PyRCS

An open-source tool for collecting railway codes used in different UK rail industry systems

Release 1.1.0

Qian Fu

Birmingham Energy Institute, University of Birmingham

First release: **August 2019**

Last updated: **April 2026**

Table of Contents

Chapter 1

About PyRCS

PyRCS is an open-source Python package that simplifies the collection and management of railway codes used across different systems in the UK rail industry. It provides a practical toolkit for researchers, practitioners and frequent users of the Railway Codes website who work extensively with railway codes in the UK. By leveraging Python's capabilities, PyRCS enables efficient access, retrieval and manipulation of railway code data, enhancing productivity and effectiveness in working with these codes.

During *Installation*, PyRCS includes a set of pre-packaged data. When users request data from a specific category on the Railway Codes website, PyRCS loads the corresponding pre-packaged data for that category by default. Additionally, it provides functionality for direct access to the latest data from the source website, ensuring users stay up to date. Users can also update the pre-packaged data as needed, keeping their resources synchronized with the latest developments.

With PyRCS, users can leverage Python's power to streamline workflows and enhance productivity when working with railway codes in the UK rail industry.

Chapter 2

Installation

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

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

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

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

Note

- If using a virtual environment, make sure it is activated.
- It is recommended to add pip install the option `--upgrade` (or `-U`) to ensure that you are getting the latest stable release of the package.
- For more general instructions on the installation of Python packages, please refer to the official guide on Installing Packages.

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

```
>>> import pyrcs

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

```
The latest version is: 1.1.0
```

Chapter 3

Quick Start

This brief tutorial provides a step-by-step guide to using PyRCS, highlighting its key functionalities. It demonstrates how to retrieve three key categories of codes used in the UK railway system, which are commonly applied in both practical and research contexts:

- Location identifiers (CRS, NLC, TIPLOC and STANOX codes);
- Engineer's Line References (ELRs) and their associated mileage files;
- Railway station data (mileages, operators and grid coordinates).

Through practical examples, this tutorial will guide you in understanding how PyRCS works and how to use it effectively.

3.1 Location Identifiers

< Back to Top | *Next >*

The location identifiers, including CRS, NLC, TIPLOC and STANOX codes, are classified as line data on the Railway Codes website. To retrieve these codes using PyRCS, we can use the `LocationIdentifiers` class, contained in the `line_data` subpackage.

First, let's import the class and create an instance:

```
>>> from pyrcs.line_data import LocationIdentifiers
>>> # Alternatively, from pyrcs import LocationIdentifiers
>>> lid = LocationIdentifiers()
>>> lid.NAME
'CRS, NLC, TIPLOC and STANOX codes'
>>> lid.URL
'http://www.railwaycodes.org.uk/crs/crs0.shtm'
```

Alternatively, we can create the instance using the `LineData` class:

```
>>> from pyrcs.collector import LineData
>>> # Alternatively, from pyrcs import LineData
>>> ld = LineData()
>>> lid_ = ld.LocationIdentifiers
>>> lid.NAME == lid_.NAME
True
```

Note

- The instance `ld` encompasses all classes within the line data category.
- `lid_` is equivalent to `lid`.

3.1.1 Location identifiers by initial letter

We can retrieve codes (in pandas.DataFrame format) for all locations starting with a specific letter using the *`LocationIdentifiers.collect_loc_id()`* method. This input value for the parameter is case-insensitive. For example, to get the codes for locations whose names begin with the letter `'A'` (or `'a'`):

```
>>> loc_a_codes = lid.collect_loc_id(initial='a', verbose=True)
To collect data of CRS, NLC, TIPLOC and STANOX codes beginning with "A"
? [No]|Yes: yes
Collecting the data ... Done.
>>> type(loc_a_codes)
dict
>>> list(loc_a_codes.keys())
['A', 'Notes', 'Last updated date']
```

As shown above, `loc_a_codes` is a dictionary (i.e. in dict format) with the following *keys*:

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

The corresponding *values* are:

- `loc_a_codes['A']` - CRS, NLC, TIPLOC and STANOX codes for the locations whose names begin with `'A'`, referring to the table on the Locations beginning A web page.
- `loc_a_codes['Notes']` - Any additional information provided on the web page (if available).
- `loc_a_codes['Last updated date']` - The date when the Locations beginning A web page was last updated.

A snapshot of the data contained in `loc_a_codes` is demonstrated below:

```
>>> loc_a_codes_dat = loc_a_codes['A']
>>> type(loc_a_codes_dat)
pandas.core.frame.DataFrame
>>> loc_a_codes_dat
                              Location CRS  ... STANME_Note STANOX_Note
0                    1999 Reorganisations      ...
1                                      A1      ...
2                          A463 Traded In      ...
3     A483 Road Scheme Supervisors Closed      ...
4                                  Aachen      ...
...                                   ...  ..  ...         ...         ...
3322                       Ayr Wagon Team      ...
3323                       Ayr Wagon Team      ...
3324                       Ayr Wagon Team      ...
```

(continues on next page)

(continued from previous page)

```
3325                          Ayr Welders      ...
3326                    Aztec Travel S378      ...
[3327 rows x 12 columns]
>>> print(f"Notes: {loc_a_codes['Notes']}")
>>> print(f"Last updated date: {loc_a_codes['Last updated date']}")
Notes: None
Last updated date: 2025-12-11

>>> ## Try more examples! Uncomment the lines below and run:
>>> # loc_a_codes = lid.fetch_loc_id('a')  # Fetch location codes starting with 'A'
>>> # loc_codes = lid.fetch_loc_id()  # Fetch all location codes
```

3.1.2 All available location identifiers

Beyond retrieving location codes for a specific letter, we can use the *LocationIdentifiers.fetch_codes()* method to obtain codes for all locations with names starting from `'A'` to `'Z'`:

```
>>> loc_codes = lid.fetch_codes()
>>> type(loc_codes)
dict
>>> list(loc_codes.keys())
['Location ID', 'Other systems', 'Notes', 'Last updated date']
```

The `loc_codes` object is a dictionary with the following *keys*:

- `'Location ID'`
- `'Other systems'`
- `'Notes'`
- `'Latest update date'`

The corresponding *values* are:

- `loc_codes['Location ID']` - CRS, NLC, TIPLOC, and STANOX codes for all locations listed across the relevant web pages.
- `loc_codes['Other systems']` - Codes related to the other systems.
- `loc_codes['Notes']` - Any notes and information (if available).
- `loc_codes['Latest update date']` - The latest `'Last updated date'` across all initial-specific data.

Here is a snapshot of the data contained in `loc_codes`:

```
>>> lid.KEY
'Location ID'
>>> loc_codes_dat = loc_codes[lid.KEY]  # loc_codes['Location ID']
>>> type(loc_codes_dat)
pandas.core.frame.DataFrame
>>> loc_codes_dat
                                Location CRS  ... STANME_Note STANOX_Note
0                    1999 Reorganisations      ...
```

(continues on next page)

(continued from previous page)

```
1                                    A1      ...
2                           A463 Traded In   ...
3      A483 Road Scheme Supervisors Closed   ...
4                                  Aachen    ...
...                                   ...  ..  ...         ...         ...
60023                             ZZTYALS    ...
60024                             ZZTYKKH    ...
60025                             ZZTYLIN    ...
60026                             ZZTYSGY    ...
60027                             ZZWMNST    ...
[60028 rows x 12 columns]
>>> loc_codes_dat[['Location', 'Location_Note']]
                                Location    Location_Note
0                    1999 Reorganisations
1                                    A1
2                           A463 Traded In
3      A483 Road Scheme Supervisors Closed
4                                  Aachen
...                                   ...              ...
60023                             ZZTYALS       see Alston
60024                             ZZTYKKH    see Kirkhaugh
60025                             ZZTYLIN      see Lintley
60026                             ZZTYSGY   see Slaggyford
60027                             ZZWMNST  see Westminster
[60028 rows x 2 columns]
```

To access codes from other systems, such as Crossrail or the Tyne & Wear Metro:

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

For example, to view the data for Crossrail:

```
>>> crossrail_codes_dat = os_codes_dat['Crossrail']
>>> type(crossrail_codes_dat)
pandas.core.frame.DataFrame
>>> crossrail_codes_dat.head()
                                       Location  ... New operating code
0                                    Abbey Wood  ...                ABW
1        Abbey Wood Bolthole Berth/Crossrail Sidings  ...
2                              Abbey Wood Sidings  ...
3                                     Bond Street  ...                BDS
4                                   Canary Wharf  ...                CWX
...                                          ...  ...                ...
26                                   Whitechapel  ...                ZLW
```

(continues on next page)

(continued from previous page)

```
27              Whitechapel Vallance Road Crossover  ...
28                                        Woolwich  ...               WWC
29                                       [unknown]  ...
30                                       [unknown]  ...
[31 rows x 5 columns]

>>> ## Try more examples! Uncomment the lines below and run:
>>> ## Get a dictionary for STANOX codes and location names
>>> # stanox_dict = lid.make_xref_dict('STANOX')
>>> ## ... and for STANOX, TIPLOC and location names starting with 'A'
>>> # stanox_tiploc_dict_a = lid.make_xref_dict(['STANOX', 'TIPLOC'], initials='a')
```

3.2 ELRs and mileages

< Previous | *Back to Top* | *Next >*

Engineer's Line References (ELRs) are also commonly encountered in various data sets within the UK's railway system. To retrieve the codes for ELRs along with their associated mileage files, we can use the *ELRMileages* class:

```
>>> from pyrcs.line_data import ELRMileages
>>> # Alternatively, from pyrcs import ELRMileages
>>> em = ELRMileages()
>>> em.NAME
"Engineer's Line References (ELRs)"
>>> em.URL
'http://www.railwaycodes.org.uk/elrs/elr0.shtm'
```

3.2.1 Engineer's Line References (ELRs)

Similar to location identifiers, the ELR codes on the Railway Codes website are arranged alphabetically based on their initial letters. We can use the *ELRMileages.collect_elr()* method to obtain ELRs starting with a specific letter. For example, to get the data for ELRs beginning with the letter `'A'`:

```
>>> elrs_a_codes = em.collect_elr(initial='a', verbose=True)
To collect data of Engineer's Line References (ELRs) beginning with "A"
? [No]|Yes: yes
Collecting the data ... Done.
>>> type(elrs_a_codes)
dict
>>> list(elrs_a_codes.keys())
['A', 'Last updated date']
```

The `elrs_a_codes` object is a dictionary with the following *keys*:

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

The corresponding *values* are:

- `elrs_a_codes['A']` - Data for ELRs that begin with `'A'`, referring to the table presented on the ELRs beginning with A web page.

- `elrs_a_codes['Last updated date']` - The date when the ELRs beginning with A web page was last updated.

Here is a snapshot of the data contained in `elrs_a_codes`:

```
>>> elrs_a_codes_dat = elrs_a_codes['A']
>>> type(elrs_a_codes_dat)
pandas.core.frame.DataFrame
>>> elrs_a_codes_dat
      ELR  ...         Notes
0     AAL  ...      Now NAJ3
1     AAM  ...  Formerly AML
2     AAV  ...
3     ABB  ...       Now AHB
4     ABB  ...
..    ...  ...           ...
188  AYR4  ...
189  AYR5  ...
190  AYR6  ...
191   AYS  ...
192   AYT  ...
[193 rows x 5 columns]
>>> print(f"Last updated date: {elrs_a_codes['Last updated date']}")
Last updated date: 2025-06-23
```

To retrieve data for all ELRs (from `'A'` to `'Z'`), we can use the *`ELRMileages.fetch_elr()`* method:

```
>>> elrs_codes = em.fetch_elr()
>>> type(elrs_codes)
dict
>>> list(elrs_codes.keys())
['ELRs and mileages', 'Last updated date']
```

Similarly, `elrs_codes` is a dictionary with the following *keys*:

- `'ELRs and mileages'`
- `'Latest update date'`

The corresponding *values* are:

- `elrs_codes['ELRs and mileages']` - Codes for all available ELRs (with the initial letters ranging from `'A'` to `'Z'`).
- `elrs_codes['Latest update date']` - The most recent update date among all the ELR data.

Here is a snapshot of the data contained in `elrs_codes`:

```
>>> elrs_codes_dat = elrs_codes[em.KEY]
>>> type(elrs_codes_dat)
pandas.core.frame.DataFrame
>>> elrs_codes_dat
       ELR  ...         Notes
0      AAL  ...      Now NAJ3
1      AAM  ...  Formerly AML
2      AAV  ...
3      ABB  ...       Now AHB
4      ABB  ...
```

(continues on next page)

(continued from previous page)

```
...    ...  ...           ...
4573  ZGW1  ...
4574  ZGW2  ...
4575   ZZY  ...
4576   ZZZ  ...
4577  ZZZ9  ...
[4578 rows x 5 columns]

>>> ## Try more examples! Uncomment the lines below and run:
>>> # elrs_a_codes = em.fetch_elr(initial='a')  # Fetch ELRs starting with 'A'
>>> # elrs_b_codes = em.fetch_elr(initial='B')  # Fetch ELRs starting with 'B'
```

3.2.2 Mileage file of a given ELR

In addition to the codes of ELRs, each ELR is associated with a mileage file that specifies the major mileages along the line. To retrieve this data, we can use the *ELRMileages.fetch_mileage_file()* method.

For example, to get the mileage file for 'AAM':

```
>>> amm_mileage_file = em.fetch_mileage_file(elr='AAM')
>>> type(amm_mileage_file)
dict
>>> list(amm_mileage_file.keys())
['ELR', 'Line', 'Sub-Line', 'Mileage', 'Notes']
```

The `amm_mileage_file` object is also a dictionary and has the following *keys*:

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

The corresponding *values* are:

- `amm_mileage_file['ELR']` - The given ELR (in this example, `'AAM'`).
- `amm_mileage_file['Line']` - The name of the line associated with the ELR.
- `amm_mileage_file['Sub-Line']` - The sub-line name (if applicable).
- `amm_mileage_file['Mileage']` - The major mileages along the line.
- `amm_mileage_file['Notes']` - Additional notes or information (if available).

Here is a snapshot of the data contained in `amm_mileage_file`:

```
>>> amm_mileage_file['Line']
'Ashchurch and Malvern Line'
>>> amm_mileage_file['Mileage']
   Mileage Mileage_Note  ... Link_2_ELR Link_2_Mile_Chain
0   0.0000               ...
```

(continues on next page)

(continued from previous page)

```
1   0.0154              ...
2   0.0396              ...
3   1.1012              ...
4   1.1408              ...
5   5.0330              ...
6   7.0374              ...
7  11.1298              ...
8  13.0638              ...
[9 rows x 11 columns]

>>> ## Try more examples! Uncomment the lines below and run:
>>> # xre_mileage_file = em.fetch_mileage_file('XRE')  # Fetch mileage file for 'XRE'
>>> # your_mileage_file = em.fetch_mileage_file(elr='?')  # ... and for a given ELR '?'
```

3.3 Railway station data

< Previous | *Back to Top* | *Next >*

The railway station data includes information such as the station name, ELR, mileage, status, owner, operator, coordinates and grid reference. This data is available in the other assets section of the Railway Codes website and can be retrieved using the *Stations* class contained in the *other_assets* subpackage.

To get the data, let's import the *Stations* class and create an instance:

```
>>> from pyrcs.other_assets import Stations  # from pyrcs import Stations
>>> stn = Stations()
>>> stn.NAME
'Railway station data'
>>> stn.URL
'http://www.railwaycodes.org.uk/stations/station0.shtm'
```

Alternatively, we can also create the instance by using the *OtherAssets* class:.

```
>>> from pyrcs.collector import OtherAssets  # from pyrcs import OtherAssets
>>> oa = OtherAssets()
>>> stn_ = oa.Stations
>>> stn.NAME == stn_.NAME
True
```

Note

- The instance `stn` encompasses all classes within the other assets category.
- `stn_` is equivalent to `stn`.

3.3.1 Railway stations by initial letter

We can obtain railway station data based on the first letter (e.g. `'A'` or `'Z'`) of the station's name using the *Stations.collect_locations()* method. For example, to get data for stations starting with `'A'`:

```
>>> stn_loc_a_codes = stn.collect_locations(initial='a', verbose=True)
To collect data of mileages, operators and grid coordinates beginning with "A"
? [No]|Yes: yes
Collecting the data ... Done.
>>> type(stn_loc_a_codes)
dict
>>> list(stn_loc_a_codes.keys())
['A', 'Last updated date']
```

The dictionary `stn_loc_a_codes` includes the following *keys*:

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

The corresponding *values* are:

- `stn_loc_a_codes['A']` - Data for railway stations whose names begin with `'A'`, including mileages, operators and grid coordinates, referring to the table on the Stations beginning with A web page.
- `stn_loc_a_codes['Last updated date']` - The date when the Stations beginning with A web page was last updated.

Here is a snapshot of the data contained in `stn_loc_a`:

```
>>> stn_loc_a_codes_dat = stn_loc_a_codes['A']
>>> type(stn_loc_a_codes_dat)
pandas.core.frame.DataFrame
>>> stn_loc_a_codes_dat
                              Station  ...                  Former Operator
0    Abbey Wood Abbey Wood / ABBEY WOOD  ...  MTR Corporation (Crossrail)...
1    Abbey Wood Abbey Wood / ABBEY WOOD  ...  MTR Corporation (Crossrail)...
2                                 Aber  ...  Keolis Amey Operations/Gwei...
3                            Abercynon  ...  Keolis Amey Operations/Gwei...
4                            Abercynon  ...  Keolis Amey Operations/Gwei...
..                                 ...  ...                             ...
138              Aylesbury Vale Parkway  ...
139                          Aylesford  ...  London & South Eastern Rail...
140                           Aylesham  ...  London & South Eastern Rail...
141                                Ayr  ...  Abellio ScotRail from 1 Apr...
142                                Ayr  ...  Abellio ScotRail from 1 Apr...
[143 rows x 14 columns]
>>> stn_loc_a_codes_dat.columns.to_list()
['Station',
 'Station Note',
 'ELR',
 'Mileage',
 'Note',
 'Degrees Longitude',
 'Degrees Latitude',
 'Grid Reference',
 'CRS',
 'CRS Note',
 'Owner',
 'Former Owner',
 'Operator',
```

(continues on next page)

(continued from previous page)

```
 'Former Operator']
>>> stn_loc_a_codes_dat[['Station', 'ELR', 'Mileage']]
                              Station   ELR   Mileage
0    Abbey Wood Abbey Wood / ABBEY WOOD   NKL  11m 43ch
1    Abbey Wood Abbey Wood / ABBEY WOOD   XRS  24.458km
2                                 Aber   CAR   8m 69ch
3                            Abercynon   CAM  16m 28ch
4                            Abercynon   ABD  16m 28ch
..                                 ...   ...       ...
138             Aylesbury Vale Parkway  MCJ2  40m 38ch
139                          Aylesford  PWS2  38m 74ch
140                           Aylesham   FDM  68m 66ch
141                                Ayr  AYR6  40m 49ch
142                                Ayr  STR1  40m 49ch
[143 rows x 3 columns]
>>> print(f"Last updated date: {stn_loc_a_codes['Last updated date']}")
Last updated date: 2025-12-17
```

3.3.2 All available railway stations

To retrieve data for all railway stations available in the other assets category, we can use the *`Stations.fetch_locations()`* method:

```
>>> stn_loc_codes = stn.fetch_locations()
>>> type(stn_loc_codes)
dict
>>> list(stn_loc_codes.keys())
['Mileages, operators and grid coordinates', 'Last updated date']
```

The dictionary `stn_loc_codes` includes the following *keys*:

- `'Mileages, operators and grid coordinates'`
- `'Latest update date'`

The corresponding *values* are:

- `stn_loc_codes['Mileages, operators and grid coordinates']` - Data for all railway stations, with the initial letters ranging from `'A'` to `'Z'`.
- `stn_loc_codes['Latest update date']` - The most recent update date among all the station data.

Here is a snapshot of the data contained in `stn_loc_codes`:

```
>>> stn.KEY_TO_STN
'Mileages, operators and grid coordinates'
>>> stn_loc_codes_dat = stn_loc_codes[stn.KEY_TO_STN]
>>> type(stn_loc_codes_dat)
pandas.core.frame.DataFrame
>>> stn_loc_codes_dat
                              Station  ...                 Former Operator
0    Abbey Wood Abbey Wood / ABBEY WOOD  ...  MTR Corporation (Crossrail)...
1    Abbey Wood Abbey Wood / ABBEY WOOD  ...  MTR Corporation (Crossrail)...
2                                 Aber  ...  Keolis Amey Operations/Gwei...
3                            Abercynon  ...  Keolis Amey Operations/Gwei...
```

(continues on next page)

(continued from previous page)

```
4                            Abercynon  ...  Keolis Amey Operations/Gwei...
...                                ...  ...                           ...
2912                             York  ...  East Coast Main Line Compa...
2913                             York  ...  East Coast Main Line Compa...
2914                           Yorton  ...  Keolis Amey Operations/Gwe...
2915                    Ystrad Mynach  ...  Keolis Amey Operations/Gwe...
2916                   Ystrad Rhondda  ...  Keolis Amey Operations/Gwe...
[2917 rows x 14 columns]
>>> loc_cols = ['Station', 'ELR', 'Mileage', 'Degrees Longitude', 'Degrees Latitude']
>>> stn_loc_codes_dat[loc_cols].head()
                           Station  ELR  ... Degrees Longitude  Degrees Latitude
0  Abbey Wood Abbey Wood / ABBEY WOOD  NKL  ...            0.1204           51.4908
1  Abbey Wood Abbey Wood / ABBEY WOOD  XRS  ...            0.1204           51.4908
2                               Aber  CAR  ...           -3.2305           51.5755
3                          Abercynon  CAM  ...           -3.3294           51.6434
4                          Abercynon  ABD  ...           -3.3294           51.6434
[5 rows x 5 columns]
>>> print(f"Last updated date: {stn_loc_codes['Last updated date']}")
Last updated date: 2025-12-23

>>> ## Try more examples! Uncomment the lines below and run:
>>> # stn_loc_a_codes = em.fetch_locations('a')  # railway stations starting with 'A'
>>> # your_stn_loc_codes = em.fetch_locations('?')  # ... and a given letter '?'
```

< Previous | *Back to Top*

Any issues regarding the use of pyrcs are welcome and can be logged/reported onto the Issue Tracker.

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

Chapter 4

Subpackages

1anti-flashwhitewhite	
line_data	A subpackage for collecting codes of line data.
other_assets	A subpackage for collecting codes of other assets.

4.1 line_data

A subpackage for collecting codes of line data.

(See also the *LineData* class.)

1anti-flashwhitewhite	
ELRMileages([data_dir, update, verbose])	A class for collecting data of Engineer's Line References (ELRs).
Electrification([data_dir, update, verbose])	A class for collecting section codes for overhead line electrification (OLE) installations.
LocationIdentifiers([data_dir, update, verbose])	A class for collecting data of location identifiers (including CRS, NLC, TIPLOC and STANOX codes and other systems' station codes).
LOR([data_dir, update, verbose])	A class for collecting data of Line of Route (LOR/PRIDE).
LineNames([data_dir, update, verbose])	A class for collecting data of railway line names.
TrackDiagrams([data_dir, update, verbose])	A class for collecting data of British railway track diagrams.
Bridges([data_dir, update, verbose])	A class for collecting data of railway bridges.

4.1.1 ELRMileages

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

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

This class provides methods to access and manage ELR data, including their mileages and last updated information.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the data catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.
- **measure_headers** (*list*) – A list of potential headers for various measures in the data.

Examples:

```
>>> from pyrcs.line_data import ELRMileages  # from pyrcs import ELRMileages
>>> em = ELRMileages()
>>> em.NAME
"Engineer's Line References (ELRs)"
>>> em.URL
'http://www.railwaycodes.org.uk/elrs/elr0.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
NAME	The name of the data.
URL	The URL of the main web page for the data.

ELRMileages.KEY

ELRMileages.**KEY: str = 'ELRs and mileages'**

The key for accessing the data.

ELRMileages.KEY_TO_LAST_UPDATED_DATE

ELRMileages.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

ELRMileages.NAME

ELRMileages.**NAME: str = "Engineer's Line References (ELRs)"**

The name of the data.

ELRMileages.URL

ELRMileages.**URL: str = 'http://www.railwaycodes.org.uk/elrs/elr0.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_elr(initial[, ...])	Collects Engineer's Line References (ELRs) that begin with a specified initial letter from the source web page.
collect_mileage_file(elr[, parsed, ...])	Collects the mileage file for a specific ELR from the source web page.
fetch_elr([initial, update, dump_dir, verbose])	Fetches data of ELRs and their associated mileages.
fetch_mileage_file(elr[, update, dump_dir, ...])	Fetches the mileage file for a specific ELR.
get_conn_mileages(start_elr, end_elr[, update])	Retrieves the connection point between two pairs of ELRs and their associated mileages.
search_conn(start_elr, start_em, end_elr, end_em)	Searches for connections between two pairs of ELRs and their associated mileages.

ELRMileages.collect_elr

ELRMileages.**collect_elr**(*initial*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects Engineer's Line References (ELRs) that begin with a specified initial letter from the source web page.

Parameters

- **initial** (*str*) – The initial letter (e.g. `'a'`, `'z'`) of an ELR.

