dat = features.fetch_water_troughs()

>>> water_troughs_codes = water_troughs_dat['Water troughs']

>>> type(water_troughs_codes)
<class 'pandas.core.frame.DataFrame'>
>>> print(water_troughs_codes.head())
    ELR  Trough Name  ... Length_yard                        Notes
0   BEI    Eckington  ...         NaN
1   BHL  Aldermaston  ...  620.000000            Installed by 1904
2  CGJ2        Moore  ...  506.666667
3  CGJ6     Lea Road  ...  561.000000  Taken out of use 8 May 1967
4  CGJ6        Brock  ...  560.000000

[5 rows x 5 columns]
```

feature.Features.parse_vulgar_fraction_in_length

static Features.**parse_vulgar_fraction_in_length**(*x*)

Parse 'VULGAR FRACTION' for 'Length' of water trough locations.

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.

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

3.2 Modules

3.2.1 _line_data

Collect line data.

class pyrcs._line_data.**LineData**(*update=False*)

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`

Example:

```
>>> from pyrcs import LineData

>>> ld = LineData()

>>> # To get 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 location codes
>>> 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
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]
```

3.2.2 _other_assets

Collect data of other assets.

class pyrcs._other_assets.**OtherAssets**(*update=False*)

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`

Example:

```
>>> from pyrcs import OtherAssets

>>> oa = OtherAssets()

>>> # To get data of railway stations
>>> railway_station_data = oa.Stations.fetch_railway_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_signal_box_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]
```

3.2.3 updater

Update package data.

Site map

collect_site_map([confirmation_required])	Collect data of the site map from source web page.
fetch_site_map([update, ...])	Fetch the site map from the package data.

pyrcs.updater.collect_site_map

pyrcs.updater.**collect_site_map**(*confirmation_required=True*)

Collect data of the site map from source web page.

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

Returns dictionary of site map data

Return type dict

Examples:

```
>>> from pyrcs.updater import collect_site_map

>>> site_map_dat = collect_site_map()
To collect the site map? [No]|Yes: yes

>>> type(site_map_dat)
<class 'dict'>
>>> print(list(site_map_dat.keys()))
['Home', 'Line data', 'Other assets', '"Legal/financial" lists', 'Miscellaneous']
```

pyrcs.updater.fetch_site_map

pyrcs.updater.**fetch_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*, *int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns dictionary of site map data

Return type dict

Examples:

```
>>> from pyrcs.updater import fetch_site_map

>>> site_map_dat = fetch_site_map()

>>> type(site_map_dat)
<class 'dict'>
>>> print(site_map_dat['Home'])
http://www.railwaycodes.org.uk/index.shtml
```

Local backup

update_backup_data([verbose, time_gap])	Update package data.

pyrcs.updater.update_backup_data

pyrcs.updater.**update_backup_data**(*verbose=False*, *time_gap=5*)

Update package data.

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)
```

3.2.4 utils

Utilities - Helper functions.

Source homepage

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

pyrcs.utils.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 directory

cd_dat(*sub_dir[, dat_dir, mkdir])	Change directory to *dat_dir/* and sub-directories within a package.

pyrcs.utils.cd_dat

pyrcs.utils.**cd_dat**(**sub_dir*, *dat_dir='dat'*, *mkdir=False*, ***kwargs*)

Change directory to *dat_dir/* and sub-directories within a package.

Parameters

- **sub_dir** (*str*) – name of directory; names of directories (and/or a filename)
- **dat_dir** (*str*) – name of a directory to store data, defaults to `"dat"`
- **mkdir** (*bool*) – whether to create a directory, defaults to `False`
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns a full path to a directory (or a file) under `data_dir`

Return type str

Example:

```
>>> import os
>>> from pyrcs.utils import cd_dat
```

(continues on next page)

(continued from previous page)

```
>>> path_to_dat_dir = cd_dat("line-data", dat_dir="dat", mkdir=False)
>>> print(os.path.relpath(path_to_dat_dir))
pyrcs\dat\line-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.
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.

pyrcs.utils.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
```

(continues on next page)

(continued from previous page)

```
>>> 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)
```

pyrcs.utils.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)
>>> 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)
```

pyrcs.utils.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
```

(continues on next page)

(continued from previous page)

```
>>> 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
```

pyrcs.utils.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
>>> 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'>
```

pyrcs.utils.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
```

pyrcs.utils.yards_to_nr_mileage

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

Convert yards to Network Rail mileages.