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `True`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing ELR data whose names start with the given initial letter, along with the date of the last update.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import ELRMileages  # from pyrcs import ELRMileages
>>> em = ELRMileages()
>>> elrs_a_codes = em.collect_elr(initial='a')
>>> type(elrs_a_codes)
dict
>>> list(elrs_a_codes.keys())
['A', 'Last updated date']
>>> elrs_a_codes_dat = elrs_a_codes['A']
>>> type(elrs_a_codes_dat)
pandas.core.frame.DataFrame
>>> elrs_a_codes_dat.head()
   ELR  ...         Notes
0  AAL  ...      Now NAJ3
1  AAM  ...  Formerly AML
2  AAV  ...
3  ABB  ...       Now AHB
4  ABB  ...
[5 rows x 5 columns]
>>> elrs_q_codes = em.collect_elr(initial='Q')
>>> elrs_q_codes_dat = elrs_q_codes['Q']
>>> elrs_q_codes_dat.head()
    ELR  ...            Notes
0   QAB  ...  Duplicates ALB?
1   QBL  ...
2   QDS  ...
3   QLT  ...
4  QLT1  ...
[5 rows x 5 columns]
```

ELRMileages.collect_mileage_file

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

Collects the mileage file for a specific ELR from the source web page.

Parameters

- **elr** (*str*) – The ELR for which the mileage file is requested (e.g. `'CJD'`, `'MLA'`, `'FED'`).
- **parsed** (*bool*) – Whether to parse the scraped mileage data; defaults to `True`.
- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **dump_dir** (*str | os.PathLike | bool | None*) – The path to a directory where the mileage file data is saved; if `False` (default), the data will not be dumped.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
A dictionary containing the mileage file for the specified ELR.

Return type
dict

Note

- In some cases, mileages may be unknown and thus 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  # from pyrcs import ELRMileages
>>> em = ELRMileages()
>>> gam_mileage_file = em.collect_mileage_file(elr='GAM')
To collect mileage file of "GAM"
? [No]|Yes: yes
>>> type(gam_mileage_file)
dict
>>> list(gam_mileage_file.keys())
['ELR', 'Line', 'Sub-Line', 'Mileage', 'Notes']
>>> gam_mileage_file['Mileage']
   Mileage Mileage_Note Miles_Chains  ... Link_1 Link_1_ELR Link_1_Mile_Chain
0   8.1518                     8.69  ...   None
1  10.0264                    10.12  ...   None
[2 rows x 8 columns]
>>> xrc2_mileage_file = em.collect_mileage_file(elr='XRC2')
To collect mileage file of "XRC2"
```

(continues on next page)

(continued from previous page)

```
? [No]|Yes: yes
>>> xrc2_mileage_file['Mileage']
  Mileage Mileage_Note  ... Link_1_ELR Link_1_Mile_Chain
0  9.0158     14.629km  ...
1  9.0447     14.893km  ...
2  9.0557     14.994km  ...
[3 rows x 8 columns]
>>> xre_mileage_file = em.collect_mileage_file(elr='XRE')
To collect mileage file of "XRE"
? [No]|Yes: yes
>>> xre_mileage_file['Mileage']
  Mileage Mileage_Note  ... Link_2_ELR Link_2_Mile_Chain
0  7.0073     11.333km  ...
1  7.0174     11.425km  ...
2  9.0158     14.629km  ...
3  9.0198     14.666km  ...
4  9.0389     14.840km  ...
5  9.0439   (14.886)km  ...
6  9.0540   (14.978)km  ...
[7 rows x 11 columns]
>>> mor_mileage_file = em.collect_mileage_file(elr='MOR')
To collect mileage file of "MOR"
? [No]|Yes: yes
>>> type(mor_mileage_file['Mileage'])
dict
>>> list(mor_mileage_file['Mileage'].keys())
['Original measure', 'Later measure']
>>> mor_mileage_file['Mileage']['Original measure']
  Mileage Mileage_Note Miles_Chains  ...        Link_1 Link_1_ELR Link_1_Mile_Chain
0  0.0000                     0.00  ...  SWA (215.18)        SWA            215.18
1  0.0792                     0.36  ...
2  0.1716                     0.78  ...
3  1.1166                     1.53  ...
4  2.0066                     2.03  ...
5  2.0836                     2.38  ...
6                                ...
7  3.0462                     3.21  ...   SDI2 (2.79)       SDI2              2.79
[8 rows x 8 columns]
>>> mor_mileage_file['Mileage']['Later measure']
  Mileage Mileage_Note Miles_Chains  ...        Link_1 Link_1_ELR Link_1_Mile_Chain
0  0.0000                     0.00  ...  SWA (215.26)        SWA            215.26
1  0.0176                     0.08  ...  SWA (215.18)        SWA            215.18
2  0.0968                     0.44  ...
3  1.0132                     1.06  ...
4  1.1342                     1.61  ...
5  2.0242                     2.11  ...
6  2.1012                     2.46  ...
7                                ...
8  3.0638                     3.29  ...   SDI2 (2.79)       SDI2              2.79
[9 rows x 8 columns]
>>> fed_mileage_file = em.collect_mileage_file(elr='FED')
To collect mileage file of "FED"
? [No]|Yes: yes
>>> type(fed_mileage_file['Mileage'])
dict
>>> list(fed_mileage_file['Mileage'].keys())
['Current measure', 'Original route']
```

(continues on next page)

(continued from previous page)

```
>>> fed_mileage_file['Mileage']['Current measure']
   Mileage Mileage_Note  ... Link_1_ELR Link_1_Mile_Chain
0  83.1254               ...        FEL
1  84.0198               ...
2  84.1430               ...
3  84.1540               ...
4  85.0484               ...
5  85.1122               ...
6  85.1188               ...        TFN              2.13
[7 rows x 8 columns]
>>> fed_mileage_file['Mileage']['Original route']
  Mileage Mileage_Note Miles_Chains  ...       Link_1 Link_1_ELR Link_1_Mile_Chain
0  0.0000                     0.00  ...  FEL (84.22)        FEL             84.22
1  1.0176                     1.08  ...
2  1.1540                     1.70  ...
3  1.1694                     1.77  ...
4                                   ...
[5 rows x 8 columns]
```

ELRMileages.fetch_elr

ELRMileages.**fetch_elr**(*initial=None*, *update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of ELRs and their associated mileages.

Parameters

- **initial** (*str | None*) – The initial letter (e.g. `'a'`, `'z'`) of an ELR; defaults to `None`.
- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **dump_dir** (*str | None*) – Path to a directory where the data file will be saved; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Returns

A dictionary containing data for all available ELRs, along with the date of the last update.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import ELRMileages  # from pyrcs import ELRMileages
>>> em = ELRMileages()
>>> elrs_codes = em.fetch_elr()
>>> type(elrs_codes)
dict
>>> list(elrs_codes.keys())
['ELRs and mileages', 'Last updated date']
>>> em.KEY
```

(continues on next page)

(continued from previous page)

```
'ELRs and mileages'
>>> elrs_codes_dat = elrs_codes[em.KEY]
>>> type(elrs_codes_dat)
pandas.core.frame.DataFrame
>>> elrs_codes_dat.head()
   ELR  ...         Notes
0  AAL  ...      Now NAJ3
1  AAM  ...  Formerly AML
2  AAV  ...
3  ABB  ...       Now AHB
4  ABB  ...
[5 rows x 5 columns]
```

ELRMileages.fetch_mileage_file

ELRMileages.**fetch_mileage_file**(*elr*, *update=False*, *dump_dir=None*, *verbose=False*, *raise_error=False*)

Fetches the mileage file for a specific ELR.

Parameters

- **elr** (*str*) – The ELR for which the mileage file is requested (e.g. `'CJD'`, `'MLA'`, `'FED'`).
- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **dump_dir** (*str | os.PathLike | None*) – Path to the directory where the data file will be saved; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the mileage file (codes), line name and any additional information or notes.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import ELRMileages  # from pyrcs import ELRMileages
>>> import tempfile
>>> import pathlib
>>> tmp_path = pathlib.Path(tempfile.TemporaryDirectory().name)
>>> em = ELRMileages()
>>> # Get the mileage file of 'AAL' (Now 'NAJ3')
>>> aal_mileage_file = em.fetch_mileage_file(elr='AAL', dump_dir=tmp_path)
>>> type(aal_mileage_file)
dict
>>> list(aal_mileage_file.keys())
```

(continues on next page)

(continued from previous page)

```
['ELR', 'Line', 'Sub-Line', 'Mileage', 'Notes', 'Formerly']
>>> aal_mileage_file['ELR']
'NAJ3'
>>> aal_mileage_file['Notes']
'Note that Ashendon Junction up line junction is on NAJ2'
>>> aal_mileage_file['Mileage']
    Mileage Mileage_Note  ... Link_1_ELR Link_1_Mile_Chain
0    0.0000               ...       NAJ2             33.69
1    0.0594               ...        GUA            164.75
2    1.0396               ...
3    3.0682               ...
4    6.0704               ...
5    8.0572               ...        BSG              0.00
6    8.0990               ...        WEJ
7    9.0594               ...
8   13.0264               ...
9   17.0858               ...
10  17.0968               ...
11  18.0572               ...        DCL             81.10
12  18.0638               ...        DCL             81.12
[13 rows x 8 columns]
>>> # Get the mileage file of 'MLA'
>>> mla_mileage_file = em.fetch_mileage_file(elr='MLA', dump_dir=tmp_path)
>>> type(mla_mileage_file)
dict
>>> list(mla_mileage_file.keys())
['ELR', 'Line', 'Sub-Line', 'Mileage', 'Notes']
>>> mla_mileage_file_mileages = mla_mileage_file['Mileage']
>>> type(mla_mileage_file_mileages)
dict
>>> list(mla_mileage_file_mileages.keys())
['Current measure', 'Original measure']
>>> mla_mileage_file_mileages['Original measure']
  Mileage Mileage_Note  ... Link_3_ELR Link_3_Mile_Chain
0  4.1386               ...       NEM4              0.00
1  5.0616               ...
2  5.1122               ...
[3 rows x 14 columns]
>>> mla_mileage_file_mileages['Current measure']
  Mileage Mileage_Note Miles_Chains  ...       Link_1 Link_1_ELR Link_1_Mile_Chain
0  0.0000                     0.00  ...  MRL2 (4.44)       MRL2              4.44
1  0.0572                     0.26  ...         None
2  0.1540                     0.70  ...         None
3  0.1606                     0.73  ...         None
[4 rows x 8 columns]
>>> # Get the mileage file of 'LCG'
>>> mla_mileage_file = em.fetch_mileage_file(elr='LCG', dump_dir=tmp_path)
```

ELRMileages.get_conn_mileages

ELRMileages.**get_conn_mileages**(*start_elr*, *end_elr*, *update=False*, ***kwargs*)

Retrieves the connection point between two pairs of ELRs and their associated mileages.

Specifically, it finds the end mileage for the starting ELR and the start mileage for the ending ELR.

Note

This function may not be able to find a connection for every pair of ELRs. Please refer to *Example 2* for more information.

Parameters

- **start_elr** (*str*) – The starting ELR.
- **end_elr** (*str*) – The ending ELR.
- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **kwargs** – [Optional] Additional parameters for the method `fetch_mileage_file()`.

Returns

A tuple containing the connection ELR(s) and mileage(s) between the specified `start_elr` and `end_elr`.

Return type

tuple

Example 1:

```
>>> from pyrcs.line_data import ELRMileages  # from pyrcs import ELRMileages
>>> em = ELRMileages()
>>> conn = em.get_conn_mileages(start_elr='NAY', end_elr='LTN2')
>>> (s_dest_mlg, c_elr, c_orig_mlg, c_dest_mlg, e_orig_mlg) = conn
>>> s_dest_mlg
'5.1606'
>>> c_elr
'NOL'
>>> c_orig_mlg
'5.1606'
>>> c_dest_mlg
'0.0638'
>>> e_orig_mlg
'123.1320'
```

Example 2:

```
>>> from pyrcs.line_data import ELRMileages  # from pyrcs import ELRMileages
>>> em = ELRMileages()
>>> conn = em.get_conn_mileages(start_elr='MAC3', end_elr='DBP1', dump_dir="tests")
>>> conn
('', '', '', '', '')
```

ELRMileages.search_conn

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

Searches for connections between two pairs of ELRs and their associated mileages.

Parameters

- **start_elr** (*str*) – The starting ELR.
- **start_em** (*pandas.DataFrame*) – The mileage file associated with the starting ELR.
- **end_elr** (*str*) – The ending ELR.
- **end_em** (*pandas.DataFrame*) – The mileage file associated with the ending ELR.

Returns

A tuple containing the end mileage of the starting ELR and the start mileage of the ending ELR.

Return type

tuple

Examples:

```
>>> from pyrcs.line_data import ELRMileages  # from pyrcs import ELRMileages
>>> em = ELRMileages()
>>> elr_1 = 'AAM'
>>> mileage_file_1 = em.collect_mileage_file(elr_1, confirmation_required=False)
>>> mf_1_mileages = mileage_file_1['Mileage']
>>> mf_1_mileages.head()
  Mileage Mileage_Note  ... Link_2_ELR Link_2_Mile_Chain
0  0.0000               ...
1  0.0154               ...
2  0.0396               ...
3  1.1012               ...
4  1.1408               ...
[5 rows x 11 columns]
>>> elr_2 = 'ANZ'
>>> mileage_file_2 = em.collect_mileage_file(elr_2, confirmation_required=False)
>>> mf_2_mileages = mileage_file_2['Mileage']
>>> mf_2_mileages.head()
   Mileage Mileage_Note Miles_Chains  ...      Link_1 Link_1_ELR Link_1_Mile_Chain
0  84.0924                    84.42  ...         BEA        BEA
1  84.1364                    84.62  ...  AAM (0.18)        AAM              0.18
[2 rows x 8 columns]
>>> elr_1_dest, elr_2_orig = em.search_conn(elr_1, mf_1_mileages, elr_2, mf_2_mileages)
>>> elr_1_dest
'0.0396'
>>> elr_2_orig
'84.1364'
```

4.1.2 Electrification

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

A class for collecting section codes for overhead line electrification (OLE) installations.

Parameters

- **data_dir** (*str | None*) – Name of the data directory; defaults to None.
- **update** (*bool*) – Whether to check for updates to the data catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.line_data import Electrification  # from pyrcs import Electrification
>>> elec = Electrification()
>>> elec.NAME
'Section codes for overhead line electrification (OLE) installations'
>>> elec.URL
'http://www.railwaycodes.org.uk/electrification/mast_prefix0.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_ETZ	The key used to reference the data of the '*UK railway electrification tariff zones*'.
KEY_TO_INDEPENDENT_LINES	The key used to reference the data of the '*independent lines*'.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
KEY_TO_NATIONAL_NETWORK	The key used to reference the data of the '*national network*'.
KEY_TO_OHNS	The key used to reference the data of the '*overhead line electrification neutral sections (OHNS)*'.
NAME	The name of the data.
URL	The URL of the main web page for the data.

Electrification.KEY

Electrification.**KEY: str = 'Electrification'**

The key for accessing the data.

Electrification.KEY_TO_ETZ

Electrification.**KEY_TO_ETZ: str = 'National network energy tariff zones'**

The key used to reference the data of the *'UK railway electrification tariff zones'*.

Electrification.KEY_TO_INDEPENDENT_LINES

Electrification.**KEY_TO_INDEPENDENT_LINES: str = 'Independent lines'**

The key used to reference the data of the *'independent lines'*.

Electrification.KEY_TO_LAST_UPDATED_DATE

Electrification.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

Electrification.KEY_TO_NATIONAL_NETWORK

Electrification.**KEY_TO_NATIONAL_NETWORK: str = 'National network'**

The key used to reference the data of the *'national network'*.

Electrification.KEY_TO_OHNS

Electrification.**KEY_TO_OHNS: str = 'National network neutral sections'**

The key used to reference the data of the *'overhead line electrification neutral sections (OHNS)'*.

Electrification.NAME

Electrification.**NAME: str = 'Section codes for overhead line electrification (OLE) installations'**

The name of the data.

Electrification.URL

Electrification.**URL: str = 'http://www.railwaycodes.org.uk/electrification/mast_prefix0.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_etz_codes([confirmation_required, ...])	Collects OLE section codes for national network energy tariff zones from the source web page.
collect_independent_lines_codes([...])	Collects OLE section codes for independent lines from the source web page.
collect_national_network_codes([...])	Collects the section codes for Overhead Line Electrification (OLE) installations for the national network from the source web page.
collect_ohns_codes([confirmation_required, ...])	Collects codes for overhead line electrification neutral sections (OHNS) from the source web page.
fetch_codes([update, dump_dir, verbose])	Fetches OLE section codes listed in the *Electrification* catalogue.
fetch_etz_codes([update, dump_dir, verbose])	Fetches OLE section codes for national network energy tariff zones.
fetch_independent_lines_codes([update, ...])	Fetches OLE section codes for independent lines.
fetch_national_network_codes([update, ...])	Fetches section codes for Overhead Line Electrification (OLE) installations on the national network.
fetch_ohns_codes([update, dump_dir, verbose])	Fetches the overhead line electrification neutral sections (OHNS) codes.
get_independent_lines_catalogue([update ...])	Gets the catalogue for the independent lines.

Electrification.collect_etz_codes

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

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

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary of OLE section codes for national network energy tariff zones.

Return type

dict | None

Examples:

```
>>> from pyrcs.line_data import Electrification  # from pyrcs import Electrification
>>> elec = Electrification()
>>> rail_etz_codes = elec.collect_etz_codes(verbose=True)
To collect section codes for OLE installations: National network energy tariff zones
? [No]|Yes: yes
Collecting the data ... Done.
>>> type(rail_etz_codes)
dict
>>> list(rail_etz_codes.keys())
['National network energy tariff zones', 'Last updated date']
>>> elec.KEY_TO_ETZ
'National network energy tariff zones'
>>> rail_etz_codes_dat = rail_etz_codes[elec.KEY_TO_ETZ]
>>> type(rail_etz_codes_dat)
dict
>>> list(rail_etz_codes_dat.keys())
['Railtrack', 'Network Rail']
>>> rail_etz_codes_dat['Railtrack']['Codes']
   Code                    Energy tariff zone
0    EA                            East Anglia
1    EC                  East Coast Main Line
2    GE                       Great Eastern †
3    LT                                  LTS †
4    MD                     Midland Main Line
5    ME                           Merseyside †
6    MS  Merseyside (North West DC traction)
7    NE                              North East
8    NL           North London (DC traction)
9    SC                                Scotland
10   SO                                   South
11   SW                              South West
12   WA                            West Anglia †
13   WC                 West Coast/North West
```

Electrification.collect_independent_lines_codes

Electrification.**collect_independent_lines_codes**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects OLE section codes for independent lines from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary of OLE section codes for independent lines, or `None` if not applicable.

Return type
dict | None

Examples:

```
>>> from pyrcs.line_data import Electrification  # from pyrcs import Electrification
>>> elec = Electrification()
>>> indep_lines_codes = elec.collect_independent_lines_codes(verbose=True)
To collect section codes for OLE installations: Independent lines
? [No]|Yes: yes
Collecting the data ... Done.
>>> type(indep_lines_codes)
dict
>>> list(indep_lines_codes.keys())
['Independent lines', 'Last updated date']
>>> elec.KEY_TO_INDEPENDENT_LINES
'Independent lines'
>>> indep_lines_codes_dat = indep_lines_codes[elec.KEY_TO_INDEPENDENT_LINES]
>>> type(indep_lines_codes_dat)
dict
>>> len(indep_lines_codes_dat)
23
>>> list(indep_lines_codes_dat.keys())[-5:]
['Sheffield Supertram',
 'Snaefell Mountain Railway',
 'Summerlee, Museum of Scottish Industrial Life Tramway',
 'Tyne & Wear Metro',
 'West Midlands Metro [West Midlands]']
>>> indep_lines_codes_dat['Summerlee, Museum of Scottish Industrial Life Tramway']
{'Codes': None, 'Notes': 'Masts do not carry any labels.'}
```

Electrification.collect_national_network_codes

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

Collects the section codes for Overhead Line Electrification (OLE) installations for the national network from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
A dictionary of OLE section codes for the national network, or `None` if not applicable.

Return type
dict | None

Examples:

```
>>> from pyrcs.line_data import Electrification  # from pyrcs import Electrification
>>> elec = Electrification()
>>> nn_codes = elec.collect_national_network_codes(verbose=True)
To collect section codes for OLE installations: National network
? [No]|Yes: yes
Collecting the data ... Done.
>>> type(nn_codes)
dict
>>> list(nn_codes.keys())
['National network', 'Last updated date']
>>> elec.KEY_TO_NATIONAL_NETWORK
'National network'
>>> nn_codes_dat = nn_codes[elec.KEY_TO_NATIONAL_NETWORK]
>>> type(nn_codes_dat)
dict
>>> list(nn_codes_dat.keys())
['Traditional numbering system [distance and sequence]',
 'New numbering system [km and decimal]',
 'Codes not certain [confirmation is welcome]',
 'Suspicious data',
 'An odd one to complete the record',
 'LBSC/Southern Railway overhead system',
 'Codes not known']
>>> tns_codes = nn_codes_dat['Traditional numbering system [distance and sequence]']
>>> type(tns_codes)
dict
>>> list(tns_codes.keys())
['Codes', 'Notes']
>>> tns_codes_dat = tns_codes['Codes']
>>> tns_codes_dat.head()
  Code  ...                           Datum
0    A  ...               Fenchurch Street
1    A  ...             Newbridge Junction
2    A  ...               Fenchurch Street
3    A  ...  Guide Bridge Station Junction
4   AB  ...
[5 rows x 4 columns]
```

Electrification.collect_ohns_codes

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

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

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if

raise_error=False (default), the error will be suppressed.

Returns
A dictionary of OHNS codes, or None if not applicable.

Return type
dict | None

Examples:

```
>>> from pyrcs.line_data import Electrification  # from pyrcs import Electrification
>>> elec = Electrification()
>>> ohl_ns_codes = elec.collect_ohns_codes(verbose=True)
To collect section codes for OLE installations: National network neutral sections
? [No]|Yes: yes
Collecting the data ... Done.
>>> type(ohl_ns_codes)
dict
>>> list(ohl_ns_codes.keys())
['National network neutral sections', 'Last updated date']
>>> elec.KEY_TO_OHNS
'National network neutral sections'
>>> ohl_ns_codes_dat = ohl_ns_codes[elec.KEY_TO_OHNS]
>>> type(ohl_ns_codes_dat)
dict
>>> list(ohl_ns_codes_dat.keys())
['Codes', 'Notes']
>>> ohl_ns_codes_dat['Codes'].head()
    ELR         OHNS Name  Mileage    Tracks Dates
0  ARG1        Rutherglen  0m 03ch
1  ARG2   Finnieston East  4m 23ch      Down
2  ARG2   Finnieston West  4m 57ch        Up
3  AYR1  Shields Junction  0m 68ch    Up Ayr
4  AYR1  Shields Junction  0m 69ch  Down Ayr
```

Electrification.fetch_codes

Electrification.**fetch_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches OLE section codes listed in the Electrification catalogue.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – Path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns
A dictionary of the section codes for OLE installations.

Return type
dict

Examples:

```
>>> from pyrcs.line_data import Electrification  # from pyrcs import Electrification
>>> elec = Electrification()
>>> elec_codes = elec.fetch_codes()
>>> type(elec_codes)
dict
>>> list(elec_codes.keys())
['Electrification', 'Last updated date']
>>> elec.KEY
'Electrification'
>>> elec_codes_dat = elec_codes[elec.KEY]
>>> type(elec_codes_dat)
dict
>>> list(elec_codes_dat.keys())
['National network energy tariff zones',
 'Independent lines',
 'National network',
 'National network neutral sections']
```

Electrification.fetch_etz_codes

`Electrification.`**`fetch_etz_codes`**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches OLE section codes for national network energy tariff zones.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **dump_dir** (*str | None*) – Path to a directory where the data file will be saved; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Returns

A dictionary of OLE section codes for national network energy tariff zones.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import Electrification  # from pyrcs import Electrification
>>> elec = Electrification()
>>> rail_etz_codes = elec.fetch_etz_codes()
>>> type(rail_etz_codes)
dict
>>> list(rail_etz_codes.keys())
['National network energy tariff zones', 'Last updated date']
>>> elec.KEY_TO_ETZ
'National network energy tariff zones'
>>> rail_etz_codes_dat = rail_etz_codes[elec.KEY_TO_ETZ]
>>> type(rail_etz_codes_dat)
dict
>>> list(rail_etz_codes_dat.keys())
['Railtrack', 'Network Rail']
```

(continues on next page)

(continued from previous page)

```
>>> rail_etz_codes_dat['Railtrack']['Codes']
   Code                    Energy tariff zone
0    EA                             East Anglia
1    EC                  East Coast Main Line
2    GE                       Great Eastern †
3    LT                                  LTS †
4    MD                     Midland Main Line
5    ME                           Merseyside †
6    MS  Merseyside (North West DC traction)
7    NE                              North East
8    NL            North London (DC traction)
9    SC                                Scotland
10   SO                                   South
11   SW                              South West
12   WA                            West Anglia †
13   WC                  West Coast/North West
```

Electrification.fetch_independent_lines_codes

Electrification.**fetch_independent_lines_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches OLE section codes for independent lines.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – Path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary of OLE section codes for independent lines.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import Electrification  # from pyrcs import Electrification
>>> elec = Electrification()
>>> indep_lines_codes = elec.fetch_independent_lines_codes()
>>> type(indep_lines_codes)
dict
>>> list(indep_lines_codes.keys())
['Independent lines', 'Last updated date']
>>> elec.KEY_TO_INDEPENDENT_LINES
'Independent lines'
>>> indep_lines_codes_dat = indep_lines_codes[elec.KEY_TO_INDEPENDENT_LINES]
>>> type(indep_lines_codes_dat)
dict
>>> len(indep_lines_codes_dat)
```

(continues on next page)

(continued from previous page)

```
22
>>> list(indep_lines_codes_dat.keys())
['Beamish Tramway',
 'Birkenhead Tramway',
 'Black Country Living Museum [Tipton]',
 'Blackpool Tramway',
 "Brighton and Rottingdean Seashore Electric Railway [Magnus Volk's 'Daddy Long Legs'...
 'Channel Tunnel',
 'Croydon Tramlink',
 'East Anglia Transport Museum [Lowestoft]',
 'Edinburgh Tramway',
 'Heath Park Tramway [Cardiff]',
 'Heaton Park Tramway [Manchester]',
 'Iarnród Éireann',
 'Luas [Dublin]',
 'Manchester Metrolink',
 'Manx Electric Railway',
 'Nottingham Express Transit',
 'Seaton Tramway',
 'Sheffield Supertram',
 'Snaefell Mountain Railway',
 'Summerlee, Museum of Scottish Industrial Life Tramway',
 'Tyne & Wear Metro',
 'West Midlands Metro [West Midlands]']
>>> indep_lines_codes_dat['Beamish Tramway']
{'Codes': None, 'Notes': 'Masts do not appear labelled.'}
```

Electrification.fetch_national_network_codes

Electrification.**fetch_national_network_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches section codes for Overhead Line Electrification (OLE) installations on the national network.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – Path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary of OLE section codes for the national network, or None if not applicable.

Return type

dict | None

Examples:

```
>>> from pyrcs.line_data import Electrification  # from pyrcs import Electrification
>>> elec = Electrification()
>>> nn_codes = elec.fetch_national_network_codes()
>>> type(nn_codes)
dict
>>> list(nn_codes.keys())
['National network', 'Last updated date']
>>> elec.KEY_TO_NATIONAL_NETWORK
'National network'
>>> nn_codes_dat = nn_codes[elec.KEY_TO_NATIONAL_NETWORK]
>>> type(nn_codes_dat)
dict
>>> list(nn_codes_dat.keys())
['Traditional numbering system [distance and sequence]',
 'New numbering system [km and decimal]',
 'Codes not certain [confirmation is welcome]',
 'Suspicious data',
 'An odd one to complete the record',
 'LBSC/Southern Railway overhead system',
 'Codes not known']
>>> tns_codes = nn_codes_dat['Traditional numbering system [distance and sequence]']
>>> type(tns_codes)
dict
>>> list(tns_codes.keys())
['Codes', 'Notes']
>>> tns_codes_dat = tns_codes['Codes']
>>> tns_codes_dat.head()
  Code  ...                           Datum
0    A  ...               Fenchurch Street
1    A  ...             Newbridge Junction
2    A  ...               Fenchurch Street
3    A  ...  Guide Bridge Station Junction
4   AB  ...
[5 rows x 4 columns]
```

Electrification.fetch_ohns_codes

Electrification.**fetch_ohns_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches the overhead line electrification neutral sections (OHNS) codes.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – Path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary of OHNS codes.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import Electrification  # from pyrcs import Electrification
>>> elec = Electrification()
>>> ohl_ns_codes = elec.fetch_ohns_codes()
>>> type(ohl_ns_codes)
dict
>>> list(ohl_ns_codes.keys())
['National network neutral sections', 'Last updated date']
>>> elec.KEY_TO_OHNS
'National network neutral sections'
>>> ohl_ns_codes_dat = ohl_ns_codes[elec.KEY_TO_OHNS]
>>> type(ohl_ns_codes_dat)
dict
>>> list(ohl_ns_codes_dat.keys())
['Codes', 'Notes']
>>> ohl_ns_codes_dat['Codes'].head()
    ELR         OHNS Name  Mileage    Tracks Dates
0  ARG1        Rutherglen  0m 03ch
1  ARG2   Finnieston East  4m 23ch      Down
2  ARG2   Finnieston West  4m 57ch        Up
3  AYR1  Shields Junction  0m 68ch    Up Ayr
4  AYR1  Shields Junction  0m 69ch  Down Ayr
```

Electrification.get_independent_lines_catalogue

Electrification.**get_independent_lines_catalogue**(*update=False*, *verbose=False*)

Gets the catalogue for the independent lines.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A pandas DataFrame containing the names of independent lines.

Return type

pandas.DataFrame

Examples:

```
>>> from pyrcs.line_data import Electrification  # from pyrcs import Electrification
>>> from pyhelpers.settings import pd_preferences
>>> pd_preferences(max_columns=1)
>>> elec = Electrification()
>>> indep_line_cat = elec.get_independent_lines_catalogue()
>>> indep_line_cat.head()
                                          Feature  ...
0                                 Beamish Tramway  ...
1                               Birkenhead Tramway  ...
2                     Black Country Living Museum  ...
3                               Blackpool Tramway  ...
```

(continues on next page)

(continued from previous page)

```
4  Brighton and Rottingdean Seashore Electric Rai...  ...
[5 rows x 3 columns]
```

4.1.3 LocationIdentifiers

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

A class for collecting data of location identifiers (including CRS, NLC, TIPLOC and STANOX codes and other systems' station codes).

The class retrieves and organises location identifiers for various railway stations and other related systems.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the data catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.line_data import LocationIdentifiers
>>> # from pyrcs import LocationIdentifiers
>>> lid = LocationIdentifiers()
>>> lid.NAME
'CRS, NLC, TIPLOC and STANOX codes'
>>> lid.URL
'http://www.railwaycodes.org.uk/crs/crs0.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
KEY_TO_MSCEN	The key for accessing the data of *multiple station codes explanatory note*.

continues on next page

Table 7 – continued from previous page

KEY_TO_NOTES	The key for accessing the data of *additional notes*.
KEY_TO_OTHER_SYSTEMS	The key for accessing the data of *other systems*.
NAME	The name of the data.
URL	The URL of the main web page for the data.

LocationIdentifiers.KEY

LocationIdentifiers.**KEY: str = 'Location ID'**

The key for accessing the data.

LocationIdentifiers.KEY_TO_LAST_UPDATED_DATE

LocationIdentifiers.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

LocationIdentifiers.KEY_TO_MSCEN

LocationIdentifiers.**KEY_TO_MSCEN: str = 'Multiple station codes explanatory note'**

The key for accessing the data of *multiple station codes explanatory note*.

LocationIdentifiers.KEY_TO_NOTES

LocationIdentifiers.**KEY_TO_NOTES: str = 'Notes'**

The key for accessing the data of *additional notes*.

LocationIdentifiers.KEY_TO_OTHER_SYSTEMS

LocationIdentifiers.**KEY_TO_OTHER_SYSTEMS: str = 'Other systems'**

The key for accessing the data of *other systems*.

LocationIdentifiers.NAME

LocationIdentifiers.**NAME: str = 'CRS, NLC, TIPLOC and STANOX codes'**

The name of the data.

LocationIdentifiers.URL

LocationIdentifiers.**URL: str = 'http://www.railwaycodes.org.uk/crs/crs0.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_loc_id(initial[, ...])	Collects CRS, NLC, TIPLOC, STANME and STANOX codes for a given initial letter.
collect_notes([confirmation_required, ...])	Collects the explanatory note related to multiple station codes (CRS codes) from the source web page.
collect_other_systems_codes([...])	Collects data of other systems' station codes from the source web page.
fetch_codes([update, dump_dir, verbose])	Fetches location codes listed in the CRS, NLC, TIPLOC and STANOX codes catalogue (including other systems' station codes).
fetch_loc_id([initial, update, dump_dir, ...])	Fetches data of CRS, NLC, TIPLOC, STANME and STANOX codes.
fetch_notes([update, dump_dir, verbose])	Fetches the explanatory note for multiple station codes (CRS codes).
fetch_other_systems_codes([update, ...])	Fetches data of other systems' station codes.
make_xref_dict(keys[, initials, ...])	Creates a dictionary or dataframe containing location code data for the specified `keys`.

LocationIdentifiers.collect_loc_id

`LocationIdentifiers.`**`collect_loc_id`**(*initial*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

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

Parameters

- **`initial`** (*`str`*) – The initial letter (e.g. `'a'`, `'z'`) of a location name.
- **`confirmation_required`** (*`bool`*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **`verbose`** (*`bool | int`*) – Whether to print relevant information to the console; defaults to `False`.
- **`raise_error`** (*`bool`*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing data of locations whose names start with the given initial letter, along with the date of the last update.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import LocationIdentifiers
>>> # from pyrcs import LocationIdentifiers
>>> lid = LocationIdentifiers()
>>> loc_a_codes = lid.collect_loc_id(initial='a')
```

(continues on next page)

(continued from previous page)

```
To collect data of CRS, NLC, TIPLOC and STANOX codes beginning with "A"
? [No]|Yes: yes
>>> type(loc_a_codes)
dict
>>> list(loc_a_codes.keys())
['A', 'Additional notes', 'Last updated date']
>>> loc_a_codes_dat = loc_a_codes['A']
>>> type(loc_a_codes_dat)
pandas.core.frame.DataFrame
>>> loc_a_codes_dat.head()
                              Location CRS  ... STANME_Note STANOX_Note
0                1999 Reorganisations      ...
1                                    A1      ...
2                       A463 Traded In      ...
3  A483 Road Scheme Supervisors Closed      ...
4                                Aachen      ...
[5 rows x 12 columns]
```

LocationIdentifiers.collect_notes

LocationIdentifiers.**collect_notes**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects the explanatory note related to multiple station codes (CRS codes) from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the data of the multiple station codes explanatory note, or `None` if the note is not found or the collection is not performed.

Return type

dict | None

Examples:

```
>>> from pyrcs.line_data import LocationIdentifiers
>>> # from pyrcs import LocationIdentifiers
>>> lid = LocationIdentifiers()
>>> notes = lid.collect_notes()
To collect data of multiple station codes explanatory note
? [No]|Yes: yes
>>> type(notes)
dict
```

(continues on next page)

(continued from previous page)

```
>>> list(notes.keys())
['Notes', 'Last updated date']
>>> lid.KEY_TO_NOTES
'Notes'
>>> notes_ = notes[lid.KEY_TO_NOTES]
>>> type(notes_)
dict
>>> notes_[lid.KEY_TO_MSCEN][2].head()
             location_name CRS1 CRS2 CRS3
0                Bletchley  BLY  BLU
1  Ebbsfleet International  EBD  EBF
2          Glasgow Central  GLC  GCL
3     Glasgow Queen Street  GLQ  GQL
4                  Heworth  HEW  HEZ
```

LocationIdentifiers.collect_other_systems_codes

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

Collects data of other systems' station codes from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required before proceeding; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; defaults to `True`. if `raise_error=False`, the error will be suppressed.

Returns

A dictionary containing station codes for other systems, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.line_data import LocationIdentifiers
>>> # from pyrcs import LocationIdentifiers
>>> lid = LocationIdentifiers()
>>> os_codes = lid.collect_other_systems_codes()
To collect data of Other systems
? [No]|Yes: yes
>>> type(os_codes)
dict
>>> list(os_codes.keys())
['Other systems', 'Last updated date']
>>> lid.KEY_TO_OTHER_SYSTEMS
'Other systems'
>>> os_codes_dat = os_codes[lid.KEY_TO_OTHER_SYSTEMS]
>>> type(os_codes_dat)
```

(continues on next page)

(continued from previous page)

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

LocationIdentifiers.fetch_codes

LocationIdentifiers.**fetch_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches location codes listed in the CRS, NLC, TIPLOC and STANOX codes catalogue (including other systems' station codes).

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing location codes and date of when the data was last updated.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import LocationIdentifiers
>>> # from pyrcs import LocationIdentifiers
>>> lid = LocationIdentifiers()
>>> loc_codes = lid.fetch_codes()
>>> type(loc_codes)
dict
>>> list(loc_codes.keys())
['Location ID', 'Other systems', 'Additional notes', 'Last updated date']
>>> lid.KEY
'LocationID'
>>> loc_codes_dat = loc_codes[lid.KEY]
>>> type(loc_codes_dat)
pandas.core.frame.DataFrame
>>> loc_codes_dat.head()
                             Location CRS  ... STANME_Note STANOX_Note
0                1999 Reorganisations      ...
1                                  A1      ...
2                      A463 Traded In      ...
3  A483 Road Scheme Supervisors Closed      ...
```

(continues on next page)

(continued from previous page)

```
4                              Aachen      ...
[5 rows x 12 columns]
```

LocationIdentifiers.fetch_loc_id

LocationIdentifiers.**fetch_loc_id**(*initial=None*, *update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of CRS, NLC, TIPLOC, STANME and STANOX codes.

Parameters

- **initial** (*str*) – The initial letter (e.g. `'a'`, `'z'`) of a location name.
- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **dump_dir** (*str | None*) – Path to a directory where the data file will be saved; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `True`.

Returns

A dictionary containing data of locations whose names start with the given initial letter, along with the date of the last update.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import LocationIdentifiers
>>> # from pyrcs import LocationIdentifiers
>>> lid = LocationIdentifiers()
>>> loc_a_codes = lid.fetch_loc_id(initial='a')
>>> type(loc_a_codes)
dict
>>> list(loc_a_codes.keys())
['A', 'Additional notes', 'Last updated date']
>>> loc_a_codes_dat = loc_a_codes['A']
>>> type(loc_a_codes_dat)
pandas.core.frame.DataFrame
>>> loc_a_codes_dat.head()
                            Location CRS  ... STANME_Note STANOX_Note
0                 1999 Reorganisations      ...
1                                   A1      ...
2                       A463 Traded In      ...
3  A483 Road Scheme Supervisors Closed      ...
4                              Aachen      ...
[5 rows x 12 columns]
>>> loc_codes = lid.fetch_loc_id()
>>> list(loc_codes.keys())
['Location ID', 'Last updated date']
>>> loc_codes[lid.KEY].shape
(59873, 12)
```

LocationIdentifiers.fetch_notes

LocationIdentifiers.**fetch_notes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches the explanatory note for multiple station codes (CRS codes).

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the data of the multiple station codes explanatory note.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import LocationIdentifiers
>>> # from pyrcs import LocationIdentifiers
>>> lid = LocationIdentifiers()
>>> exp_note = lid.fetch_notes()
>>> type(exp_note)
dict
>>> list(exp_note.keys())
['Multiple station codes explanatory note', 'Notes', 'Last updated date']
>>> lid.KEY_TO_MSCEN
'Multiple station codes explanatory note'
>>> exp_note_dat = exp_note[lid.KEY_TO_MSCEN]
>>> type(exp_note_dat)
pandas.core.frame.DataFrame
>>> exp_note_dat.head()
                Location  CRS CRS_alt1 CRS_alt2
0         Glasgow Central  GLC      GCL
1    Glasgow Queen Street  GLQ      GQL
2                Heworth  HEW      HEZ
3    Highbury & Islington  HHY      HII      XHZ
4  Lichfield Trent Valley  LTV      LIF
```

LocationIdentifiers.fetch_other_systems_codes

LocationIdentifiers.**fetch_other_systems_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of other systems' station codes.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.

- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns
A dictionary containing station codes for other systems.

Return type
dict

Examples:

```
>>> from pyrcs.line_data import LocationIdentifiers
>>> # from pyrcs import LocationIdentifiers
>>> lid = LocationIdentifiers()
>>> os_codes = lid.fetch_other_systems_codes()
>>> type(os_codes)
dict
>>> list(os_codes.keys())
['Other systems', 'Last updated date']
>>> lid.KEY_TO_OTHER_SYSTEMS
'Other systems'
>>> os_codes_dat = os_codes[lid.KEY_TO_OTHER_SYSTEMS]
>>> type(os_codes_dat)
collections.defaultdict
>>> list(os_codes_dat.keys())
['Córas Iompair Éireann (Republic of Ireland)',
 'Crossrail',
 'Croydon Tramlink',
 'Docklands Light Railway',
 'Manchester Metrolink',
 'Translink (Northern Ireland)',
 'Tyne & Wear Metro']
```

LocationIdentifiers.make_xref_dict

LocationIdentifiers.**make_xref_dict**(*keys*, *initials=None*, *drop_duplicates=False*, *as_dict=False*, *main_key=None*, *update=False*, *dump_it=False*, *dump_dir=None*, *verbose=False*)

Creates a dictionary or dataframe containing location code data for the specified keys.

Parameters

- **keys** (*str | list*) – A string or list of keys from ['CRS', 'NLC', 'TIPLOC', 'STANOX', 'STANME'] representing the location codes.
- **initials** (*str | list | None*) – A string or list of initials for which the codes are filtered; defaults to None.
- **drop_duplicates** (*bool*) – If True, removes duplicate entries from the result; defaults to False.
- **as_dict** (*bool*) – If True, returns the result as a dictionary; otherwise, returns a dataframe; defaults to False.

- **main_key** (*str | None*) – The key used for the returned dictionary when as_dict=True; defaults to None.
- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_it** (*bool*) – If True, saves the result to a file; defaults to False.
- **dump_dir** (*str | None*) – The directory path where the file can be saved, if dump_it=True; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary or dataframe containing cross-reference location data for the specified keys, or None if no data is available.

Return type

dict | pandas.DataFrame | None

Examples:

```
>>> from pyrcs.line_data import LocationIdentifiers
>>> # from pyrcs import LocationIdentifiers
>>> lid = LocationIdentifiers()
>>> stanox_dict = lid.make_xref_dict(keys='STANOX')
>>> type(stanox_dict)
pandas.core.frame.DataFrame
>>> stanox_dict.head()
                         Location
STANOX
00005                       Aachen
04309           Abbeyhill Junction
04311        Abbeyhill Signal E811
04308   Abbeyhill Turnback Sidings
88601                   Abbey Wood
>>> s_t_dictionary = lid.make_xref_dict(keys=['STANOX', 'TIPLOC'], initials='a')
>>> type(s_t_dictionary)
pandas.core.frame.DataFrame
>>> s_t_dictionary.head()
                                  Location
STANOX TIPLOC
00005  AACHEN                       Aachen
04309  ABHLJN           Abbeyhill Junction
04311  ABHL811       Abbeyhill Signal E811
04308  ABHLTB   Abbeyhill Turnback Sidings
88601  ABWD                     Abbey Wood
>>> ks = ['STANOX', 'TIPLOC']
>>> ini = 'b'
>>> main_k = 'Data'
>>> s_t_dictionary = lid.make_xref_dict(ks, ini, main_k, as_dict=True)
>>> type(s_t_dictionary)
dict
>>> list(s_t_dictionary.keys())
['Data']
>>> list(s_t_dictionary['Data'].keys())[:5]
[('55115', ''),
```

(continues on next page)

(continued from previous page)

```
('23490', 'BABWTHL'),
('38306', 'BACHE'),
('66021', 'BADESCL'),
('81003', 'BADMTN')]
```

4.1.4 LOR

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

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

> **Note**
>
> 'LOR' and 'PRIDE' stands for 'Line Of Route' and 'Possession Resource Information Database', respectively.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the data catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.
- **valid_prefixes** (*list*) – A list of valid prefixes.

Examples:

```
>>> from pyrcs.line_data import LOR  # from pyrcs import LOR
>>> lor = LOR()
>>> lor.NAME
'Possession Resource Information Database (PRIDE)/Line Of Route (LOR) codes'
>>> lor.URL
'http://www.railwaycodes.org.uk/pride/pride0.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_ELC	The key for accessing the data of *ELR/LOR converter*.
KEY_P	The key for accessing the data of *prefixes*.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
NAME	The name of the data.
SHORT_NAME	A short name of the data
URL	The URL of the main web page for the data.

LOR.KEY

LOR.**KEY: str = 'LOR'**

The key for accessing the data.

LOR.KEY_ELC

LOR.**KEY_ELC: str = 'ELR/LOR converter'**

The key for accessing the data of *ELR/LOR converter*.

LOR.KEY_P

LOR.**KEY_P: str = 'Key to prefixes'**

The key for accessing the data of *prefixes*.

LOR.KEY_TO_LAST_UPDATED_DATE

LOR.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

LOR.NAME

LOR.**NAME: str = 'Possession Resource Information Database (PRIDE)/Line Of Route (LOR) codes'**

The name of the data.

LOR.SHORT_NAME

LOR.**SHORT_NAME: str = 'Line of Route (LOR/PRIDE) codes'**

A short name of the data

LOR.URL

LOR.**URL: str = 'http://www.railwaycodes.org.uk/pride/pride0.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_codes(prefix[, ...])	Collects data of PRIDE/LOR codes for the given prefix.
collect_elr_lor_converter([...])	Collects data of ELR/LOR converter from the source web page.
collect_keys_to_prefixes([...])	Collects the keys to PRIDE/LOR code prefixes from the source web page.
collect_page_urls([confirmation_required, ...])	Collects a list of URLs to PRIDE/LOR codes web pages.
fetch_codes([prefix, update, dump_dir, verbose])	Fetches data of PRIDE/LOR codes.
fetch_elr_lor_converter([update, dump_dir, ...])	Fetches data of ELR/LOR converter.
get_keys_to_prefixes([prefixes_only, ...])	Gets the keys to PRIDE/LOR code prefixes.
get_page_urls([update, dump_dir, verbose])	Gets a list of URLs to PRIDE/LOR codes web pages.
get_url(prefix)	Generates the URL for the given PRIDE/LOR code prefix.
validate_prefix(prefix)	Validates and standardises a PRIDE/LOR code prefix.

LOR.collect_codes

LOR.**collect_codes**(*prefix*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects data of PRIDE/LOR codes for the given prefix.

Parameters

- **prefix** (*str*) – The prefix of LOR codes to collect.
- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the LOR codes for the given `prefix`, or `None` if no data is available.

Return type
dict | None

Examples:

```
>>> from pyrcs.line_data import LOR  # from pyrcs import LOR
>>> lor = LOR()
>>> lor_codes_cy = lor.collect_codes(prefix='CY')
>>> type(lor_codes_cy)
dict
>>> list(lor_codes_cy.keys())
['CY', 'Notes', 'Last updated date']
>>> cy_codes = lor_codes_cy['CY']
>>> type(cy_codes)
pandas.core.frame.DataFrame
>>> cy_codes.head()
     Code  ...                         RA Note
0   CY240  ...           Caerwent branch RA4
1  CY1540  ...  Pembroke - Pembroke Dock RA6
[2 rows x 5 columns]
>>> lor_codes_nw = lor.collect_codes(prefix='NW')
>>> type(lor_codes_nw)
dict
>>> list(lor_codes_nw.keys())
['NW/NZ', 'Notes', 'Last updated date']
>>> nw_codes = lor_codes_nw['NW/NZ']
>>> nw_codes.head()
     Code  ... RA Note
0  NW1001  ...
1  NW1002  ...
2  NW1003  ...
3  NW1004  ...
4  NW1005  ...
[5 rows x 5 columns]
>>> lor_codes_xr = lor.collect_codes(prefix='XR')
>>> type(lor_codes_xr)
dict
>>> list(lor_codes_xr.keys())
['XR', 'Last updated date']
>>> xr_codes = lor_codes_xr['XR']
>>> type(xr_codes)
dict
>>> list(xr_codes.keys())
['Current codes', 'Current codes note', 'Past codes', 'Past codes note']
>>> xr_codes['Past codes'].head()
    Code  ... RA Note
0  XR001  ...
1  XR002  ...
[2 rows x 5 columns]
>>> xr_codes['Current codes'].head()
    Code  ...                       RA Note
0  XR001  ...  Originally reported as RA4
1  XR002  ...  Originally reported as RA4
[2 rows x 5 columns]
```

LOR.collect_elr_lor_converter

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

Collects data of ELR/LOR converter from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the data of ELR/LOR converter, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.line_data import LOR  # from pyrcs import LOR
>>> lor = LOR()
>>> elr_lor_conv = lor.collect_elr_lor_converter()
To collect data of ELR/LOR converter
? [No]|Yes: yes
>>> type(elr_lor_conv)
dict
>>> list(elr_lor_conv.keys())
['ELR/LOR converter', 'Last updated date']
>>> elr_loc_conv_data = elr_lor_conv['ELR/LOR converter']
>>> type(elr_loc_conv_data)
pandas.core.frame.DataFrame
>>> elr_loc_conv_data.head()
    ELR  ...                                            LOR_URL
0   AAV  ...  http://www.railwaycodes.org.uk/pride/pridesw.s...
1   ABD  ...  http://www.railwaycodes.org.uk/pride/pridegw.s...
2   ABE  ...  http://www.railwaycodes.org.uk/pride/prideln.s...
3  ABE1  ...  http://www.railwaycodes.org.uk/pride/prideln.s...
4  ABE2  ...  http://www.railwaycodes.org.uk/pride/prideln.s...
[5 rows x 6 columns]
```

LOR.collect_keys_to_prefixes

LOR.**collect_keys_to_prefixes**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects the keys to PRIDE/LOR code prefixes from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.