Parameters **yards** (*int or float, numpy.nan, 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'>
```

pyrcs.utils.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)
>>> print(num_mileage_dat)
10.022
```

pyrcs.utils.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
```

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.

pyrcs.utils.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', ' ']
```

pyrcs.utils.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'>
```

pyrcs.utils.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)
```

(continues on next page)

(continued from previous page)

```
>>> 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')
```

pyrcs.utils.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, 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

fake_requests_headers([randomized])	Make a fake HTTP headers for requests.get.
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_station_data_catalogue(source_url, ...)	Get catalogue of railway station data.
get_track_diagrams_items(source_url, source_key)	Get catalogue of track diagrams.

pyrcs.utils.fake_requests_headers

pyrcs.utils.**fake_requests_headers**(*randomized=False*)

Make a fake HTTP headers for requests.get.

Parameters **randomized** (*bool*) – whether to go for a random agent, defaults to False

Returns fake HTTP headers

Return type dict

Examples:

```
>>> from pyhelpers.ops import fake_requests_headers

>>> fake_headers_ = fake_requests_headers()
>>> print(fake_headers_)
{'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Ch...

>>> fake_headers_ = fake_requests_headers(randomized=True)
>>> print(fake_headers_)
{'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36␣
↪(KHTML ...
```

Note: The above fake_headers_ may be different every time we run the examples.

pyrcs.utils.get_last_updated_date

pyrcs.utils.**get_last_updated_date**(*url*, *parsed=True*, *as_date_type=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`

Returns date of when the specified web page was last updated

Return type str, datetime.date, None

Examples:

```
>>> from pyrcs.utils import get_last_updated_date

>>> last_update_date_ = get_last_updated_date(
...     url='http://www.railwaycodes.org.uk/crs/CRSa.shtm', parsed=True,
...     as_date_type=False)
>>> type(last_update_date_)
<class 'str'>

>>> last_update_date_ = get_last_updated_date(
...     url='http://www.railwaycodes.org.uk/crs/CRSa.shtm', parsed=True,
...     as_date_type=True)
>>> type(last_update_date_)
<class 'datetime.date'>

>>> last_update_date_ = get_last_updated_date(
...     url='http://www.railwaycodes.org.uk/linedatamenu.shtm')
>>> print(last_update_date_)
None
```

pyrcs.utils.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*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns catalogue in the form {'<title>': '<URL>'}

Return type dict

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'>
```

pyrcs.utils.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*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns

Return type dict

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']
```

pyrcs.utils.get_station_data_catalogue

pyrcs.utils.**get_station_data_catalogue**(*source_url*, *source_key*, *update=False*)

Get catalogue of railway station data.

Parameters

- **source_url** (*str*) – URL to the source web page
- **source_key** (*str*) – key of the returned catalogue (which is a dictionary)
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`

Returns catalogue of railway station data

Return type dict

See `pyrcs.other_assets.Stations()`

pyrcs.utils.get_track_diagrams_items

pyrcs.utils.**get_track_diagrams_items**(*source_url*, *source_key*, *update=False*)

Get catalogue of track diagrams.

Parameters

- **source_url** (*str*) – URL to the source web page
- **source_key** (*str*) – key of the returned catalogue (which is a dictionary)
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`

Returns catalogue of railway station data

Return type dict

See `pyrcs.line_data.TrackDiagrams()`

Rectification of location names

fetch_location_names_repl_dict([k, regex, . . .])	Create a dictionary for rectifying location names.
update_location_name_repl_dict(new_items, regex)	Update the location-name replacement dictionary in the package data.

pyrcs.utils.fetch_location_names_repl_dict

pyrcs.utils.**fetch_location_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_location_names_repl_dict

>>> repl_dict = fetch_location_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_location_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-
```

pyrcs.utils.update_location_name_repl_dict

pyrcs.utils.**update_location_name_repl_dict**(*new_items*, *regex*, *verbose=False*)

Update the location-name 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*) – whether to print relevant information in console as the function runs, defaults to False

Example:

```
>>> from pyrcs.utils import update_location_name_repl_dict
```

```
>>> new_items_ = {}
>>> update_location_name_repl_dict(new_items_, regex=False)
```

Fixers

fix_num_stanox(stanox_code)	Fix 'STANOX' if it is loaded as numbers.

pyrcs.utils.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
```

Misc

is_str_float(str_val)	Check if a string-type variable can express a float-type value.

pyrcs.utils.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
```

_line_data	Collect line data.
_other_assets	Collect data of other assets.
updater	Update package data.
utils	Utilities - Helper functions.

CHAPTER

FOUR

LICENSE

PyRCS is licensed under GNU General Public License v3.0 (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 *railwaycodes* website. The author of the package would like to thank the website editor and all contributors to the data resources.

PYTHON MODULE INDEX

p

INDEX

G

H

I

L

M

N

P

S

T

U

V

Y