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

Returns
A dictionary containing code prefixes and the last updated date, or `None` if no data is available.

Return type
dict | None

Examples:

```
>>> from pyrcs.line_data import LOR  # from pyrcs import LOR
>>> lor = LOR()
>>> lor_page_urls = lor.collect_keys_to_prefixes()
To collect data of URLs to LOR codes web pages
? [No]|Yes: yes
Collecting the data ... Done.
>>> lor_page_urls[0]
'http://www.railwaycodes.org.uk/pride/pridecy.shtm'
```

LOR.collect_page_urls

LOR.**collect_page_urls**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects a list of URLs to PRIDE/LOR codes web pages.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
A list of URLs of web pages the LOR codes.

Return type
list

Examples:

```
>>> from pyrcs.line_data import LOR  # from pyrcs import LOR
>>> lor = LOR()
>>> lor_page_urls = lor.collect_page_urls()
To collect data of URLs to LOR codes web pages
? [No]|Yes: yes
Collecting the data ... Done.
>>> lor_page_urls[0]
'http://www.railwaycodes.org.uk/pride/pridecy.shtm'
```

LOR.fetch_codes

LOR.**fetch_codes**(*prefix=None*, *update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of PRIDE/LOR codes.

Parameters

- **prefix** (*str | None*) – The prefix of LOR codes; defaults to None.
- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the LOR codes.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import LOR  # from pyrcs import LOR
>>> lor = LOR()
>>> lor_codes_dat_cy = lor.fetch_codes(prefix='CY')
>>> type(lor_codes_dat_cy)
dict
>>> list(lor_codes_dat_cy)
['CY', 'Notes', 'Last updated date']
>>> lor_codes_dat_cy['CY']
     Code  ...                          RA Note
0   CY240  ...           Caerwent branch RA4
1  CY1540  ...  Pembroke - Pembroke Dock RA6
[2 rows x 5 columns]
>>> lor_codes_dat = lor.fetch_codes()
>>> type(lor_codes_dat)
dict
>>> list(lor_codes_dat.keys())
['LOR', 'Last updated date']
>>> l_codes = lor_codes_dat['LOR']
>>> type(l_codes)
dict
>>> list(l_codes.keys())[:5]
['CY', 'CY Notes', 'EA', 'EA Notes', 'GW']
>>> cy_codes = l_codes['CY']
>>> type(cy_codes)
pandas.core.frame.DataFrame
>>> cy_codes
     Code  ...                          RA Note
0   CY240  ...           Caerwent branch RA4
1  CY1540  ...  Pembroke - Pembroke Dock RA6
[2 rows x 5 columns]
>>> xr_codes = l_codes['XR']
>>> type(xr_codes)
```

(continues on next page)

(continued from previous page)

```
dict
>>> list(xr_codes.keys())
['Current codes', 'Current codes note', 'Past codes', 'Past codes note']
>>> xr_codes['Past codes']
    Code  ... RA Note
0  XR001  ...
1  XR002  ...
[2 rows x 5 columns]
>>> xr_codes['Current codes']
    Code  ...                      RA Note
0  XR001  ...  Originally reported as RA4
1  XR002  ...  Originally reported as RA4
[2 rows x 5 columns]
```

LOR.fetch_elr_lor_converter

LOR.**fetch_elr_lor_converter**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of ELR/LOR converter.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Returns

A dictionary containing the data of ELR/LOR converter.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import LOR  # from pyrcs import LOR
>>> lor = LOR()
>>> elr_lor_conv = lor.fetch_elr_lor_converter()
>>> type(elr_lor_conv)
dict
>>> list(elr_lor_conv.keys())
['ELR/LOR converter', 'Last updated date']
>>> elr_loc_conv_data = elr_lor_conv['ELR/LOR converter']
>>> type(elr_loc_conv_data)
pandas.core.frame.DataFrame
>>> elr_loc_conv_data.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.get_keys_to_prefixes

LOR.**get_keys_to_prefixes**(*prefixes_only=True*, *update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Gets the keys to PRIDE/LOR code prefixes.

Parameters

- **prefixes_only** (*bool*) – If `True` (default), returns only the prefixes; otherwise, additional information, including the last updated date.
- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Returns

A list of the keys to LOR code prefixes if `prefixes_only=True`, otherwise a dictionary containing code prefixes and the last updated date, or `None` if no data is available.

Return type

list | dict | None

Examples:

```
>>> from pyrcs.line_data import LOR  # from pyrcs import LOR
>>> lor = LOR()
>>> keys_to_pfx = lor.get_keys_to_prefixes()
>>> keys_to_pfx
['CY', 'EA', 'GW', 'LN', 'MD', 'NW', 'NZ', 'SC', 'SO', 'SW', 'XR']
>>> keys_to_pfx = lor.get_keys_to_prefixes(prefixes_only=False)
>>> type(keys_to_pfx)
dict
>>> list(keys_to_pfx.keys())
['Key to prefixes', 'Last updated date']
>>> keys_to_pfx_codes = keys_to_pfx['Key to prefixes']
>>> type(keys_to_pfx_codes)
pandas.core.frame.DataFrame
>>> keys_to_pfx_codes.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.get_page_urls

LOR.**get_page_urls**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Gets a list of URLs to PRIDE/LOR codes web pages.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A list of URLs of the web pages hosting LOR codes for each prefix.

Return type

list | None

Examples:

```
>>> from pyrcs.line_data import LOR  # from pyrcs import LOR
>>> lor = LOR()
>>> lor_urls = lor.get_page_urls()
>>> type(lor_urls)
list
>>> lor_urls[0]
'http://www.railwaycodes.org.uk/pride/pridecy.shtm'
```

LOR.get_url

LOR.**get_url**(*prefix*)

Generates the URL for the given PRIDE/LOR code prefix.

This method constructs the appropriate webpage URL based on the provided *prefix*, ensuring that it is valid before appending the correct suffix.

Parameters

prefix (*str*) – The PRIDE/LOR code prefix.

Returns

A fully constructed URL corresponding to the given prefix.

Return type

str

Examples:

```
>>> from pyrcs.line_data import LOR  # from pyrcs import LOR
>>> lor = LOR()
>>> lor.get_url(prefix='CY')
'http://www.railwaycodes.org.uk/pride/pridecy.shtm'
>>> lor.get_url(prefix='CA')
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
    ...
AssertionError: `prefix` must be one of ['CY', 'EA', 'GW', 'LN', 'MD', 'NW', 'NZ', ...
```

LOR.validate_prefix

LOR.**validate_prefix**(*prefix*)

Validates and standardises a PRIDE/LOR code prefix.

If the provided *prefix* is not found in the list of valid prefixes, an attempt is made to find the closest matching valid prefix. If no match is found, an error is raised.

Parameters
prefix (*str*) – The PRIDE/LOR code prefix to validate.

Raises
AssertionError – If the *prefix* is not a valid PRIDE/LOR prefix.

Returns
A validated and standardised uppercase prefix.

Return type
str

Examples:

```
>>> from pyrcs.line_data import LOR  # from pyrcs import LOR
>>> lor = LOR()
>>> lor.validate_prefix(prefix='cy')
'CY'
>>> lor.validate_prefix(prefix='ca')
Traceback (most recent call last):
    ...
AssertionError: `prefix` must be one of ['CY', 'EA', 'GW', 'LN', 'MD', 'NW', 'NZ', ...
```

4.1.5 LineNames

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

A class for collecting data of railway line names.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the data catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.

- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.line_data import LineNames  # from pyrcs import LineNames
>>> ln = LineNames()
>>> ln.NAME
'Railway line names'
>>> ln.URL
'http://www.railwaycodes.org.uk/misc/line_names.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
NAME	The name of the data.
URL	The URL of the main web page for the data.

LineNames.KEY

LineNames.**KEY: str = 'Line names'**

The key for accessing the data.

LineNames.KEY_TO_LAST_UPDATED_DATE

LineNames.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

LineNames.NAME

LineNames.**NAME: str = 'Railway line names'**

The name of the data.

LineNames.URL

LineNames.**URL: str = 'http://www.railwaycodes.org.uk/line/line_names.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_codes([confirmation_required, ...])	Collects data of railway line names and associated route data from the source web page.
fetch_codes([update, dump_dir, verbose])	Fetches data of railway line names and associated route data.

LineNames.collect_codes

LineNames.**collect_codes**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects data of railway line names and associated route data from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing railway line names, route data and the last update date, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.line_data import LineNames  # from pyrcs import LineNames
>>> ln = LineNames()
>>> line_names_codes = ln.collect_codes()
To collect British railway line names
? [No]|Yes: yes
>>> type(line_names_codes)
dict
>>> list(line_names_codes.keys())
['Line names', 'Last updated date']
>>> ln.KEY
'Line names'
>>> line_names_codes_dat = line_names_codes[ln.KEY]
>>> type(line_names_codes_dat)
pandas.core.frame.DataFrame
>>> line_names_codes_dat.head()
          Line name  ... Route_note
0       Abbey Line  ...       None
1    Aberdare Line  ...       None
2    Airedale Line  ...       None
3      Argyle Line  ...       None
4  Arun Valley Line  ...       None
[5 rows x 3 columns]
```

LineNames.fetch_codes

LineNames.**fetch_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of railway line names and associated route data.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing railway line names, route data and the last update date.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import LineNames  # from pyrcs import LineNames
>>> ln = LineNames()
>>> line_names_codes = ln.fetch_codes()
>>> type(line_names_codes)
dict
>>> list(line_names_codes.keys())
['Line names', 'Last updated date']
>>> ln.KEY
'Line names'
>>> line_names_codes_dat = line_names_codes[ln.KEY]
>>> type(line_names_codes_dat)
pandas.core.frame.DataFrame
>>> line_names_codes_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]
```

4.1.6 TrackDiagrams

class pyrcs.line_data.**TrackDiagrams**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting data of British railway track diagrams.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.

- **update** (*bool*) – Whether to check for updates to the data catalogue; defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `True`.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.line_data import TrackDiagrams  # from pyrcs import TrackDiagrams
>>> td = TrackDiagrams()
>>> td.NAME
'Railway track diagrams'
>>> td.URL
'http://www.railwaycodes.org.uk/line/diagrams0.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
NAME	The name of the data.
URL	The URL of the main web page for the data.

TrackDiagrams.KEY

TrackDiagrams.**KEY: str = 'Track diagrams'**

The key for accessing the data.

TrackDiagrams.KEY_TO_LAST_UPDATED_DATE

TrackDiagrams.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

TrackDiagrams.NAME

TrackDiagrams.**NAME: str = 'Railway track diagrams'**

The name of the data.

TrackDiagrams.URL

TrackDiagrams.**URL: str = 'http://www.railwaycodes.org.uk/line/diagrams0.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_catalogue([confirmation_required, ...])	Collects the catalogue of sample railway track diagrams from the source web page.
fetch_catalogue([update, dump_dir, verbose])	Fetches the catalogue of railway track diagrams.

TrackDiagrams.collect_catalogue

TrackDiagrams.**collect_catalogue**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects the catalogue of sample railway track diagrams from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the railway track diagram catalogue and the date it was last updated, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.line_data import TrackDiagrams  # from pyrcs import TrackDiagrams
>>> td = TrackDiagrams()
>>> track_diagrams_catalog = td.collect_catalogue()
To collect the catalogue of track diagrams
? [No]|Yes: yes
>>> type(track_diagrams_catalog)
dict
>>> list(track_diagrams_catalog.keys())
['Track diagrams', 'Last updated date']
>>> td_dat = track_diagrams_catalog['Track diagrams']
>>> type(td_dat)
dict
```

(continues on next page)

(continued from previous page)

```
>>> list(td_dat.keys())
['Main line diagrams', 'Tram systems', 'London Underground', 'Miscellaneous']
>>> main_line_diagrams = td_dat['Main line diagrams']
>>> type(main_line_diagrams)
tuple
>>> type(main_line_diagrams[1])
pandas.core.frame.DataFrame
>>> main_line_diagrams[1].head()
                            Description                                        FileURL
0  South Central area (1985) 10.4Mb file  http://www.railwaycodes.org.uk/line/track/d...
1   South Eastern area (1976) 5.4Mb file  http://www.railwaycodes.org.uk/line/track/d...
```

TrackDiagrams.fetch_catalogue

TrackDiagrams.**fetch_catalogue**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches the catalogue of railway track diagrams.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the catalogue of railway track diagrams and the date it was last updated.

Return type

dict

Examples:

```
>>> from pyrcs.line_data import TrackDiagrams  # from pyrcs import TrackDiagrams
>>> td = TrackDiagrams()
>>> trk_diagr_cat = td.fetch_catalogue()
>>> type(trk_diagr_cat)
dict
>>> list(trk_diagr_cat.keys())
['Track diagrams', 'Last updated date']
>>> td_dat = trk_diagr_cat['Track diagrams']
>>> type(td_dat)
dict
>>> list(td_dat.keys())
['Main line diagrams', 'Tram systems', 'London Underground', 'Miscellaneous']
>>> main_line_diagrams = td_dat['Main line diagrams']
>>> type(main_line_diagrams)
tuple
>>> type(main_line_diagrams[1])
pandas.core.frame.DataFrame
>>> main_line_diagrams[1].head()
```

(continues on next page)

(continued from previous page)

```
                              Description                                        FileURL
0  South Central area (1985) 10.4Mb file  http://www.railwaycodes.org.uk/line/track/d...
1   South Eastern area (1976) 5.4Mb file  http://www.railwaycodes.org.uk/line/track/d...
```

4.1.7 Bridges

class pyrcs.line_data.**Bridges**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting data of railway bridges.

Parameters

- **data_dir** (*str | None*) – Directory where the data is stored; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.line_data import Bridges  # from pyrcs import Bridges
>>> bdg = Bridges()
>>> bdg.NAME
'Railway bridges'
>>> bdg.URL
'http://www.railwaycodes.org.uk/bridges/bridges0.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
NAME	The name of the data.
URL	The URL of the main webpage for the data.

Bridges.KEY

Bridges.**KEY: str = 'Bridges'**

The key for accessing the data.

Bridges.KEY_TO_LAST_UPDATED_DATE

Bridges.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

Bridges.NAME

Bridges.**NAME: str = 'Railway bridges'**

The name of the data.

Bridges.URL

Bridges.**URL: str = 'http://www.railwaycodes.org.uk/bridges/bridges0.shtm'**

The URL of the main webpage for the data.

Methods

1anti-flashwhitewhite

collect_codes([confirmation_required, ...])	Collects codes of railway bridges from its source webpage.
fetch_codes([update, dump_dir, verbose])	Fetches codes of railway bridges.

Bridges.collect_codes

Bridges.**collect_codes**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects codes of railway bridges from its source webpage.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information in the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing railway bridge data and the date of the last update, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.line_data import Bridges  # from pyrcs import Bridges
>>> bdg = Bridges()
>>> bdg_codes = bdg.collect_codes(verbose=True)
```

(continues on next page)

(continued from previous page)

```
To collect data of railway bridges
? [No]|Yes: yes
>>> type(bdg_codes)
dict
>>> list(bdg_codes.keys())
['East Coast Main Line',
 'Midland Main Line',
 'West Coast Main Line',
 'Scotland',
 'Elizabeth Line',
 'London Overground',
 'Anglia',
 'London Underground',
 'Key to text presentation conventions']
>>> bdg_codes['Key to text presentation conventions']
{'Bold': 'Existing bridges',
 'Bold italic': 'Existing locations',
 'Light italic': 'Former/historical locations',
 'Red': 'Stations',
 'Deep red': 'Level crossings',
 'Brown': 'Ventilation shafts',
 'Purple': 'Junctions',
 'Black,grey': 'Bridges and culverts',
 'Green': 'Tunnel portals',
 'Bright blue': 'Viaducts',
 'Deep blue': 'Boundaries'}
```

Bridges.fetch_codes

Bridges.**fetch_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches codes of railway bridges.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing railway bridge data and the date of the last update, or None if no data is retrieved.

Return type

dict | None

Examples:

```
>>> from pyrcs.line_data import Bridges  # from pyrcs import Bridges
>>> bdg = Bridges()
>>> bdg_codes = bdg.fetch_codes()
```

(continues on next page)

(continued from previous page)

```
>>> type(bdg_codes)
dict
>>> list(bdg_codes.keys())
['East Coast Main Line',
 'West Coast Main Line',
 'Scotland',
 'Elizabeth Line',
 'London Overground',
 'Anglia',
 'London Underground',
 'Addendum',
 'Key to text presentation conventions']
>>> bdg_codes['Key to text presentation conventions']
{'Bold': 'Existing bridges',
 'Bold italic': 'Existing locations',
 'Light italic': 'Former/historical locations',
 'Red': 'Stations',
 'Deep red': 'Level crossings',
 'Brown': 'Ventilation shafts',
 'Purple': 'Junctions',
 'Black,grey': 'Bridges and culverts',
 'Green': 'Tunnel portals',
 'Bright blue': 'Viaducts',
 'Deep blue': 'Boundaries'}
```

4.2 other_assets

A subpackage for collecting codes of other assets.

(See also the *OtherAssets* class.)

1anti-flashwhitewhite

SignalBoxes([data_dir, update, verbose])	A class for collecting data of signal box prefix codes.
Tunnels([data_dir, update, verbose])	A class for collecting data of railway tunnel lengths.
Viaducts([data_dir, update, verbose])	A class for collecting codes of railway viaducts.
Stations([data_dir, update, verbose])	A class for collecting railway station data.
Depots([data_dir, update, verbose])	A class for collecting data of depot codes.
Features([data_dir, update, verbose])	A class for collecting data of several infrastructure features, including *HABDs and WILDs*, *water troughs locations*, *telegraph code words* and *buzzer codes*.
HabdWild([data_dir, update, verbose])	A class for collecting data of HABDs and WILDs.
WaterTroughs([data_dir, update, verbose])	A class for collecting data of water troughs locations.
Telegraph([data_dir, update, verbose])	A class for collecting data of telegraph code words.

continues on next page

Table 17 – continued from previous page

Buzzer([data_dir, update, verbose])	A class for collecting data of buzzer codes.

4.2.1 SignalBoxes

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

A class for collecting data of signal box prefix codes.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.other_assets import SignalBoxes  # from pyrcs import SignalBoxes
>>> sb = SignalBoxes()
>>> sb.NAME
'Signal box prefix codes'
>>> sb.URL
'http://www.railwaycodes.org.uk/signal/signal_boxes0.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_BELL_CODES	The key for accessing the data of *bell codes*.
KEY_TO_IRELAND	The key for accessing the data of *Ireland*.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
KEY_TO_NON_NATIONAL_RAIL	The key for accessing the data of *non-national rail*.
KEY_TO_WRMASD	The key for accessing the data of *WR (Western region) MAS (multiple aspect signalling) dates*.

continues on next page

Table 18 – continued from previous page

NAME	The name of the data.
URL	The URL of the main web page for the data.

SignalBoxes.KEY

SignalBoxes.**KEY: str = 'Signal boxes'**

The key for accessing the data.

SignalBoxes.KEY_TO_BELL_CODES

SignalBoxes.**KEY_TO_BELL_CODES: str = 'Bell codes'**

The key for accessing the data of *bell codes*.

SignalBoxes.KEY_TO_IRELAND

SignalBoxes.**KEY_TO_IRELAND: str = 'Ireland'**

The key for accessing the data of *Ireland*.

SignalBoxes.KEY_TO_LAST_UPDATED_DATE

SignalBoxes.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

SignalBoxes.KEY_TO_NON_NATIONAL_RAIL

SignalBoxes.**KEY_TO_NON_NATIONAL_RAIL: str = 'Non-National Rail'**

The key for accessing the data of *non-national rail*.

SignalBoxes.KEY_TO_WRMASD

SignalBoxes.**KEY_TO_WRMASD: str = 'WR MAS dates'**

The key for accessing the data of *WR (Western region) MAS (multiple aspect signalling) dates*.

SignalBoxes.NAME

SignalBoxes.**NAME: str = 'Signal box prefix codes'**

The name of the data.

SignalBoxes.URL

SignalBoxes.**URL: str =**
'http://www.railwaycodes.org.uk/signal/signal_boxes0.shtm'

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_bell_codes([confirmation_required, ...])	Collects data of bell codes from the source web page.
collect_ireland_codes([...])	Collects data of Irish signal cabin prefix codes from the source web page.
collect_non_national_rail_codes([...])	Collects signal box prefix codes for non-national rail from the source web page.
collect_prefix_codes(initial[, ...])	Collects signal box prefix codes for a given initial letter from the source web page.
collect_wr_mas_dates([...])	Collects data of WR (western region) MAS (multiple aspect signalling) dates from the source web page.
fetch_bell_codes([update, dump_dir, verbose])	Fetches data of bell codes.
fetch_ireland_codes([update, dump_dir, verbose])	Fetches data of Irish signal cabin prefix codes.
fetch_non_national_rail_codes([update, ...])	Fetches signal box prefix codes for non-national rail.
fetch_prefix_codes([initial, update, ...])	Fetches data of signal box prefix codes.
fetch_wr_mas_dates([update, dump_dir, verbose])	Fetches data of WR (western region) MAS (multiple aspect signalling) dates.

SignalBoxes.collect_bell_codes

SignalBoxes.**collect_bell_codes**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects data of bell codes from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the data of bell codes and the date when they were last updated, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.other_assets import SignalBoxes  # from pyrcs import SignalBoxes
>>> sb = SignalBoxes()
>>> sb_bell_codes = sb.collect_bell_codes()
To collect data of Bell codes
? [No]|Yes: yes
>>> type(sb_bell_codes)
dict
>>> list(sb_bell_codes.keys())
['Bell codes', 'Last updated date']
>>> sb.KEY_TO_BELL_CODES
'Bell codes'
>>> sb_bell_codes_dat = sb_bell_codes[sb.KEY_TO_BELL_CODES]
>>> type(sb_bell_codes_dat)
dict
>>> list(sb_bell_codes_dat.keys())
['Network Rail codes',
 'Southern Railway codes',
 'Lancashire & Yorkshire Railway codes']
>>> sb_nr_bell_codes = sb_bell_codes_dat['Network Rail codes']
>>> type(sb_nr_bell_codes)
dict
>>> list(sb_nr_bell_codes.keys())
['Codes', 'Notes']
>>> sb_nr_bell_codes_dat = sb_nr_bell_codes['Codes']
>>> type(sb_nr_bell_codes_dat)
pandas.core.frame.DataFrame
>>> sb_nr_bell_codes_dat.head()
    Code                                       Meaning
0      1                                Call attention
1    1-1            Answer telephone [withdrawn 2007]
2  1-1-6          Police assistance urgently required
3    1-2  Signaller required on telephone [added 2007]
4  1-2-1                            Train approaching
```

SignalBoxes.collect_ireland_codes

SignalBoxes.**collect_ireland_codes**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects data of Irish signal cabin prefix codes from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the data of Irish signal cabin prefix codes and the date when they were last updated, or `None` if no data is collectd.

Return type
dict | None

Examples:

```
>>> from pyrcs.other_assets import SignalBoxes  # from pyrcs import SignalBoxes
>>> sb = SignalBoxes()
>>> ireland_sb_codes = sb.collect_ireland_codes()
To collect data of signal box prefix codes of Ireland
? [No]|Yes: yes
>>> type(ireland_sb_codes)
dict
>>> list(ireland_sb_codes.keys())
['Ireland', 'Notes', 'Last updated date']
>>> sb.KEY_TO_IRELAND
'Ireland'
>>> ireland_sb_codes_dat = ireland_sb_codes[sb.KEY_TO_IRELAND]
>>> type(ireland_sb_codes_dat)
pandas.core.frame.DataFrame
>>> ireland_sb_codes_dat.head()
   Code Signal Cabin                    Note
0    AD     Adelaide
1    AN       Antrim
2    AE      Athlone
3  AE R                      Distant signals
4    XG               Level crossing signals
```

SignalBoxes.collect_non_national_rail_codes

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

Collects signal box prefix codes for non-national rail from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
A dictionary containing the signal box prefix codes for non-national rail and the date when they were last updated, or `None` if no data is collected.

Return type
dict | None

Examples:

```
>>> from pyrcs.other_assets import SignalBoxes  # from pyrcs import SignalBoxes
>>> sb = SignalBoxes()
```

(continues on next page)

(continued from previous page)

```
>>> nnr_codes = sb.collect_non_national_rail_codes()
To collect data of non-national rail signal box prefix codes
? [No]|Yes: yes
>>> type(nnr_codes)
dict
>>> list(nnr_codes.keys())
['Non-National Rail', 'Last updated date']
>>> sb.KEY_TO_NON_NATIONAL_RAIL
'Non-National Rail'
>>> nnr_codes_dat = nnr_codes[sb.KEY_TO_NON_NATIONAL_RAIL]
>>> type(nnr_codes_dat)
dict
>>> list(nnr_codes_dat.keys())
['Croydon Tramlink signals',
 'Docklands Light Railway signals',
 'Edinburgh Tramway signals',
 'Glasgow Subway signals',
 'London Underground signals',
 'Luas signals',
 'Manchester Metrolink signals',
 'Midland Metro signals',
 'Nottingham Tram signals',
 'Sheffield Supertram signals',
 'Tyne & Wear Metro signals',
 "Heritage, minor and miniature railways and other 'special' signals"]
>>> lu_signals_codes = nnr_codes_dat['London Underground signals']
>>> type(lu_signals_codes)
dict
>>> list(lu_signals_codes.keys())
['Codes', 'Notes']
>>> type(lu_signals_codes['Codes'])
pandas.core.frame.DataFrame
>>> lu_signals_codes['Codes'].head()
  Code  ... Became or taken over by (where known)
0  BMX  ...                                      -
1    A  ...                                      -
2    S  ...                                      -
3    X  ...                                      -
4    R  ...                                      -
[5 rows x 5 columns]
```

SignalBoxes.collect_prefix_codes

SignalBoxes.**collect_prefix_codes**(*initial*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects signal box prefix codes for a given initial letter from the source web page.

Parameters

- **initial** (*str*) – The initial letter (e.g. 'a', 'z') of signal box prefix code.
- **confirmation_required** (*bool*) – Whether user confirmation is required; if confirmation_required=True (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the

console; defaults to `False`.

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

Returns

A dictionary containing data of signal box prefix codes whose initial letters are the specified `initial` and the date of when the data was last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import SignalBoxes  # from pyrcs import SignalBoxes
>>> sb = SignalBoxes()
>>> sb_a_codes = sb.collect_prefix_codes(initial='a')
>>> type(sb_a_codes)
dict
>>> list(sb_a_codes.keys())
['A', 'Last updated date']
>>> sb_a_codes_dat = sb_a_codes['A']
>>> type(sb_a_codes_dat)
pandas.core.frame.DataFrame
>>> sb_a_codes_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]
```

SignalBoxes.collect_wr_mas_dates

`SignalBoxes.`**collect_wr_mas_dates**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects data of WR (western region) MAS (multiple aspect signalling) dates from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the data of WR MAS dates and the date when they were last updated, or `None` if no data is collected.

Return type
dict | None

Examples:

```
>>> from pyrcs.other_assets import SignalBoxes  # from pyrcs import SignalBoxes
>>> sb = SignalBoxes()
>>> sb_wr_mas_dates = sb.collect_wr_mas_dates()
To collect data of WR MAS dates
? [No]|Yes: yes
>>> type(sb_wr_mas_dates)
dict
>>> list(sb_wr_mas_dates.keys())
['WR MAS dates', 'Last updated date']
>>> sb.KEY_TO_WRMASD
'WR MAS dates'
>>> sb_wr_mas_dates_dat = sb_wr_mas_dates[sb.KEY_TO_WRMASD]
>>> type(sb_wr_mas_dates_dat)
dict
>>> list(sb_wr_mas_dates_dat.keys())[:5]
['Paddington-Hayes',
 'Birmingham',
 'Plymouth',
 'Reading-Hayes',
 'Newport Multiple Aspect Signalling']
>>> sb_wr_mas_dates_dat['Paddington-Hayes']
  Stage             Date                          Area
0    1A    12 April 1953                Hayes-Hanwell
1    1B    20 March 1955         Hanwell-Acton Middle
2    1C  1 February 1959  Acton West-Friars Junction
```

SignalBoxes.fetch_bell_codes

SignalBoxes.**fetch_bell_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of bell codes.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns
A dictionary containing the data of bell codes and the date when they were last updated.

Return type
dict

Examples:

```
>>> from pyrcs.other_assets import SignalBoxes  # from pyrcs import SignalBoxes
>>> sb = SignalBoxes()
>>> sb_bell_codes = sb.fetch_bell_codes()
>>> type(sb_bell_codes)
dict
>>> list(sb_bell_codes.keys())
['Bell codes', 'Last updated date']
>>> sb.KEY_TO_BELL_CODES
'Bell codes'
>>> sb_bell_codes_dat = sb_bell_codes[sb.KEY_TO_BELL_CODES]
>>> type(sb_bell_codes_dat)
dict
>>> list(sb_bell_codes_dat.keys())
['Network Rail codes',
 'Southern Railway codes',
 'Lancashire & Yorkshire Railway codes']
>>> sb_nr_bell_codes = sb_bell_codes_dat['Network Rail codes']
>>> type(sb_nr_bell_codes)
dict
>>> list(sb_nr_bell_codes.keys())
['Codes', 'Notes']
>>> sb_nr_bell_codes_dat = sb_nr_bell_codes['Codes']
>>> type(sb_nr_bell_codes_dat)
pandas.core.frame.DataFrame
>>> sb_nr_bell_codes_dat.head()
    Code                                    Meaning
0      1                             Call attention
1    1-1             Answer telephone [withdrawn 2007]
2  1-1-6          Police assistance urgently required
3    1-2  Signaller required on telephone [added 2007]
4  1-2-1                           Train approaching
```

SignalBoxes.fetch_ireland_codes

SignalBoxes.**fetch_ireland_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of Irish signal cabin prefix codes.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the data of Irish signal cabin prefix codes and the date when they were last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import SignalBoxes  # from pyrcs import SignalBoxes
>>> sb = SignalBoxes()
>>> ireland_sb_codes = sb.fetch_ireland_codes()
>>> type(ireland_sb_codes)
dict
>>> list(ireland_sb_codes.keys())
['Ireland', 'Notes', 'Last updated date']
>>> sb.KEY_TO_IRELAND
'Ireland'
>>> ireland_sb_codes_dat = ireland_sb_codes[sb.KEY_TO_IRELAND]
>>> type(ireland_sb_codes_dat)
pandas.core.frame.DataFrame
>>> ireland_sb_codes_dat.head()
   Code Signal Cabin                    Note
0    AD     Adelaide
1    AN       Antrim
2    AE      Athlone
3  AE R                      Distant signals
4    XG               Level crossing signals
```

SignalBoxes.fetch_non_national_rail_codes

SignalBoxes.**fetch_non_national_rail_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches signal box prefix codes for non-national rail.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the signal box prefix codes for non-national rail and the date when they were last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import SignalBoxes  # from pyrcs import SignalBoxes
>>> sb = SignalBoxes()
>>> nnr_codes = sb.fetch_non_national_rail_codes()
>>> type(nnr_codes)
dict
>>> list(nnr_codes.keys())
['Non-National Rail', 'Last updated date']
>>> sb.KEY_TO_NON_NATIONAL_RAIL
'Non-National Rail'
>>> nnr_codes_dat = nnr_codes[sb.KEY_TO_NON_NATIONAL_RAIL]
```

(continues on next page)

(continued from previous page)

```
>>> type(nnr_codes_dat)
dict
>>> list(nnr_codes_dat.keys())
['Croydon Tramlink signals',
 'Docklands Light Railway signals',
 'Edinburgh Tramway signals',
 'Glasgow Subway signals',
 'London Underground signals',
 'Luas signals',
 'Manchester Metrolink signals',
 'Midland Metro signals',
 'Nottingham Tram signals',
 'Sheffield Supertram signals',
 'Tyne & Wear Metro signals',
 "Heritage, minor and miniature railways and other 'special' signals"]
>>> lu_signals_codes = nnr_codes_dat['London Underground signals']
>>> type(lu_signals_codes)
dict
>>> list(lu_signals_codes.keys())
['Codes', 'Notes']
>>> type(lu_signals_codes['Codes'])
pandas.core.frame.DataFrame
>>> lu_signals_codes['Codes'].head()
  Code  ... Became or taken over by (where known)
0  BMX  ...                                      -
1    A  ...                                      -
2    S  ...                                      -
3    X  ...                                      -
4    R  ...                                      -
[5 rows x 5 columns]
```

SignalBoxes.fetch_prefix_codes

SignalBoxes.**fetch_prefix_codes**(*initial=None*, *update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of signal box prefix codes.

Parameters

- **initial** (*str*) – The initial letter (e.g. `'a'`, `'z'`) of signal box prefix code; defaults to `None`.
- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Returns

A dictionary containing the data of signal box prefix codes and the date whey they were last updated.

Return type
dict

Examples:

```
>>> from pyrcs.other_assets import SignalBoxes  # from pyrcs import SignalBoxes
>>> sb = SignalBoxes()
>>> sb_prefix_codes = sb.fetch_prefix_codes()
>>> type(sb_prefix_codes)
dict
>>> list(sb_prefix_codes.keys())
['Signal boxes', 'Last updated date']
>>> sb.KEY
'Signal boxes'
>>> sb_prefix_codes_dat = sb_prefix_codes[sb.KEY]
>>> type(sb_prefix_codes_dat)
pandas.core.frame.DataFrame
>>> sb_prefix_codes_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]
```

SignalBoxes.fetch_wr_mas_dates

SignalBoxes.**fetch_wr_mas_dates**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of WR (western region) MAS (multiple aspect signalling) dates.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns
A dictionary containing the data of WR MAS dates and the date when they were last updated.

Return type
dict

Examples:

```
>>> from pyrcs.other_assets import SignalBoxes  # from pyrcs import SignalBoxes
>>> sb = SignalBoxes()
>>> sb_wr_mas_dates = sb.fetch_wr_mas_dates()
>>> type(sb_wr_mas_dates)
dict
```

(continues on next page)

(continued from previous page)

```
>>> list(sb_wr_mas_dates.keys())
['WR MAS dates', 'Last updated date']
>>> sb.KEY_TO_WRMASD
'WR MAS dates'
>>> sb_wr_mas_dates_dat = sb_wr_mas_dates[sb.KEY_TO_WRMASD]
>>> type(sb_wr_mas_dates_dat)
dict
>>> list(sb_wr_mas_dates_dat.keys())[:5]
['Paddington-Hayes',
 'Birmingham',
 'Plymouth',
 'Reading-Hayes',
 'Newport Multiple Aspect Signalling']
>>> sb_wr_mas_dates_dat['Paddington-Hayes']
  Stage             Date                        Area
0    1A    12 April 1953                Hayes-Hanwell
1    1B    20 March 1955         Hanwell-Acton Middle
2    1C  1 February 1959  Acton West-Friars Junction
```

4.2.2 Tunnels

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

A class for collecting data of railway tunnel lengths.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.other_assets import Tunnels  # from pyrcs import Tunnels
>>> tunl = Tunnels()
>>> tunl.NAME
'Railway tunnel lengths'
>>> tunl.URL
'http://www.railwaycodes.org.uk/tunnels/tunnels0.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
NAME	The name of the data.
URL	The URL of the main web page for the data.

Tunnels.KEY

Tunnels.**KEY: str = 'Tunnels'**

The key for accessing the data.

Tunnels.KEY_TO_LAST_UPDATED_DATE

Tunnels.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

Tunnels.NAME

Tunnels.**NAME: str = 'Railway tunnel lengths'**

The name of the data.

Tunnels.URL

Tunnels.**URL: str = 'http://www.railwaycodes.org.uk/tunnels/tunnels0.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_codes(page_no[, ...])	Collects data of railway tunnel lengths for a specific page number from the source web page.
fetch_codes([page_no, update, dump_dir, verbose])	Fetches data of railway tunnel lengths.

Tunnels.collect_codes

Tunnels.**collect_codes**(*page_no*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects data of railway tunnel lengths for a specific page number from the source web page.

Parameters

- **page_no** (*int | str*) – The page number to collect data from; valid values are 1, 2, 3 and 4.
- **confirmation_required** (*bool*) – Whether user confirmation is required; if confirmation_required=True (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Returns

A dictionary containing the data of tunnel lengths for the specified page_no and the date of when the data was last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Tunnels  # from pyrcs import Tunnels
>>> tunl = Tunnels()
>>> page_1 = tunl.collect_codes(page_no=1)
>>> type(page_1)
dict
>>> list(page_1.keys())
['Page 1 (A-F)', 'Last updated date']
>>> page_1_codes = page_1['Page 1 (A-F)']
>>> type(page_1_codes)
pandas.core.frame.DataFrame
>>> page_1_codes.head()
            Name  Other names, remarks  ... Length (metres) Length (note)
0    Abbotscliffe                       ...       1775.7648
1      Abercanaid          see Merthyr  ...             NaN   Unavailable
2     Aberchalder         see Loch Oich  ...             NaN   Unavailable
3  Aberdovey No 1  also called Frongoch  ...        182.8800
4  Aberdovey No 2    also called Morfor  ...        200.2536
[5 rows x 10 columns]
>>> page_4 = tunl.collect_codes(page_no=4)
>>> type(page_4)
dict
>>> list(page_4.keys())
['Page 4 (others)', 'Last updated date']
>>> page_4_codes = page_4['Page 4 (others)']
>>> type(page_4_codes)
dict
>>> list(page_4_codes.keys())
['Tunnels on industrial and other minor lines',
 'Large bridges that are not officially tunnels but could appear to be so']
>>> key1 = 'Tunnels on industrial and other minor lines'
>>> page_4_dat = page_4_codes[key1]
>>> type(page_4_dat)
pandas.core.frame.DataFrame
>>> page_4_dat.head()
                    Name Other names, remarks  ... Length (metres) Length (note)
```

(continues on next page)

(continued from previous page)

```
0             Ashes Quarry                          ...         56.6928
1        Ashey Down Quarry                          ...         33.8328
2  Baileycroft Quarry No 1                          ...         28.3464
3  Baileycroft Quarry No 2                          ...         21.0312
4           Basfords Hill                           ...         46.6344
[5 rows x 6 columns]
>>> key2 = 'Large bridges that are not officially tunnels but could appear to be so'
>>> page_4_dat_ = page_4_codes[key2]
>>> type(page_4_dat_)
pandas.core.frame.DataFrame
>>> page_4_dat_.head()
                Name Other names, remarks  ... Length (metres) Length (note)
0  A470/A472 (north)                       ...          35.6616
1  A470/A472 (south)                       ...          28.3464
2               A720                       ...         145.3896
3                 A9       Aberdeen line  ...         141.7320
4                 A9          Perth line  ...         146.3040
[5 rows x 8 columns]
```

Tunnels.fetch_codes

Tunnels.**fetch_codes**(*page_no=None*, *update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of railway tunnel lengths.

Parameters

- **page_no** (*int | str*) – The page number to collect data from; valid values are 1, 2, 3 and 4; defaults to None.
- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the data of tunnel lengths (including the name, length, owner and relative location) and the date of when the data was last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Tunnels  # from pyrcs import Tunnels
>>> tunl = Tunnels()
>>> tunl_len_codes = tunl.fetch_codes()
>>> type(tunl_len_codes)
dict
>>> list(tunl_len_codes.keys())
['Tunnels', 'Last updated date']
```

(continues on next page)

(continued from previous page)

```
>>> tunl.KEY
'Tunnels'
>>> tunl_len_codes_dat = tunl_len_codes[tunl.KEY]
>>> type(tunl_len_codes_dat)
dict
>>> list(tunl_len_codes_dat.keys())
['Page 1 (A-F)', 'Page 2 (G-P)', 'Page 3 (Q-Z)', 'Page 4 (others)']
>>> page_1_codes = tunl_len_codes_dat['Page 1 (A-F)']
>>> type(page_1_codes)
pandas.core.frame.DataFrame
>>> page_1_codes.head()
             Name  Other names, remarks  ... Length (metres) Length (note)
0    Abbotscliffe                        ...       1775.7648
1      Abercanaid           see Merthyr  ...             NaN   Unavailable
2     Aberchalder         see Loch Oich  ...             NaN   Unavailable
3  Aberdovey No 1  also called Frongoch  ...        182.8800
4  Aberdovey No 2    also called Morfor  ...        200.2536
[5 rows x 10 columns]
>>> page_4_codes = tunl_len_codes_dat['Page 4 (others)']
>>> type(page_4_codes)
dict
>>> list(page_4_codes.keys())
['Tunnels on industrial and other minor lines',
 'Large bridges that are not officially tunnels but could appear to be so']
>>> key1 = 'Tunnels on industrial and other minor lines'
>>> page_4_dat = page_4_codes[key1]
>>> type(page_4_dat)
pandas.core.frame.DataFrame
>>> page_4_dat.head()
                     Name Other names, remarks  ... Length (metres) Length (note)
0             Ashes Quarry                      ...         56.6928
1        Ashey Down Quarry                      ...         33.8328
2  Baileycroft Quarry No 1                      ...         28.3464
3  Baileycroft Quarry No 2                      ...         21.0312
4           Basfords Hill                       ...         46.6344
[5 rows x 6 columns]
>>> key2 = 'Large bridges that are not officially tunnels but could appear to be so'
>>> page_4_dat_ = page_4_codes[key2]
>>> type(page_4_dat_)
pandas.core.frame.DataFrame
>>> page_4_dat_.head()
               Name Other names, remarks  ... Length (metres) Length (note)
0  A470/A472 (north)                      ...         35.6616
1  A470/A472 (south)                      ...         28.3464
2              A720                       ...        145.3896
3                A9       Aberdeen line  ...        141.7320
4                A9          Perth line  ...        146.3040
[5 rows x 8 columns]
```

4.2.3 Viaducts

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

A class for collecting codes of railway viaducts.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data;

defaults to None.

- **update** (*bool*) – Whether to check for updates to the catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.other_assets import Viaducts  # from pyrcs import Viaducts
>>> vdct = Viaducts()
>>> vdct.NAME
'Railway viaducts'
>>> vdct.URL
'http://www.railwaycodes.org.uk/viaducts/viaducts0.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
NAME	The name of the data.
URL	The URL of the main web page for the data.

Viaducts.KEY

Viaducts.**KEY: str = 'Viaducts'**

The key for accessing the data.

Viaducts.KEY_TO_LAST_UPDATED_DATE

Viaducts.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

Viaducts.NAME

Viaducts.**NAME: str = 'Railway viaducts'**

The name of the data.

Viaducts.URL

Viaducts.**URL: str = 'http://www.railwaycodes.org.uk/viaducts/viaducts0.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_codes(page_no[, ...])	Collects data of railway viaducts for a specific page number from the source web page.
fetch_codes([page_no, update, dump_dir, verbose])	Fetches data of railway viaducts.

Viaducts.collect_codes

Viaducts.**collect_codes**(*page_no*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects data of railway viaducts for a specific page number from the source web page.

Parameters

- **page_no** (*int | str*) – The page number to collect data from; valid values are 1, 2, 3 and 4.
- **confirmation_required** (*bool*) – Whether user confirmation is required; if confirmation_required=True (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Returns

A dictionary containing the data of railway viaducts for the specified page_no and the date of when the data was last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Viaducts  # from pyrcs import Viaducts
>>> vdct = Viaducts()
>>> page_1_codes = vdct.collect_codes(page_no=1)
>>> type(page_1_codes)
dict
>>> list(page_1_codes.keys())
['Page 1 (A-C)', 'Last updated date']
>>> page_1_dat = page_1_codes['Page 1 (A-C)']
>>> type(page_1_dat)
```

(continues on next page)

(continued from previous page)

```
pandas.core.frame.DataFrame
>>> page_1_dat.head()
            Name  ... Spans
0       7 Arches  ...     7
1        36 Arch  ...    36
2        42 Arch  ...
3           A698  ...     5
4  Abattoir Road  ...     8
[5 rows x 7 columns]
```

Viaducts.fetch_codes

Viaducts.**fetch_codes**(*page_no=None*, *update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of railway viaducts.

Parameters

- **page_no** (*int | str*) – The page number to collect data from; valid values are 1, 2, 3 and 4; defaults to None.
- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the data of railway viaducts and the date of when the data was last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Viaducts  # from pyrcs import Viaducts
>>> vdct = Viaducts()
>>> vdct_codes = vdct.fetch_codes()
>>> type(vdct_codes)
dict
>>> list(vdct_codes.keys())
['Viaducts', 'Last updated date']
>>> vdct.KEY
'Viaducts'
>>> vdct_codes_dat = vdct_codes[vdct.KEY]
>>> type(vdct_codes_dat)
dict
>>> list(vdct_codes_dat.keys())
['Page 1 (A-C)',
 'Page 2 (D-G)',
 'Page 3 (H-K)',
 'Page 4 (L-P)',
```

(continues on next page)

(continued from previous page)

```
 'Page 5 (Q-S)',
 'Page 6 (T-Z)']
>>> page_6_codes = vdct_codes_dat['Page 6 (T-Z)']
>>> type(page_6_codes)
pandas.core.frame.DataFrame
>>> page_6_codes.head()
         Name                                  Notes  ... End mileage Spans
0   Tadcaster  crosses River Wharfe; grade II listed  ...                11
1        Taff                     see Red Bridge  ...
2        Taff                                     ...
3  Taff River             also called Afon Taff  ...   170m 42ch
4  Taffs Well                     see River Taff  ...
[5 rows x 7 columns]
```

4.2.4 Stations

class pyrcs.other_assets.**Stations**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting railway station data.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.other_assets import Stations  # from pyrcs import Stations
>>> stn = Stations()
>>> stn.NAME
'Railway station data'
>>> stn.URL
'http://www.railwaycodes.org.uk/stations/station0.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.

continues on next page

Table 24 – continued from previous page

KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
KEY_TO_STN	The key for accessing the data of *Mileages, operators and grid coordinates*.
NAME	The name of the data.
URL	The URL of the main web page for the data.

Stations.KEY

Stations.**KEY: str = 'Stations'**

The key for accessing the data.

Stations.KEY_TO_LAST_UPDATED_DATE

Stations.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

Stations.KEY_TO_STN

Stations.**KEY_TO_STN: str = 'Mileages, operators and grid coordinates'**

The key for accessing the data of *Mileages, operators and grid coordinates*.

Stations.NAME

Stations.**NAME: str = 'Railway station data'**

The name of the data.

Stations.URL

Stations.**URL: str = 'http://www.railwaycodes.org.uk/stations/station0.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_catalogue([confirmation_required, ...])	Collects the catalogue of railway station data from the source web page.
collect_locations(initial[, ...])	Collects data of railway station locations (mileages, operators and grid coordinates) for a given initial letter.
fetch_catalogue([update, dump_dir, verbose])	Fetches the catalogue of railway station data.
fetch_locations([initial, update, dump_dir, ...])	Fetches data of railway station locations (mileages, operators and grid coordinates).

Stations.collect_catalogue

Stations.**collect_catalogue**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects the catalogue of railway station data from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the catalogue of railway station data, or `None` if no data catalogue is collected.

Return type

dict | None

Stations.collect_locations

Stations.**collect_locations**(*initial*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects data of railway station locations (mileages, operators and grid coordinates) for a given initial letter.

Parameters

- **initial** (*str*) – The initial letter (e.g. `'a'`, `'z'`) of railway station names.
- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `True`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the data of railway station locations whose initial letters are the given `initial` and date of when the data was last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Stations  # from pyrcs import Stations
>>> stn = Stations()
>>> stn_loc_a_codes = stn.collect_locations(initial='a')
>>> type(stn_loc_a_codes)
dict
>>> list(stn_loc_a_codes.keys())
['A', 'Last updated date']
>>> stn_loc_a_codes_dat = stn_loc_a_codes['A']
>>> type(stn_loc_a_codes_dat)
pandas.core.frame.DataFrame
>>> stn_loc_a_codes_dat.head()
      Station  ...                                    Former Operator
0  Abbey Wood  ...  London & South Eastern Railway from 1 April 20...
1  Abbey Wood  ...  London & South Eastern Railway from 1 April 20...
2        Aber  ...  Keolis Amey Operations/Gweithrediadau Keolis A...
3   Abercynon  ...  Keolis Amey Operations/Gweithrediadau Keolis A...
4   Abercynon  ...  Keolis Amey Operations/Gweithrediadau Keolis A...
[5 rows x 14 columns]
>>> stn_loc_a_codes_dat.columns.to_list()
['Station',
 'Station Note',
 'ELR',
 'Mileage',
 'Status',
 'Degrees Longitude',
 'Degrees Latitude',
 'Grid Reference',
 'CRS',
 'CRS Note',
 'Owner',
 'Former Owner',
 'Operator',
 'Former Operator']
>>> stn_loc_a_codes_dat[['Station', 'ELR', 'Mileage']].head()
      Station  ELR   Mileage
0  Abbey Wood  NKL  11m 43ch
1  Abbey Wood  XRS  24.458km
2        Aber  CAR   8m 69ch
3   Abercynon  CAM  16m 28ch
4   Abercynon  ABD  16m 28ch
```

Stations.fetch_catalogue

`Stations.`**`fetch_catalogue`**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches the catalogue of railway station data.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Returns
A dictionary containing the catalogue of railway station data, or `None` if no data catalogue is collected.

Return type
dict | None

Examples:

```
>>> from pyrcs.other_assets import Stations  # from pyrcs import Stations
>>> stn = Stations()
>>> stn_data_cat = stn.fetch_catalogue()
>>> type(stn_data_cat)
dict
>>> list(stn_data_cat.keys())
['Mileages, operators and grid coordinates',
 'Bilingual names',
 'Sponsored signs',
 'Not served by SFO',
 'International',
 'Trivia',
 'Access rights',
 'Barrier error codes',
 'London Underground',
 'Railnet']
```

Stations.fetch_locations

`Stations.`**`fetch_locations`**(*initial=None*, *update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of railway station locations (mileages, operators and grid coordinates).

Parameters

- **`initial`** (*`str`*) – The initial letter (e.g. `'a'`, `'z'`) of railway station names.
- **`update`** (*`bool`*) – Whether to check for updates to the package data; defaults to `False`.
- **`dump_dir`** (*`str | None`*) – The path to a directory where the data file will be saved; defaults to `None`.
- **`verbose`** (*`bool | int`*) – Whether to print relevant information to the console; defaults to `False`.

Returns
A dictionary containing the data of railway station locations and the date of when the data was last updated.

Return type
dict

Examples:

```
>>> from pyrcs.other_assets import Stations  # from pyrcs import Stations
>>> stn = Stations()
```

(continues on next page)

(continued from previous page)

```
>>> stn_loc_codes = stn.fetch_locations()
>>> type(stn_loc_codes)
dict
>>> list(stn_loc_codes.keys())
['Mileages, operators and grid coordinates', 'Last updated date']
>>> stn.KEY_TO_STN
'Mileages, operators and grid coordinates'
>>> stn_loc_codes_dat = stn_loc_codes[stn.KEY_TO_STN]
>>> type(stn_loc_codes_dat)
pandas.core.frame.DataFrame
>>> stn_loc_codes_dat.head()
      Station  ...                                     Former Operator
0  Abbey Wood  ...  London & South Eastern Railway from 1 April 20...
1  Abbey Wood  ...  London & South Eastern Railway from 1 April 20...
2        Aber  ...  Keolis Amey Operations/Gweithrediadau Keolis A...
3   Abercynon  ...  Keolis Amey Operations/Gweithrediadau Keolis A...
4   Abercynon  ...  Keolis Amey Operations/Gweithrediadau Keolis A...
[5 rows x 14 columns]
>>> stn_loc_codes_dat.columns.to_list()
['Station',
 'Station Note',
 'ELR',
 'Mileage',
 'Status',
 'Degrees Longitude',
 'Degrees Latitude',
 'Grid Reference',
 'CRS',
 'CRS Note',
 'Owner',
 'Former Owner',
 'Operator',
 'Former Operator']
>>> stn_loc_codes_dat[['Station', 'ELR', 'Mileage']].head()
      Station  ELR   Mileage
0  Abbey Wood  NKL  11m 43ch
1  Abbey Wood  XRS  24.458km
2        Aber  CAR   8m 69ch
3   Abercynon  CAM  16m 28ch
4   Abercynon  ABD  16m 28ch
```

4.2.5 Depots

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

A class for collecting data of depot codes.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.other_assets import Depots  # from pyrcs import Depots
>>> depots = Depots()
>>> depots.NAME
'Depot codes'
>>> depots.URL
'http://www.railwaycodes.org.uk/depots/depots0.shtm'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_1950_SYSTEM	The key for accessing the data of 1950 system (pre-TOPS) codes
KEY_TO_GWR	The key for accessing the data of GWR codes
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
KEY_TO_PRE_TOPS	The key for accessing the data of four digit pre-TOPS codes
KEY_TO_TOPS	The key for accessing the data of two character TOPS codes
NAME	The name of the data.
URL	The URL of the main web page for the data.

Depots.KEY

Depots.**KEY: str = 'Depots'**

The key for accessing the data.

Depots.KEY_TO_1950_SYSTEM

Depots.**KEY_TO_1950_SYSTEM: str = '1950 system (pre-TOPS)'**

The key for accessing the data of 1950 system (pre-TOPS) codes

Depots.KEY_TO_GWR

Depots.**KEY_TO_GWR: str = 'GWR'**

The key for accessing the data of GWR codes

Depots.KEY_TO_LAST_UPDATED_DATE

Depots.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

Depots.KEY_TO_PRE_TOPS

Depots.**KEY_TO_PRE_TOPS: str = 'Four digit pre-TOPS'**

The key for accessing the data of four digit pre-TOPS codes

Depots.KEY_TO_TOPS

Depots.**KEY_TO_TOPS: str = 'Two character TOPS'**

The key for accessing the data of two character TOPS codes

Depots.NAME

Depots.**NAME: str = 'Depot codes'**

The name of the data.

Depots.URL

Depots.**URL: str = 'http://www.railwaycodes.org.uk/depots/depots0.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_1950_system_codes([...])	Collects 1950 system (pre-TOPS) codes from the source web page.
collect_gwr_codes([confirmation_required, ...])	Collects Great Western Railway (GWR) depot codes from the source web page.
collect_pre_tops_codes([...])	Collects four-digit pre-TOPS codes from the source web page.
collect_tops_codes([confirmation_required, ...])	Collects two-character TOPS codes from the source web page.
fetch_1950_system_codes([update, dump_dir, ...])	Fetches data of 1950 system (pre-TOPS) codes.
fetch_codes([update, dump_dir, verbose])	Fetches data of depot codes.
fetch_gwr_codes([update, dump_dir, verbose])	Fetches data of Great Western Railway (GWR) depot codes.
fetch_pre_tops_codes([update, dump_dir, verbose])	Fetches data of four-digit pre-TOPS codes.
fetch_tops_codes([update, dump_dir, verbose])	Fetches data of two-character TOPS codes.

Depots.collect_1950_system_codes

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

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

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the 1950 system (pre-TOPS) codes and the date they were last updated, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.other_assets import Depots  # from pyrcs import Depots
>>> depots = Depots()
>>> s1950_codes = depots.collect_1950_system_codes()
To collect data of 1950 system (pre-TOPS) codes
? [No]|Yes: yes
>>> type(s1950_codes)
dict
>>> list(s1950_codes.keys())
['1950 system (pre-TOPS) codes', 'Last updated date']
>>> depots.KEY_TO_1950_SYSTEM
'1950 system (pre-TOPS) codes'
>>> s1950_codes_dat = s1950_codes[depots.KEY_TO_1950_SYSTEM]
>>> type(s1950_codes_dat)
pandas.core.frame.DataFrame
>>> s1950_codes_dat.head()
  Code        Depot name                                              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 f...
```

Depots.collect_gwr_codes

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

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

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the GWR depot codes and the date they were last updated, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.other_assets import Depots  # from pyrcs import Depots
>>> depots = Depots()
>>> gwr_codes = depots.collect_gwr_codes()
To collect data of GWR codes
? [No]|Yes: yes
>>> type(gwr_codes)
dict
>>> list(gwr_codes.keys())
['GWR codes', 'Last updated date']
>>> depots.KEY_TO_GWR
'GWR codes'
>>> gwr_codes_dat = gwr_codes[depots.KEY_TO_GWR]
>>> type(gwr_codes_dat)
dict
>>> list(gwr_codes_dat.keys())
['Alphabetical codes', 'Numerical codes']
>>> gwr_alpha_codes = gwr_codes_dat['Alphabetical codes']
>>> type(gwr_alpha_codes)
pandas.core.frame.DataFrame
>>> gwr_alpha_codes.head()
    Code   Depot name
0  ABEEG     Aberbeeg
1    ABG     Aberbeeg
2    AYN    Abercynon
3   ABDR     Aberdare
4    ABH  Aberystwyth
```

Depots.collect_pre_tops_codes

`Depots.`**`collect_pre_tops_codes`**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects four-digit pre-TOPS codes from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.

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

Returns

A dictionary containing the four-digit pre-TOPS codes and the date they were last updated, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.other_assets import Depots  # from pyrcs import Depots
>>> depots = Depots()
>>> fdpt_codes = depots.collect_pre_tops_codes()
To collect data of four digit pre-TOPS codes
? [No]|Yes: yes
>>> type(fdpt_codes)
dict
>>> list(fdpt_codes.keys())
['Four digit pre-TOPS codes', 'Last updated date']
>>> depots.KEY_TO_PRE_TOPS
'Four digit pre-TOPS codes'
>>> fdpt_codes_dat = fdpt_codes[depots.KEY_TO_PRE_TOPS]
>>> type(fdpt_codes_dat)
pandas.core.frame.DataFrame
>>> fdpt_codes_dat.head()
   Code             Depot name          Region  Main Works site
0  2000             Accrington  London Midland            False
1  2001   Derby Litchurch Lane  London Midland             True
2  2003              Blackburn  London Midland            False
3  2004  Bolton Trinity Street  London Midland            False
4  2006                Burnley  London Midland            False
```

Depots.collect_tops_codes

`Depots.`**`collect_tops_codes`**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects two-character TOPS codes from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the two-character TOPS codes and the date they were last updated, or `None` if no data is collected.

Return type
dict | None

Examples:

```
>>> from pyrcs.other_assets import Depots  # from pyrcs import Depots
>>> depots = Depots()
>>> tct_codes = depots.collect_tops_codes()
To collect data of two character TOPS codes
? [No]|Yes: yes
>>> type(tct_codes)
dict
>>> list(tct_codes.keys())
['Two character TOPS codes', 'Last updated date']
>>> depots.KEY_TO_TOPS
'Two character TOPS codes'
>>> tct_codes_dat = tct_codes[depots.KEY_TO_TOPS]
>>> type(tct_codes_dat)
pandas.core.frame.DataFrame
>>> tct_codes_dat.head()
  Code  ...                Notes
0   AB  ...          Closed 1987
1   AB  ...
2   AC  ...  Became WH from 1994
3   AC  ...
4   AD  ...
[5 rows x 5 columns]
```

Depots.fetch_1950_system_codes

Depots.**fetch_1950_system_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of 1950 system (pre-TOPS) codes.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns
A dictionary containing the 1950 system (pre-TOPS) codes and the date they were last updated.

Return type
dict

Examples:

```
>>> from pyrcs.other_assets import Depots  # from pyrcs import Depots
>>> depots = Depots()
>>> s1950_codes = depots.fetch_1950_system_codes()
```

(continues on next page)

(continued from previous page)

```
>>> type(s1950_codes)
dict
>>> list(s1950_codes.keys())
['1950 system (pre-TOPS) codes', 'Last updated date']
>>> depots.KEY_TO_1950_SYSTEM
'1950 system (pre-TOPS) codes'
>>> s1950_codes_dat = s1950_codes[depots.KEY_TO_1950_SYSTEM]
>>> type(s1950_codes_dat)
pandas.core.frame.DataFrame
>>> s1950_codes_dat.head()
  Code        Depot name                                              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 f...
```

Depots.fetch_codes

Depots.**fetch_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of depot codes.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the depot codes and the date they were last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Depots  # from pyrcs import Depots
>>> depots = Depots()
>>> depots_codes = depots.fetch_codes()
>>> type(depots_codes)
dict
>>> list(depots_codes.keys())
['Depots', 'Last updated date']
>>> depots.KEY
'Depots'
>>> depots_codes_dat = depots_codes[depots.KEY]
>>> type(depots_codes_dat)
dict
>>> list(depots_codes_dat.keys())
['1950 system (pre-TOPS) codes',
```

(continues on next page)

(continued from previous page)

```
 'Four digit pre-TOPS codes',
 'GWR codes',
 'Two character TOPS codes']
>>> depots.KEY_TO_PRE_TOPS
'Four digit pre-TOPS codes'
>>> depots_codes_dat[depots.KEY_TO_PRE_TOPS].head()
   Code             Depot name          Region  Main Works site
0  2000             Accrington  London Midland            False
1  2001   Derby Litchurch Lane  London Midland             True
2  2003              Blackburn  London Midland            False
3  2004  Bolton Trinity Street  London Midland            False
4  2006                Burnley  London Midland            False
>>> depots.KEY_TO_TOPS
'Two character TOPS codes'
>>> depots_codes_dat[depots.KEY_TO_TOPS].head()
  Code  ...                Notes
0   AB  ...          Closed 1987
1   AB  ...
2   AC  ...  Became WH from 1994
3   AC  ...
4   AD  ...
[5 rows x 5 columns]
```

Depots.fetch_gwr_codes

Depots.**fetch_gwr_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of Great Western Railway (GWR) depot codes.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the GWR depot codes and the date they were last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Depots  # from pyrcs import Depots
>>> depots = Depots()
>>> gwr_codes = depots.fetch_gwr_codes()
>>> type(gwr_codes)
dict
>>> list(gwr_codes.keys())
['GWR codes', 'Last updated date']
```

(continues on next page)

(continued from previous page)

```
>>> depots.KEY_TO_GWR
'GWR codes'
>>> gwr_codes_dat = gwr_codes[depots.KEY_TO_GWR]
>>> type(gwr_codes_dat)
dict
>>> list(gwr_codes_dat.keys())
['Alphabetical codes', 'Numerical codes']
>>> gwr_alpha_codes = gwr_codes_dat['Alphabetical codes']
>>> type(gwr_alpha_codes)
pandas.core.frame.DataFrame
>>> gwr_alpha_codes.head()
    Code   Depot name
0  ABEEG     Aberbeeg
1    ABG     Aberbeeg
2    AYN    Abercynon
3   ABDR     Aberdare
4    ABH  Aberystwyth
```

Depots.fetch_pre_tops_codes

Depots.**fetch_pre_tops_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of four-digit pre-TOPS codes.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the four-digit pre-TOPS codes and the date they were last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Depots  # from pyrcs import Depots
>>> depots = Depots()
>>> fdpt_codes = depots.fetch_pre_tops_codes()
>>> type(fdpt_codes)
dict
>>> list(fdpt_codes.keys())
['Four digit pre-TOPS codes', 'Last updated date']
>>> depots.KEY_TO_PRE_TOPS
'Four digit pre-TOPS codes'
>>> fdpt_codes_dat = fdpt_codes[depots.KEY_TO_PRE_TOPS]
>>> type(fdpt_codes_dat)
pandas.core.frame.DataFrame
```

(continues on next page)

(continued from previous page)

```
>>> fdpt_codes_dat.head()
   Code             Depot name          Region  Main Works site
0  2000             Accrington  London Midland            False
1  2001   Derby Litchurch Lane  London Midland             True
2  2003              Blackburn  London Midland            False
3  2004  Bolton Trinity Street  London Midland            False
4  2006                Burnley  London Midland            False
```

Depots.fetch_tops_codes

Depots.**fetch_tops_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of two-character TOPS codes.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the data of two-character TOPS codes and the date they were last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Depots  # from pyrcs import Depots
>>> depots = Depots()
>>> tct_codes = depots.fetch_tops_codes()
>>> type(tct_codes)
dict
>>> list(tct_codes.keys())
['Two character TOPS codes', 'Last updated date']
>>> depots.KEY_TO_TOPS
'Two character TOPS codes'
>>> tct_codes_dat = tct_codes[depots.KEY_TO_TOPS]
>>> type(tct_codes_dat)
pandas.core.frame.DataFrame
>>> tct_codes_dat.head()
  Code  ...                Notes
0   AB  ...          Closed 1987
1   AB  ...
2   AC  ...  Became WH from 1994
3   AC  ...
4   AD  ...
[5 rows x 5 columns]
```

4.2.6 Features

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

A class for collecting data of several infrastructure features, including *HABDs and WILDs*, *water troughs locations*, *telegraph code words* and *buzzer codes*.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.other_assets import Features  # from pyrcs import Features
>>> feats = Features()
>>> feats.NAME
'Infrastructure features'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_BUZZER	The key for accessing the data of *buzzer codes*.
KEY_TO_HABD_WILD	The key for accessing the data of *HABD* and *WILD*.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
KEY_TO_TELEGRAPH	The key for accessing the data of *telegraph codes*.
KEY_TO_TROUGH	The key for accessing the data of *water troughs*.
NAME	The name of the data.
URL	The URL of the main web page for the data.

Features.KEY

Features.**KEY: str = 'Features'**

The key for accessing the data.

Features.KEY_TO_BUZZER

Features.**KEY_TO_BUZZER: str = 'Buzzer codes'**

The key for accessing the data of *buzzer codes*.

Features.KEY_TO_HABD_WILD

Features.**KEY_TO_HABD_WILD: str = 'HABD and WILD'**

The key for accessing the data of *HABD* and *WILD*.

Features.KEY_TO_LAST_UPDATED_DATE

Features.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

Features.KEY_TO_TELEGRAPH

Features.**KEY_TO_TELEGRAPH: str = 'Telegraphic codes'**

The key for accessing the data of *telegraph codes*.

Features.KEY_TO_TROUGH

Features.**KEY_TO_TROUGH: str = 'Water troughs'**

The key for accessing the data of *water troughs*.

Features.NAME

Features.**NAME: str = 'Infrastructure features'**

The name of the data.

Features.URL

Features.**URL: str = 'http://www.railwaycodes.org.uk/features/habdwild.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

fetch_codes([update, dump_dir, verbose])	Fetches data of infrastructure features.

Features.fetch_codes

Features.**fetch_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of infrastructure features.

Including:

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

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the data of features codes and the date they were last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Features  # from pyrcs import Features
>>> feats = Features()
>>> feats_codes = feats.fetch_codes()
>>> type(feats_codes)
dict
>>> list(feats_codes.keys())
['Features', 'Last updated date']
>>> feats.KEY
'Features'
>>> feats_codes_dat = feats_codes[feats.KEY]
>>> type(feats_codes_dat)
dict
>>> list(feats_codes_dat.keys())
['Buzzer codes', 'HABD and WILD', 'Telegraphic codes', 'Water troughs']
>>> water_troughs_locations = feats_codes_dat[feats.KEY_TO_TROUGH]
>>> type(water_troughs_locations)
pandas.core.frame.DataFrame
>>> water_troughs_locations.head()
    ELR  ... Length (Yard)
0   BEI  ...           NaN
1   BHL  ...    620.000000
2  CGJ2  ...      0.666667
```

(continues on next page)

(continued from previous page)

```
3  CGJ6  ...    561.000000
4  CGJ6  ...    560.000000
[5 rows x 6 columns]
>>> hw_codes_dat = feats_codes_dat[feats.KEY_TO_HABD_WILD]
>>> type(hw_codes_dat)
dict
>>> list(hw_codes_dat.keys())
['HABD', 'WILD']
>>> habd_dat = hw_codes_dat['HABD']
>>> type(habd_dat)
pandas.core.frame.DataFrame
>>> habd_dat.head()
    ELR  ...                                              Notes
0  BAG2  ...
1  BAG2  ...  installed 29 September 1997, later moved to 74...
2  BAG2  ...                              previously at 74m 51ch
3  BAG2  ...                           removed 29 September 1997
4  BAG2  ...           present in 1969, later moved to 89m 00ch
[5 rows x 5 columns]
>>> wild_dat = hw_codes_dat['WILD']
>>> type(wild_dat)
pandas.core.frame.DataFrame
>>> wild_dat.head()
    ELR  ...                                              Notes
0  AYR3  ...
1  BAG2  ...
2  BML1  ...
3  BML1  ...
4  CGJ3  ...  moved to 183m 68ch from 8 September 2018 / mov...
[5 rows x 5 columns]
```

4.2.7 HabdWild

class pyrcs.other_assets.**HabdWild**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting data of HABDs and WILDs.

Note

- HABD: Hot axle box detector
- WILD: Wheel impact load detector

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.other_assets import HabdWild  # from pyrcs import HABDWILD
>>> hw = HabdWild()
>>> hw.NAME
'Hot axle box detectors (HABDs) and wheel impact load detectors (WILDs)'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
NAME	The name of the data.
URL	The URL of the main web page for the data.

HabdWild.KEY

HabdWild.**KEY: str = 'HABD and WILD'**

The key for accessing the data.

HabdWild.KEY_TO_LAST_UPDATED_DATE

HabdWild.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

HabdWild.NAME

HabdWild.**NAME: str = 'Hot axle box detectors (HABDs) and wheel impact load detectors (WILDs)'**

The name of the data.

HabdWild.URL

HabdWild.**URL: str = 'http://www.railwaycodes.org.uk/features/habdwild.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_codes([confirmation_required, ...])	Collects codes of HABDs and WILDs from the source web page.
fetch_codes([update, dump_dir, verbose])	Fetches codes of HABDs and WILDs.

HabdWild.collect_codes

HabdWild.**collect_codes**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects codes of HABDs and WILDs from the source web page.

Note

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

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the codes of HABDs and WILDs and the date they were last updated, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.other_assets import HabdWild  # from pyrcs import HABDWILD
>>> hw = HabdWild()
>>> hw_codes = hw.collect_codes()
To collect data of HABD and WILD
? [No]|Yes: yes
>>> type(hw_codes)
dict
>>> list(hw_codes.keys())
['HABD and WILD', 'Last updated date']
>>> hw.KEY
'HABD and WILD'
```

(continues on next page)

(continued from previous page)

```
>>> hw_codes_dat = hw_codes[hw.KEY]
>>> type(hw_codes_dat)
dict
>>> list(hw_codes_dat.keys())
['HABD', 'WILD']
>>> habd_dat = hw_codes_dat['HABD']
>>> type(habd_dat)
pandas.core.frame.DataFrame
>>> habd_dat.head()
    ELR  ...                                              Notes
0  BAG2  ...
1  BAG2  ...  installed 29 September 1997, later moved to 74...
2  BAG2  ...                              previously at 74m 51ch
3  BAG2  ...                           removed 29 September 1997
4  BAG2  ...           present in 1969, later moved to 89m 00ch
[5 rows x 5 columns]
>>> wild_dat = hw_codes_dat['WILD']
>>> type(wild_dat)
pandas.core.frame.DataFrame
>>> wild_dat.head()
    ELR  ...                                              Notes
0  AYR3  ...
1  BAG2  ...
2  BML1  ...
3  BML1  ...
4  CGJ3  ...  moved to 183m 68ch from 8 September 2018 / mov...
[5 rows x 5 columns]
```

HabdWild.fetch_codes

HabdWild.**fetch_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches codes of HABDs and WILDs.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the codes of HABDs and WILDs and the date they were last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import HabdWild  # from pyrcs import HABDWILD
>>> hw = HabdWild()
```

(continues on next page)

(continued from previous page)

```
>>> hw_codes = hw.fetch_codes()
>>> type(hw_codes)
dict
>>> list(hw_codes.keys())
['HABD and WILD', 'Last updated date']
>>> hw.KEY
'HABD and WILD'
>>> hw_codes_dat = hw_codes[hw.KEY]
>>> type(hw_codes_dat)
dict
>>> list(hw_codes_dat.keys())
['HABD', 'WILD']
>>> habd_dat = hw_codes_dat['HABD']
>>> type(habd_dat)
pandas.core.frame.DataFrame
>>> habd_dat.head()
    ELR  ...                                              Notes
0  BAG2  ...
1  BAG2  ...  installed 29 September 1997, later moved to 74...
2  BAG2  ...                              previously at 74m 51ch
3  BAG2  ...                           removed 29 September 1997
4  BAG2  ...           present in 1969, later moved to 89m 00ch
[5 rows x 5 columns]
>>> wild_dat = hw_codes_dat['WILD']
>>> type(wild_dat)
pandas.core.frame.DataFrame
>>> wild_dat.head()
    ELR  ...                                              Notes
0  AYR3  ...
1  BAG2  ...
2  BML1  ...
3  BML1  ...
4  CGJ3  ...  moved to 183m 68ch from 8 September 2018 / mov...
[5 rows x 5 columns]
```

4.2.8 WaterTroughs

class pyrcs.other_assets.**WaterTroughs**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting data of water troughs locations.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.

- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.other_assets import WaterTroughs  # from pyrcs import WaterTroughs
>>> wt = WaterTroughs()
>>> wt.NAME
'Water trough locations'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
NAME	The name of the data.
URL	The URL of the main web page for the data.

WaterTroughs.KEY

WaterTroughs.**KEY: str = 'Water troughs'**

The key for accessing the data.

WaterTroughs.KEY_TO_LAST_UPDATED_DATE

WaterTroughs.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

WaterTroughs.NAME

WaterTroughs.**NAME: str = 'Water trough locations'**

The name of the data.

WaterTroughs.URL

WaterTroughs.**URL: str = 'http://www.railwaycodes.org.uk/features/troughs.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_codes([confirmation_required, ...])	Collects codes of water troughs locations from the source web page.

continues on next page

Table 33 – continued from previous page

fetch_codes([update, dump_dir, verbose])	Fetches codes of water troughs locations.

WaterTroughs.collect_codes

WaterTroughs.**collect_codes**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects codes of water troughs locations from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if confirmation_required=True (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Returns

A dictionary containing the codes of water trough locations and the date they were last updated.

Return type

dict | None

Examples:

```
>>> from pyrcs.other_assets import WaterTroughs  # from pyrcs import WaterTroughs
>>> wt = WaterTroughs()
>>> wt_codes = wt.collect_codes()
To collect data of water troughs
? [No]|Yes: yes
>>> type(wt_codes)
dict
>>> list(wt_codes.keys())
['Water troughs', 'Last updated date']
>>> wt.KEY
'Water troughs'
>>> wt_codes_dat = wt_codes[wt.KEY]
>>> type(wt_codes_dat)
pandas.core.frame.DataFrame
>>> wt_codes_dat.head()
    ELR  ... Length (Yard)
0   BEI  ...           NaN
1   BHL  ...    620.000000
2  CGJ2  ...      0.666667
3  CGJ6  ...    561.000000
4  CGJ6  ...    560.000000
[5 rows x 6 columns]
```

WaterTroughs.fetch_codes

WaterTroughs.**fetch_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches codes of water troughs locations.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the codes of water trough locations and the date they were last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import WaterTroughs  # from pyrcs import WaterTroughs
>>> wt = WaterTroughs()
>>> wt_codes = wt.fetch_codes()
>>> type(wt_codes)
dict
>>> list(wt_codes.keys())
['Water troughs', 'Last updated date']
>>> wt.KEY
'Water troughs'
>>> wt_codes_dat = wt_codes[wt.KEY]
>>> type(wt_codes_dat)
pandas.core.frame.DataFrame
>>> wt_codes_dat.head()
    ELR  ... Length (Yard)
0   BEI  ...           NaN
1   BHL  ...    620.000000
2  CGJ2  ...      0.666667
3  CGJ6  ...    561.000000
4  CGJ6  ...    560.000000
[5 rows x 6 columns]
```

4.2.9 Telegraph

class pyrcs.other_assets.**Telegraph**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting data of telegraph code words.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.

- **update** (*bool*) – Whether to check for updates to the catalogue; defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `True`.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.other_assets import Telegraph  # from pyrcs import Telegraph
>>> tel = Telegraph()
>>> tel.NAME
'Telegraph code words'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
NAME	The name of the data.
URL	The URL of the main web page for the data.

Telegraph.KEY

Telegraph.**KEY: str = 'Telegraphic codes'**

The key for accessing the data.

Telegraph.KEY_TO_LAST_UPDATED_DATE

Telegraph.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

Telegraph.NAME

Telegraph.**NAME: str = 'Telegraph code words'**

The name of the data.

Telegraph.URL

Telegraph.**URL: str = 'http://www.railwaycodes.org.uk/features/telegraph.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_codes([confirmation_required, ...])	Collects data of telegraph code words from the source web page.
fetch_codes([update, dump_dir, verbose])	Fetches data of telegraph code words.

Telegraph.collect_codes

Telegraph.**collect_codes**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects data of telegraph code words from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the data of telegraph code words and the date they were last updated, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.other_assets import Telegraph  # from pyrcs import Telegraph
>>> tel = Telegraph()
>>> tel_codes = tel.collect_codes()
To collect data of telegraphic codes
? [No]|Yes: yes
>>> type(tel_codes)
dict
>>> list(tel_codes.keys())
['Telegraphic codes', 'Last updated date']
>>> tel.KEY
'Telegraphic codes'
>>> tel_codes_dat = tel_codes[tel.KEY]
>>> type(tel_codes_dat)
dict
>>> list(tel_codes_dat.keys())
```

(continues on next page)

(continued from previous page)

```
['Official codes', 'Unofficial codes']
>>> tel_official_codes = tel_codes_dat['Official codes']
>>> type(tel_official_codes)
pandas.core.frame.DataFrame
>>> tel_official_codes.head()
      Code  ...                                In use
0    ABACK  ...     cross industry term used in 1939
1    ABASE  ...                              GWR, 1939
2  ABREAST  ...  GWR, 1939 / Railway Executive, 1950
3  ABREAST  ...   British Transport Commission, 1958
4   ABSENT  ...                              GWR, 1939
[5 rows x 3 columns]
>>> tel_unofficial_codes = tel_codes_dat['Unofficial codes']
>>> type(tel_unofficial_codes)
pandas.core.frame.DataFrame
>>> tel_unofficial_codes.head()
      Code                           Unofficial description
0  CRANKEX                                      [See KRANKEX]
1  DRUNKEX  Saturday night special train (usually a DMU) t...
2     GYFO    Strongly urge all speed ('Get your finger out')
3  KRANKEX  Special train with interesting routing or trac...
4   MYSTEX  Special excursion going somewhere no one reall...
```

Telegraph.fetch_codes

Telegraph.**fetch_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of telegraph code words.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to False.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

A dictionary containing the data of telegraph code words and the date they were last updated.

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Telegraph  # from pyrcs import Telegraph
>>> tel = Telegraph()
>>> tel_codes = tel.fetch_codes()
>>> type(tel_codes)
dict
>>> list(tel_codes.keys())
['Telegraphic codes', 'Last updated date']
```

(continues on next page)

(continued from previous page)

```
>>> tel.KEY
'Telegraphic codes'
>>> tel_codes_dat = tel_codes[tel.KEY]
>>> type(tel_codes_dat)
dict
>>> list(tel_codes_dat.keys())
['Official codes', 'Unofficial codes']
>>> tel_official_codes = tel_codes_dat['Official codes']
>>> type(tel_official_codes)
pandas.core.frame.DataFrame
>>> tel_official_codes.head()
      Code  ...                                In use
0    ABACK  ...     cross industry term used in 1939
1    ABASE  ...                           GWR, 1939
2  ABREAST  ...  GWR, 1939 / Railway Executive, 1950
3  ABREAST  ...   British Transport Commission, 1958
4   ABSENT  ...                           GWR, 1939
[5 rows x 3 columns]
>>> tel_unofficial_codes = tel_codes_dat['Unofficial codes']
>>> type(tel_unofficial_codes)
pandas.core.frame.DataFrame
>>> tel_unofficial_codes.head()
      Code                          Unofficial description
0  CRANKEX                                     [See KRANKEX]
1  DRUNKEX  Saturday night special train (usually a DMU) t...
2     GYFO    Strongly urge all speed ('Get your finger out')
3  KRANKEX  Special train with interesting routing or trac...
4   MYSTEX  Special excursion going somewhere no one reall...
```

4.2.10 Buzzer

class pyrcs.other_assets.**Buzzer**(*data_dir=None*, *update=False*, *verbose=True*)

A class for collecting data of buzzer codes.

Parameters

- **data_dir** (*str | None*) – The name of the directory for storing the data; defaults to None.
- **update** (*bool*) – Whether to check for updates to the catalogue; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to True.

Variables

- **catalogue** (*dict*) – The catalogue of the data.
- **last_updated_date** (*str*) – The date when the data was last updated.
- **data_dir** (*str*) – The path to the directory containing the data.
- **current_data_dir** (*str*) – The path to the current data directory.

Examples:

```
>>> from pyrcs.other_assets import Buzzer  # from pyrcs import Buzzer
>>> buz = Buzzer()
>>> buz.NAME
'Buzzer codes'
```

Attributes

1anti-flashwhitewhite

KEY	The key for accessing the data.
KEY_TO_LAST_UPDATED_DATE	The key used to reference the last updated date in the data.
NAME	The name of the data.
URL	The URL of the main web page for the data.

Buzzer.KEY

Buzzer.**KEY: str = 'Buzzer codes'**

The key for accessing the data.

Buzzer.KEY_TO_LAST_UPDATED_DATE

Buzzer.**KEY_TO_LAST_UPDATED_DATE: str = 'Last updated date'**

The key used to reference the last updated date in the data.

Buzzer.NAME

Buzzer.**NAME: str = 'Buzzer codes'**

The name of the data.

Buzzer.URL

Buzzer.**URL: str = 'http://www.railwaycodes.org.uk/features/buzzer.shtm'**

The URL of the main web page for the data.

Methods

1anti-flashwhitewhite

collect_codes([confirmation_required, ...])	Collects data of buzzer codes from the source web page.
fetch_codes([update, dump_dir, verbose])	Fetches data of buzzer codes.

Buzzer.collect_codes

Buzzer.**collect_codes**(*confirmation_required=True*, *verbose=False*, *raise_error=False*)

Collects data of buzzer codes from the source web page.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required; if `confirmation_required=True` (default), prompts the user for confirmation before proceeding with data collection.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns

A dictionary containing the data of buzzer codes and the date they were last updated, or `None` if no data is collected.

Return type

dict | None

Examples:

```
>>> from pyrcs.other_assets import Buzzer  # from pyrcs import Buzzer
>>> buz = Buzzer()
>>> buz_codes = buz.collect_codes()
To collect data of Buzzer codes
? [No]|Yes: yes
>>> type(buz_codes)
dict
>>> list(buz_codes.keys())
['Buzzer codes', 'Last updated date']
>>> buz.KEY
'Buzzer codes'
>>> buz_codes_dat = buz_codes[buz.KEY]
>>> type(buz_codes_dat)
pandas.core.frame.DataFrame
>>> buz_codes_dat.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
```

Buzzer.fetch_codes

Buzzer.**fetch_codes**(*update=False*, *dump_dir=None*, *verbose=False*, ***kwargs*)

Fetches data of buzzer codes.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **dump_dir** (*str | None*) – The path to a directory where the data file will be saved; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Returns

A dictionary containing the data of buzzer codes and the date they were last updated

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Buzzer  # from pyrcs import Buzzer
>>> buz = Buzzer()
>>> buz_codes = buz.fetch_codes()
>>> type(buz_codes)
dict
>>> list(buz_codes.keys())
['Buzzer codes', 'Last updated date']
>>> buz.KEY
'Buzzer codes'
>>> buz_codes_dat = buz_codes[buz.KEY]
>>> type(buz_codes_dat)
pandas.core.frame.DataFrame
>>> buz_codes_dat.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
```

Chapter 5

Modules

The following modules provide supporting functions and utilities for the subpackages *line_data* and *other_assets*.

1anti-flashwhitewhite

parser	Parses web-page contents.
converter	Converts data to a certain format.
collector	Collects data of railway codes.
utils	Provides a number of helper functions.

5.1 parser

Parses web-page contents.

5.1.1 Preprocess contents

1anti-flashwhitewhite

parse_tr(trs, ths[, sep, as_dataframe])	Parses a list of HTML <tr> elements and extracts data from a table.
parse_table(source[, parser, as_dataframe])	Parses HTML <tr> elements to create a table from the given source.
parse_date(str_date[, as_date_type])	Parses a string representation of a date into a formatted date.

parse_tr

pyrcs.parser.**parse_tr**(*trs*, *ths*, *sep=' / '*, *as_dataframe=False*)

Parses a list of HTML <tr> elements and extracts data from a table.

This function processes the rows from a table (<tr> tags) and assigns them to corresponding column headers (<th> tags). It can return the data either as a list of lists or as a dataframe.

See also [PT-1].

Parameters

- **trs** (*bs4.ResultSet | list*) – The content of <tr> tags from a web page table.
- **ths** (*list | bs4.element.Tag*) – A list of column names (typically from <th> tags) for the table.
- **sep** (*str | None*) – The separator to replace any separators found in the raw data; defaults to ' / '.
- **as_dataframe** (*bool*) – If True, returns the data as a Pandas DataFrame; defaults to False.

Returns

A list of lists representing rows of the table, or a dataframe if as_dataframe is True.

Return type

pandas.DataFrame | list[list]

Examples:

```
>>> from pyrcs.parser import parse_tr
>>> import requests
>>> import bs4
>>> example_url = 'http://www.railwaycodes.org.uk/elrs/elra.shtm'
>>> source = requests.get(example_url)
>>> parsed_text = bs4.BeautifulSoup(source.content, 'html.parser')
>>> ths_dat = [th.text for th in parsed_text.find_all('th')]
>>> trs_dat = parsed_text.find_all(name='tr')
>>> tables_list = parse_tr(trs=trs_dat, ths=ths_dat)  # returns a list of lists
>>> type(tables_list)
list
>>> len(tables_list) // 100
1
>>> tables_list[0]
['AAL',
 'Ashendon and Aynho Line',
 '0.00 - 18.29',
 'Ashendon Junction',
 'Now NAJ3']
```

parse_table

pyrcs.parser.**parse_table**(*source*, *parser='html.parser'*, *as_dataframe=False*)

Parses HTML <tr> elements to create a table from the given source.

This function extracts data from the <thead> and <tbody> elements of an HTML table and processes it into a list of lists (rows of the table) or a dataframe.

Parameters

- **source** (*requests.Response*) – The response object containing the HTML table from a requested URL.

- **parser** (*str*) – The parser to use for processing the HTML; options are `'html.parser'` (default), `'html5lib'` or `'lxml'`.
- **as_dataframe** (*bool*) – If `True`, the parsed data is returned as a dataframe. If `False`, it returns a list of lists and column names; defaults to `False`.

Returns
A tuple containing a list of column names and a list of lists representing rows of the table; if `as_dataframe=True`, returns a dataframe.

Return type
tuple[list, list] | pandas.DataFrame | list

Examples:

```
>>> from pyrcs.parser import parse_table
>>> import requests
>>> source_dat = requests.get(url='http://www.railwaycodes.org.uk/elrs/elra.shtm')
>>> columns_dat, records_dat = parse_table(source_dat)
>>> columns_dat
['ELR', 'Line name', 'Mileages', 'Datum', 'Notes']
>>> type(records_dat)
list
>>> len(records_dat) // 100
1
>>> records_dat[0]
['AAL',
 'Ashendon and Aynho Line',
 '0.00 - 18.29',
 'Ashendon Junction',
 'Now NAJ3']
```

parse_date

pyrcs.parser.**parse_date**(*str_date*, *as_date_type=False*)

Parses a string representation of a date into a formatted date.

This function attempts to parse a string date (even with slight errors or non-standard formats) into either a string in the "YYYY-MM-DD" format or a datetime.date object.

Parameters

- **str_date** (*str*) – The date as a string, whose format can vary and may include month names or other elements.
- **as_date_type** (*bool*) – If `True`, returns the result as a *datetime.date* object; if `False` (default), returns the result as a formatted string.

Returns
The parsed date either as a string in "YYYY-MM-DD" format or as a date object.

Return type
str | datetime.date

Examples:

```
>>> from pyrcs.parser import parse_date
>>> str_date_dat = '2020-01-01'
>>> parse_date(str_date_dat)
'2020-01-01'
>>> str_date_dat = '2020-jan-01'
>>> parse_date(str_date_dat)
'2020-01-01'
>>> parse_date(str_date_dat, as_date_type=True)
datetime.date(2020, 1, 1)
```

5.1.2 Extract information

1anti-flashwhitewhite

get_site_map([update, ...])	Gets the site map.
get_last_updated_date(url[, parsed, ...])	Gets the last update date of a specified web page.
get_financial_year(date)	Gets the financial year of a given date.
get_catalogue(url[, update, json_it, ...])	Gets the catalogue of items from the main page of a data cluster.
get_category_menu(name[, update, ...])	Gets a menu of the available classes from the specified URL.
get_page_catalogue(url[, head_tag_name, ...])	Gets the catalogue of features from the main page of a data cluster.
get_heading_text(heading_tag[, elem_tag_name])	Gets the text from a given HTML heading tag.
get_hypertext(hypertext_tag[, ...])	Gets hyperlinked text from a specified HTML tag.
get_introduction(url[, delimiter, update, ...])	Gets the introduction section of a specified web page.

get_site_map

pyrcs.parser.**get_site_map**(*update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=True*)

Gets the site map.

Parameters

- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **confirmation_required** (*bool*) – Whether user confirmation is required before proceeding; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False`, the error will be suppressed; defaults to `True`.

Returns
A dictionary containing the data of site map.

Return type
dict | None

Examples:

```
>>> from pyrcs.parser import get_site_map
>>> site_map = get_site_map()
>>> type(site_map)
dict
>>> list(site_map.keys())
['Home',
 'Line data',
 'Other assets',
 '"Legal/financial" lists',
 'Miscellaneous']
>>> site_map['Home']
{'index': 'http://www.railwaycodes.org.uk/index.shtml'}
```

get_last_updated_date

pyrcs.parser.**get_last_updated_date**(*url*, *parsed=True*, *as_date_type=False*, *verbose=False*, *raise_error=True*)

Gets the last update date of a specified web page.

This function extracts the date when the given web page was last updated. The date can be returned as a string or a date object.

Parameters

- **url** (*str*) – The URL of the web page for which the last update date is requested.
- **parsed** (*bool*) – Whether to reformat the date into a standardized format (`YYYY-MM-DD`); defaults to `True`.
- **as_date_type** (*bool*) – If `True`, the date is returned as a datetime.date object; if `False` (default), it's returned as a string.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False`, the error will be suppressed; defaults to `True`.

Returns
The last update date of the specified web page, or `None` if this information is not available on the web page.

Return type
str | datetime.date | None

Examples:

```
>>> from pyrcs.parser import get_last_updated_date
>>> url = 'http://www.railwaycodes.org.uk/crs/CRSa.shtm'
>>> last_upd_date = get_last_updated_date(url=url, parsed=True, as_date_type=False)
>>> type(last_upd_date)
str
>>> last_upd_date = get_last_updated_date(url=url, parsed=True, as_date_type=True)
>>> type(last_upd_date)
datetime.date
>>> url = 'http://www.railwaycodes.org.uk/linedatamenu.shtm'
>>> last_upd_date = get_last_updated_date(url=url, verbose=True)
Information of the last update date not available.
```

get_financial_year

pyrcs.parser.**get_financial_year**(*date*)

Gets the financial year of a given date.

The financial year runs from 1st April to 31st March of the following year. This function takes a date and determines the financial year it falls into.

Parameters

date (*datetime.datetime*) – The date for which the financial year is to be determined.

Returns

The financial year of the given `date`.

Return type

int

Examples:

```
>>> from pyrcs.parser import get_financial_year
>>> import datetime
>>> financial_year = get_financial_year(date=datetime.datetime(2021, 3, 31))
>>> financial_year
2020
```

get_catalogue

pyrcs.parser.**get_catalogue**(*url*, *update=False*, *json_it=True*, *verbose=False*, *raise_error=False*)

Gets the catalogue of items from the main page of a data cluster.

This function scrapes a catalogue of entries (typically hyperlinks) from a specified URL. It offers the option to save the catalogue as a JSON file.

Parameters

- **url** (*str*) – The URL of the main page of a data cluster.
- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **json_it** (*bool*) – Whether to save the catalogue as a JSON file; defaults to `True`.

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

Returns

The catalogue in the form of a dictionary, where keys are entry titles and values are URLs, or `None` if the operation is unsuccessful.

Return type

dict | None

Examples:

```
>>> from pyrcs.parser import get_catalogue
>>> elr_cat = get_catalogue(url='http://www.railwaycodes.org.uk/elrs/elr0.shtm')
>>> type(elr_cat)
dict
>>> list(elr_cat.keys())[:5]
['Introduction', 'A', 'B', 'C', 'D']
>>> list(elr_cat.keys())[-5:]
['Lines without codes',
 'ELR/LOR converter',
 'LUL system',
 'DLR system',
 'Canals']
>>> location_code_cat = get_catalogue(url='http://www.railwaycodes.org.uk/crs/crs0.shtm')
>>> type(location_code_cat)
dict
>>> list(location_code_cat.keys())[:5]
['Introduction', 'A', 'B', 'C', 'D']
>>> list(location_code_cat.keys())[-5:]
['W', 'X', 'Y', 'Z', 'Other systems']
```

get_category_menu

pyrcs.parser.**get_category_menu**(*name*, *update=False*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Gets a menu of the available classes from the specified URL.

This function scrapes a web page for available classes (typically categorised hyperlinks) and returns them as a dictionary. It also provides options to update the catalogue and save it as a JSON file.

Parameters

- **name** (*str*) – The name of the data category.
- **update** (*bool*) – Whether to check for updates to the package data; defaults to `True`.
- **confirmation_required** (*bool*) – Whether user confirmation is required before proceeding; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

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

Returns
A category menu in dictionary form, where keys are data cluster names and values are URLs.

Return type
dict | None

Examples:

```
>>> from pyrcs.parser import get_category_menu
>>> menu = get_category_menu(name='Line data')
>>> type(menu)
dict
>>> list(menu.keys())
['Line data']
>>> len(menu['Line data'])
7
```

get_page_catalogue

pyrcs.parser.**get_page_catalogue**(*url*, *head_tag_name='nav'*, *head_tag_txt='Jump to:'*, *feature_tag_name='h3'*, *verbose=False*, *raise_error=False*)

Gets the catalogue of features from the main page of a data cluster.

This function extracts structured data (features) from a web page by parsing specific tags, typically used for features like headings and links in railway-related databases.

Parameters

- **url** (*str*) – The URL of the main page of a data cluster.
- **head_tag_name** (*str*) – The tag name of the feature list at the top of the page; defaults to `'nav'`.
- **head_tag_txt** (*str*) – Text contained in the head tag; defaults to `'Jump to:'`.
- **feature_tag_name** (*str*) – The tag name of the headings of each feature; defaults to `'h3'`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
A dataframe containing the page's feature catalogue with columns for feature, URL and heading.

Return type
pandas.DataFrame

Examples:

```
>>> from pyrcs.parser import get_page_catalogue
>>> from pyhelpers.settings import pd_preferences
>>> pd_preferences(max_columns=1)
>>> elec_url = 'http://www.railwaycodes.org.uk/electrification/mast_prefix2.shtm'
>>> elec_catalogue = get_page_catalogue(elec_url)
>>> elec_catalogue
                                            Feature  ...
0                                    Beamish Tramway  ...
1                                  Birkenhead Tramway  ...
2                          Black Country Living Museum  ...
3                                  Blackpool Tramway  ...
4   Brighton and Rottingdean Seashore Electric Rai...  ...
..                                                    ...  ...
17                                   Seaton Tramway  ...
18                                 Sheffield Supertram  ...
19                           Snaefell Mountain Railway  ...
20  Summerlee, Museum of Scottish Industrial Life ...  ...
21                                   Tyne & Wear Metro  ...
[22 rows x 3 columns]
>>> elec_catalogue.columns.to_list()
['Feature', 'URL', 'Heading']
```

get_heading_text

pyrcs.parser.**get_heading_text**(*heading_tag*, *elem_tag_name='em'*)

Gets the text from a given HTML heading tag.

Parameters

- **heading_tag** (*bs4.element.Tag*) – The HTML tag of a heading element.
- **elem_tag_name** (*str*) – The tag name of an inner element within the heading; defaults to `'em'`.

Returns

Cleaned text of the heading tag.

Return type

str

Examples:

```
>>> from pyrcs.parser import get_heading_text
>>> from pyrcs.line_data import Electrification
>>> elec = Electrification()
>>> url = elec.catalogue[elec.KEY_TO_INDEPENDENT_LINES]
>>> source = requests.get(url=url, headers=fake_requests_headers())
>>> soup = bs4.BeautifulSoup(markup=source.content, features='html.parser')
>>> h3 = soup.find('h3')
>>> h3_text = get_heading_text(heading_tag=h3, elem_tag_name='em')
>>> h3_text
'Beamish Tramway'
```

get_hypertext

pyrcs.parser.**get_hypertext**(*hypertext_tag*, *hyperlink_tag_name='a'*, *md_style=True*)

Gets hyperlinked text from a specified HTML tag.

This function scrapes hypertext content, optionally returning it in Markdown format if requested.

Parameters

- **hypertext_tag** (*bs4.element.Tag | bs4.element.PageElement*) – The tag containing hyperlinked text.
- **hyperlink_tag_name** (*str*) – The tag name of the hyperlink within the hypertext; defaults to `'a'`.
- **md_style** (*bool*) – Whether to return the hypertext in Markdown style, defaults to `True`.

Returns

The hypertext.

Return type

str

Examples:

```
>>> from pyrcs.parser import get_hypertext
>>> from pyrcs.line_data import Electrification
>>> import bs4
>>> import requests
>>> elec = Electrification()
>>> url = elec.catalogue[elec.KEY_TO_INDEPENDENT_LINES]
>>> source = requests.get(url)
>>> soup = bs4.BeautifulSoup(source.content, 'html.parser')
>>> h3 = soup.find('h3')
>>> p = h3.find_all_next('p')[8]
>>> p
<p>Croydon Tramlink mast references can be found on the <a href="http://www.croydon-traml...
>>> hyper_txt = get_hypertext(hypertext_tag=p, md_style=True)
>>> hyper_txt
'Croydon Tramlink mast references can be found on the [Croydon Tramlink Unofficial Site](...
```

get_introduction

pyrcs.parser.**get_introduction**(*url*, *delimiter='\n'*, *update=False*, *verbose=False*, *raise_error=False*)

Gets the introduction section of a specified web page.

This function scrapes the introduction text from the given URL, typically used to summarise data clusters.

Parameters

- **url** (*str*) – The URL of the web page (usually the main page of a data cluster).

- **delimiter** (*str*) – The delimiter used to separate paragraphs in the returned content; defaults to `'\n'` (newline).
- **update** (*bool*) – Whether to check for updates to the package data; defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Returns
The introductory text from the web page, formatted with the specified delimiter.

Return type
str

Examples:

```
>>> from pyrcs.parser import get_introduction
>>> bridges_url = 'http://www.railwaycodes.org.uk/bridges/bridges0.shtm'
>>> intro_text = get_introduction(url=bridges_url)
>>> intro_text
"There are thousands of bridges over and under the railway system. These pages attempt to...
```

5.2 converter

Converts data to a certain format.

5.2.1 Convert mileage data

1anti-flashwhitewhite

fix_mileage(mileage)	Fixes mileage data (associated with an ELR).
yard_to_mileage(yard[, as_str])	Converts yards to mileages.
mileage_to_yard(mileage)	Converts mileages to yards.
mile_chain_to_mileage(mile_chain)	Converts <*miles*>.<*chains*> to <*miles*>.<*yards*>.
mileage_to_mile_chain(mileage)	Converts <*miles*>.<*yards*> to <*miles*>.<*chains*>.
mile_yard_to_mileage(mile, yard[, as_numeric])	Converts mile and yard to mileage.
mileage_str_to_num(mileage)	Converts string-type mileage to its corresponding numerical value.
mileage_num_to_str(mileage)	Converts numerical-type Network Rail mileage to string-type one.
shift_mileage_by_yard(mileage, shift_yards)	Shifts the given mileage by a specified number of yards.

fix_mileage

pyrcs.converter.**fix_mileage**(*mileage*)

Fixes mileage data (associated with an ELR).

Parameters
mileage (*str | float | None*) – The mileage data.

Returns
The fixed mileage data in the form of <*miles*>.<*yards*>.

Return type
str

Examples:

```
>>> from pyrcs.converter import fix_mileage
>>> fixed_mileage = fix_mileage(mileage=29.011)
>>> fixed_mileage
'29.0110'
>>> fixed_mileage = fix_mileage(mileage='.1100')
>>> fixed_mileage
'0.1100'
>>> fixed_mileage = fix_mileage(mileage=29)
>>> fixed_mileage
'29.0000'
```

yard_to_mileage

pyrcs.converter.**yard_to_mileage**(*yard*, *as_str=True*)

Converts yards to mileages.

Parameters

- **yard** (*int | float | None*) – The yard data.
- **as_str** (*bool*) – Whether to return as a string value; defaults to `True`.

Returns
The mileage in the form of <*miles*>.<*yards*> or <*miles*>.<*yards*>.

Return type
str | float

Examples:

```
>>> from pyrcs.converter import yard_to_mileage
>>> mileage_dat = yard_to_mileage(yard=396)
>>> mileage_dat
'0.0396'
>>> mileage_dat = yard_to_mileage(yard=396, as_str=False)
>>> mileage_dat
0.0396
>>> mileage_dat = yard_to_mileage(yard=None)
```

(continues on next page)

(continued from previous page)

```
>>> mileage_dat
''
>>> mileage_dat = yard_to_mileage(yard=1760)
>>> mileage_dat
'1.0000'
>>> mileage_dat = yard_to_mileage(yard=12330)
>>> mileage_dat
'7.0010'
```

mileage_to_yard

pyrcs.converter.**mileage_to_yard**(*mileage*)

Converts mileages to yards.

Parameters
mileage (*float | int | str*) – The mileage (used by Network Rail).

Returns
The corresponding yards.

Return type
int

Examples:

```
>>> from pyrcs.converter import mileage_to_yard
>>> yards_dat = mileage_to_yard(mileage='0.0396')
>>> yards_dat
396
>>> yards_dat = mileage_to_yard(mileage=0.0396)
>>> yards_dat
396
>>> yards_dat = mileage_to_yard(mileage=1.0396)
>>> yards_dat
2156
```

mile_chain_to_mileage

pyrcs.converter.**mile_chain_to_mileage**(*mile_chain*)

Converts <*miles*>.<*chains*> to <*miles*>.<*yards*>.

Parameters
mile_chain (*str | numpy.nan | None*) – Mileage data presented in the form of <*miles*>.<*chains*>.

Returns
The corresponding mileage in the form of '<miles>.<yards>'.

Return type
str

Examples:

```
>>> from pyrcs.converter import mile_chain_to_mileage
>>> # AAM 0.18 Tewkesbury Junction with ANZ (84.62)
>>> mileage_data = mile_chain_to_mileage(mile_chain='0.18')
>>> mileage_data
'0.0396'
>>> # None, nan or ''
>>> mileage_data = mile_chain_to_mileage(mile_chain=None)
>>> mileage_data
''
```

mileage_to_mile_chain

pyrcs.converter.**mileage_to_mile_chain**(*mileage*)

Converts <*miles*>.<*yards*> to <*miles*>.<*chains*>.

Parameters
mileage (*str | numpy.nan | None*) – The mileage data presented in the form of <*miles*>.<*yards*>.

Returns
The corresponding mileage presented in the form of <*miles*>.<*chains*>.

Return type
str

Examples:

```
>>> from pyrcs.converter import mileage_to_mile_chain
>>> mile_chain_data = mileage_to_mile_chain(mileage='0.0396')
>>> mile_chain_data
'0.18'
>>> mile_chain_data = mileage_to_mile_chain(mileage=1.0396)
>>> mile_chain_data
'1.18'
>>> # None, nan or ''
>>> miles_chains_dat = mileage_to_mile_chain(mileage=None)
>>> miles_chains_dat
''
```

mile_yard_to_mileage

pyrcs.converter.**mile_yard_to_mileage**(*mile*, *yard*, *as_numeric=True*)

Converts mile and yard to mileage.

Parameters

- **mile** (*float | int*) – The mile data.
- **yard** (*float | int*) – The yard data.
- **as_numeric** (*bool*) – Whether to return a numeric value; defaults to `True`.

Returns
The corresponding mileage data in the form of <*miles*>.<*yards*>.

Return type
str | float

Examples:

```
>>> from pyrcs.converter import mile_yard_to_mileage
>>> m, y = 10, 1500
>>> mileage_data = mile_yard_to_mileage(mile=m, yard=y)
>>> mileage_data
10.15
>>> mileage_data = mile_yard_to_mileage(mile=m, yard=y, as_numeric=False)
>>> mileage_data
'10.1500'
>>> m, y = 10, 500
>>> mileage_data = mile_yard_to_mileage(mile=m, yard=y, as_numeric=False)
>>> mileage_data
'10.0500'
```

mileage_str_to_num

pyrcs.converter.**mileage_str_to_num**(*mileage*)

Converts string-type mileage to its corresponding numerical value.

Parameters
mileage (*str*) – The string-type mileage data in the form of <*miles*>.<*yards*>.

Returns
The corresponding numerical-type mileage.

Return type
float

Examples:

```
>>> from pyrcs.converter import mileage_str_to_num
>>> mileage_num = mileage_str_to_num(mileage='0.0396')
>>> mileage_num
0.0396
>>> mileage_num = mileage_str_to_num(mileage='')
>>> mileage_num
nan
```

mileage_num_to_str

pyrcs.converter.**mileage_num_to_str**(*mileage*)

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

Parameters
mileage (*float | None*) – The numerical-type mileage data.

Returns
The corresponding string-type mileage in the form of <*miles*>.<*yards*>.

Return type
str

Examples:

```
>>> from pyrcs.converter import mileage_num_to_str
>>> mileage_str = mileage_num_to_str(mileage=0.0396)
>>> mileage_str
'0.0396'
>>> mileage_str = mileage_num_to_str(mileage=None)
>>> mileage_str
''
```

shift_mileage_by_yard

pyrcs.converter.**shift_mileage_by_yard**(*mileage*, *shift_yards*, *as_numeric=True*)

Shifts the given mileage by a specified number of yards.

Parameters

- **mileage** (*float | int | str*) – The initial mileage (associated with an ELR).
- **shift_yards** (*int | float*) – The number of yards by which to shift the given mileage.
- **as_numeric** (*bool*) – If True, returns the result as a numeric type; defaults to True.

Returns

The mileage shifted by the specified number of yards.

Return type

float | str

Examples:

```
>>> from pyrcs.converter import shift_mileage_by_yard
>>> n_mileage = shift_mileage_by_yard(mileage='0.0396', shift_yards=220)
>>> n_mileage
0.0616
>>> n_mileage = shift_mileage_by_yard(mileage='0.0396', shift_yards=221)
>>> n_mileage
0.0617
>>> n_mileage = shift_mileage_by_yard(mileage=10, shift_yards=220)
>>> n_mileage
10.022
```

5.2.2 Convert other data

1anti-flashwhitewhite

fix_stanox(stanox)	Standardises the format of a given STANOX (station number).
kilometer_to_yard(km)	Converts a distance from kilometres to yards.

fix_stanox

pyrcs.converter.**fix_stanox**(*stanox*)

Standardises the format of a given STANOX (station number).

Parameters
stanox (*str | int | None*) – The STANOX code to be standardised.

Returns
The standardised STANOX as a string.

Return type
str

Examples:

```
>>> from pyrcs.converter import fix_stanox
>>> fixed_stanox = fix_stanox(stanox=65630)
>>> fixed_stanox
'65630'
>>> fixed_stanox = fix_stanox(stanox='2071')
>>> fixed_stanox
'02071'
>>> fixed_stanox = fix_stanox(stanox=2071)
>>> fixed_stanox
'02071'
```

kilometer_to_yard

pyrcs.converter.**kilometer_to_yard**(*km*)

Converts a distance from kilometres to yards.

Parameters
km (*int | float | None*) – The distance in kilometres to convert.

Returns
The equivalent distance in yards.

Return type
float

Examples:

```
>>> from pyrcs.converter import kilometer_to_yard
>>> kilometer_to_yard(1)
1093.6132983377079
```

5.3 collector

Collects data of railway codes.

Note

The current release only includes line data and other assets.

1anti-flashwhitewhite

LineData([update, verbose, raise_error])	A class representation of all modules of the subpackage *line_data* for collecting the codes of line data.
OtherAssets([update, verbose, raise_error])	A class representation of all modules of the subpackage *other_assets* for collecting the codes of other assets.

5.3.1 LineData

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

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

Parameters

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

Variables

- **connected** (*bool*) – Whether the Internet / the Railway Codes website is connected.
- **catalogue** (*dict*) – The catalogue of the line data.
- **ELRMileages** (ELRMileages) – An instance of the ELRMileages class.
- **Electrification** (Electrification) – An instance of the Electrification class.
- **LocationIdentifiers** (LocationIdentifiers) – An instance of the LocationIdentifiers class.
- **LOR** (LOR) – An instance of the LOR class.
- **LineNames** (LineNames) – An instance of the LineNames class.
- **TrackDiagrams** (TrackDiagrams) – An instance of the TrackDiagrams class.
- **Bridges** (Bridges) – An instance of the Bridges class.

Examples:

```
>>> from pyrcs import LineData
>>> ld = LineData()
>>> # To get data of location codes
>>> location_codes = ld.LocationIdentifiers.fetch_codes()
>>> type(location_codes)
dict
>>> list(location_codes.keys())
['LocationID', 'Other systems', 'Additional notes', 'Last updated date']
>>> location_codes_dat = location_codes[ld.LocationIdentifiers.KEY]
>>> type(location_codes_dat)
pandas.core.frame.DataFrame
>>> location_codes_dat.head()
                              Location CRS  ... STANME_Note STANOX_Note
0                                   Aachen      ...
1                      Abbeyhill Junction      ...
2                   Abbeyhill Signal E811      ...
3              Abbeyhill Turnback Sidings      ...
4  Abbey Level Crossing (Staffordshire)      ...
[5 rows x 12 columns]
>>> # To get data of line names
>>> line_names_codes = ld.LineNames.fetch_codes()
>>> type(line_names_codes)
dict
>>> list(line_names_codes.keys())
['Line names', 'Last updated date']
>>> line_names_codes_dat = line_names_codes[ld.LineNames.KEY]
>>> type(line_names_codes_dat)
pandas.core.frame.DataFrame
>>> line_names_codes_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]
```

Attributes

1anti-flashwhitewhite

NAME	The name of the data.

LineData.NAME

LineData.**NAME: str = 'Line data'**

The name of the data.

Methods

1anti-flashwhitewhite

update([confirmation_required, verbose, ...])	Updates the pre-packed line data.

LineData.update

LineData.**update**(*confirmation_required=True*, *verbose=False*, *interval=5*, *init_update=False*)

Updates the pre-packed line data.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required before proceeding; defaults to True.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **interval** (*int or float*) – A time gap (in seconds) between the updating of different classes, defaults to 5
- **init_update** (*bool*) – Whether to update the data for each subclass when being instantiated, defaults to False

Examples:

```
>>> from pyrcs.collector import LineData
>>> ld = LineData()
>>> ld.update(verbose=True)
```

5.3.2 OtherAssets

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

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

Parameters

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

Variables

- **connected** (*bool*) – Whether the Internet / the Railway Codes website is connected.
- **catalogue** (*dict*) – The catalogue of the data.
- **SignalBoxes** (SignalBoxes) – An instance of the SignalBoxes class.
- **Tunnels** (Tunnels) – An instance of the Tunnels class.
- **Viaducts** (Viaducts) – An instance of the Viaducts class.

- **Stations** (Stations) – An instance of the Stations class.
- **Depots** (Depots) – An instance of the Depots class.
- **Features** (Features) – An instance of the Features class.
- **HabdWild** (HabdWild) – An instance of the HabdWild class.
- **WaterTroughs** (WaterTroughs) – An instance of the WaterTroughs class.
- **Telegraph** (Telegraph) – An instance of the Telegraph class.
- **Buzzer** (Buzzer) – An instance of the Buzzer class.

Examples:

```
>>> from pyrcs import OtherAssets
>>> oa = OtherAssets()
>>> # To get data of railway stations
>>> rail_stn_locations = oa.Stations.fetch_locations()
>>> type(rail_stn_locations)
dict
>>> list(rail_stn_locations.keys())
['Mileages, operators and grid coordinates', 'Last updated date']
>>> rail_stn_locations_dat = rail_stn_locations[oa.Stations.KEY_TO_STN]
>>> type(rail_stn_locations_dat)
pandas.core.frame.DataFrame
>>> rail_stn_locations_dat.head()
           Station  ...                                    Former Operator
0       Abbey Wood  ...  London & South Eastern Railway from 1 April 20...
1       Abbey Wood  ...
2            Aber   ...  Keolis Amey Operations/Gweithrediadau Keolis A...
3        Abercynon  ...  Keolis Amey Operations/Gweithrediadau Keolis A...
4  Abercynon North  ...  [Cardiff Railway Company from 13 October 1996 ...
[5 rows x 13 columns]
>>> # To get data of signal boxes
>>> signal_boxes_codes = oa.SignalBoxes.fetch_prefix_codes()
>>> type(signal_boxes_codes)
dict
>>> list(signal_boxes_codes.keys())
['Signal boxes', 'Last updated date']
>>> signal_boxes_codes_dat = signal_boxes_codes[oa.SignalBoxes.KEY]
>>> type(signal_boxes_codes_dat)
pandas.core.frame.DataFrame
>>> signal_boxes_codes_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]
```

Attributes

1anti-flashwhitewhite

NAME	The name of the data.

OtherAssets.NAME

OtherAssets.**NAME: str = 'Other assets'**

The name of the data.

Methods

1anti-flashwhitewhite

update([confirmation_required, verbose, ...])	Updates the pre-packed data of the other assets.

OtherAssets.update

OtherAssets.**update**(*confirmation_required=True*, *verbose=False*, *interval=5*, *init_update=False*)

Updates the pre-packed data of the other assets.

Parameters

- **confirmation_required** (*bool*) – Whether user confirmation is required before proceeding; defaults to True.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **interval** (*int or float*) – A time gap (in seconds) between the updating of different classes, defaults to 5
- **init_update** (*bool*) – Whether to update the data for each subclass when being instantiated, defaults to False

Examples:

```
>>> from pyrcs.collector import OtherAssets
>>> oa = OtherAssets()
>>> oa.update(verbose=True)
```

5.4 utils

Provides a number of helper functions.

5.4.1 Validate inputs

1anti-flashwhitewhite

is_homepage_connectable()	Checks and returns whether the Railway Codes website is reacheable.
is_str_float(x[, finite_only])	Checks if a string represents and can be converted to a finite float value.
validate_initial(initial[, as_is])	Validates if a value is a single ASCII letter.
validate_page_name(cls_instance, page_no, ...)	Retrieves the valid page name corresponding to a given page number.
get_collect_verbosity_for_fetch(data_dir, ...)	Creates a new parameter that indicates whether to print relevant information to the console.
get_batch_fetch_verbosity(data_dir, verbose)	Creates a new parameter that indicates whether to print relevant information to the console.

is_homepage_connectable

pyrcs.utils.**is_homepage_connectable**()

Checks and returns whether the Railway Codes website is reacheable.

Returns
Whether the Railway Codes website is reacheable.

Return type
bool

Examples:

```
>>> from pyrcs.utils import is_homepage_connectable
>>> is_homepage_connectable()
True
```

is_str_float

pyrcs.utils.**is_str_float**(*x*, *finite_only=False*)

Checks if a string represents and can be converted to a finite float value.

This function attempts to convert the input to a float. It returns `False` for non-numeric strings, `None` inputs, or special floating point values like `NaN` or `Infinity` (optional).

Parameters

- **x** (*str | Any*) – String-type data or any object that can be converted to float.
- **finite_only** (*bool*) – Whether to return `False` for special values such as `NaN` and `Infinity`; defaults to `False`.

Returns
`True` if the string represents a valid finite number, `False` otherwise.

Return type
bool

Examples:

```
>>> from pyrcs.utils import is_str_float
>>> is_str_float('')
False
>>> is_str_float('a')
False
>>> is_str_float('1')
True
>>> is_str_float('1.1')
True
>>> is_str_float('nan', finite_only=True)
False
>>> is_str_float('inf')
True
```

validate_initial

pyrcs.utils.**validate_initial**(*initial*, *as_is=False*)

Validates if a value is a single ASCII letter.

Parameters

- **initial** (*str*) – The input value to validate, which is expected to be a string representing a single letter.
- **as_is** (*bool*) – If as_is=True, the function returns the letter in its original case; if as_is=False (default), the letter is returned in uppercase.

Returns

The validated initial letter.

Return type

str

Raises

- **ValueError** – If the input is not a single ASCII letter.
- **TypeError** – If the input is not a string.

Examples:

```
>>> from pyrcs.utils import validate_initial
>>> validate_initial('x')
'X'
>>> validate_initial('x', as_is=True)
'x'
>>> validate_initial('xyz')
Traceback (most recent call last):
  ...
    ...
ValueError: 'xyz' is invalid; it must be a single letter (A-Z, a-z).
```

validate_page_name

pyrcs.utils.**validate_page_name**(*cls_instance*, *page_no*, *valid_page_no*)

Retrieves the valid page name corresponding to a given page number.

This method checks if the provided `page_no` is within the set of valid page numbers. If valid, it returns the name of the page associated with the specified `page_no` from the catalogue of the given `cls_instance`.

Parameters

- **cls_instance** (*object*) – An instance of the class that contains the page catalogue.
- **page_no** (*int | str*) – The page number to validate, which can be an integer or a string representation of a number.
- **valid_page_no** (*set | list | tuple | range*) – A collection of valid page numbers, which can be a set, list, or tuple containing the allowable page numbers.

Returns

The validated page name associated with the given `page_no` in the class's catalogue.

Return type

str

See also

Examples for the methods:

- `Tunnels.collect_codes()`
- `Viaducts.collect_codes()`

get_collect_verbosity_for_fetch

pyrcs.utils.**get_collect_verbosity_for_fetch**(*data_dir*, *verbose*)

Creates a new parameter that indicates whether to print relevant information to the console.

This function is used only if it is necessary to re-collect data when trying to fetch the data.

Parameters

- **data_dir** (*str | None*) – The directory path where data is saved.
- **verbose** (*bool | int*) – The requested verbosity level.

Returns

The resolved verbosity level (boolean or integer).

Return type

bool | int

Examples:

```
>>> from pyrcs.utils import get_collect_verbosity_for_fetch
>>> get_collect_verbosity_for_fetch(data_dir="data", verbose=True)
False
```

get_batch_fetch_verbosity

pyrcs.utils.**get_batch_fetch_verbosity**(*data_dir*, *verbose*)

Creates a new parameter that indicates whether to print relevant information to the console.

This function is used only when fetching all data of a cluster.

Parameters

- **data_dir** (*str | None*) – Name of the folder where the pickle file is to be saved.
- **verbose** (*bool | int*) – Whether to print relevant information to the console.

Returns

A boolean indicating whether to print relevant information to the console when fetching all data of a cluster.

Return type

bool | int

Examples:

```
>>> from pyrcs.utils import get_batch_fetch_verbosity
>>> get_batch_fetch_verbosity(data_dir="data", verbose=True)
False
```

5.4.2 Print messages

1anti-flashwhitewhite

format_confirmation_prompt(data_name[, ...])	Returns a message for comfirming whether to proceed to collect a certain cluster of data.
print_collection_message(data_name[, ...])	Prints a message indicating the status of data collection.
print_connection_warning([verbose])	Checks the Internet connection and prints a warning if the source is unreachable.
print_instance_connection_error([update, ...])	Prints an error message when an instance fails to establish an Internet connection.
print_void_collection_message(data_name, verbose)	Prints a warning message when the data collection process fails and no fresh data was collected.

format_confirmation_prompt

pyrcs.utils.**format_confirmation_prompt**(*data_name*, *initial=None*, *ending='\n?'*)

Returns a message for comfirming whether to proceed to collect a certain cluster of data.

Parameters

- **data_name** (*str*) – The name of the dataset to be collected, e.g. `"Railway Codes"`.
- **initial** (*str | None*) – The initial letter for the code; defaults to `None`.
- **ending** (*str*) – The ending of the confirmation message; defaults to `"\n?"`.

Returns

A confirmation message asking whether to proceed with the dataset collection.

Return type

str

Examples:

```
>>> from pyrcs.utils import format_confirmation_prompt
>>> prompt = format_confirmation_prompt(data_name="Railway Codes")
>>> print(prompt)
To collect data of Railway Codes
?
>>> prompt = format_confirmation_prompt(data_name="location codes", initial="A")
>>> print(prompt)
To collect data of location codes beginning with "A"
?
```

print_collection_message

pyrcs.utils.**print_collection_message**(*data_name*, *initial=None*, *verbose=False*, *confirmation_required=True*, *end=' ... '*)

Prints a message indicating the status of data collection.

Parameters

- **data_name** (*str*) – The name of the data being collected.
- **initial** (*str | None*) – The initial letter of the desired code or data; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console.
- **confirmation_required** (*bool*) – Whether user confirmation is required before proceeding.
- **end** (*str*) – String appended at the end of the message; defaults to `" ... "`.

Examples:

```
>>> from pyrcs.utils import print_collection_message
>>> print_collection_message("Railway Codes", verbose=True, confirmation_required=False)
Collecting the data of Railway Codes ...
```

print_connection_warning

pyrcs.utils.**print_connection_warning**(*verbose=False*)

Checks the Internet connection and prints a warning if the source is unreachable.

This function attempts to connect to the source homepage of Railway Codes. If the connection fails and `verbose` is `True`, it alerts the user that the instance will rely on local backup data.

Parameters

verbose (*bool | int*) – Whether to print the warning message; defaults to `False`.

Examples:

```
>>> from pyrcs.utils import print_connection_warning
>>> # If the website is reachable, nothing is printed:
>>> print_connection_warning(verbose=True)

>>> # If the website is unreachable, a warning is printed:
>>> print_connection_warning(verbose=True)
Failed to establish an Internet connection. The current instance relies on local backup.
```

print_instance_connection_error

pyrcs.utils.**print_instance_connection_error**(*update=False*, *verbose=False*, *e=None*, *raise_error=False*)

Prints an error message when an instance fails to establish an Internet connection.

Parameters

- **update** (*bool*) – Indicates whether the error occurred during a data update; defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **e** (*Exception | None*) – An optional exception message to display.
- **raise_error** (*bool*) – Whether to raise the exception; if `raise_error=False` (default), the error will be suppressed.

Examples:

```
>>> from pyrcs.utils import print_instance_connection_error
>>> print_instance_connection_error(verbose=True)
The Internet connection is not available.
>>> print_instance_connection_error(update=True, verbose=True)
The Internet connection is not available. Failed to update the data.
>>> print_instance_connection_error(update=True, verbose=2, raise_error=True)
Failed. The Internet connection is not available. Failed to update the data.
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
  ...
    ...
TypeError: exceptions must derive from BaseException
```

print_void_collection_message

pyrcs.utils.**print_void_collection_message**(*data_name*, *verbose*)

Prints a warning message when the data collection process fails and no fresh data was collected.

Parameters

- **data_name** (*str*) – Name of the data being collected.
- **verbose** (*bool | int*) – Whether to print relevant information to the console.

Examples:

```
>>> from pyrcs.utils import print_void_collection_message
>>> print_void_collection_message(data_name="Railway Codes", verbose=True)
No data of "Railway Codes" has been freshly collected.
```

5.4.3 Save and retrieve pre-packed data

1anti-flashwhitewhite

fetch_location_names_errata([k, regex, ...])	Fetches a dictionary or dataframe to rectify location names.

fetch_location_names_errata

pyrcs.utils.**fetch_location_names_errata**(*k=None*, *regex=False*, *as_dataframe=False*, *column_name=None*)

Fetches a dictionary or dataframe to rectify location names.

Parameters

- **k** (*str | int | float | bool | None*) – The key for the errata dictionary; defaults to `None`.
- **regex** (*bool*) – Whether to create the dictionary for replacements based on regular expressions; defaults to `False`.
- **as_dataframe** (*bool*) – Whether to return the dictionary as a dataframe; defaults to `False`.
- **column_name** (*str | list | None*) – If `as_dataframe=True`, the column name for the dataframe; defaults to `None`.

Returns
A dictionary for rectifying location names, or a dataframe if requested.

Return type
dict | pandas.DataFrame

Examples:

```
>>> from pyrcs.utils import fetch_location_names_errata
>>> repl_dict = fetch_location_names_errata()
>>> type(repl_dict)
dict
>>> list(repl_dict.keys())[:5]
['"Tyndrum Upper" (Upper Tyndrum)',
 'AISH EMERGENCY CROSSOVER',
 'ATLBRJN',
 'Aberdeen Craiginches',
 'Aberdeen Craiginches T.C.']
>>> repl_dict = fetch_location_names_errata(regex=True, as_dataframe=True)
>>> type(repl_dict)
pandas.core.frame.DataFrame
>>> 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-
```

License

- PyRCS (since version 1.0.0) is licensed under the MIT License.
- Versions 0.3.7 and earlier are licensed under the GPLv3+ License.

Use of Data

For information on using the data pre-packaged with and collected by PyRCS, please refer to the Use/Contribute page.

Acknowledgement

PyRCS uses data available from the Railway Codes website. The time and effort that the website's editor and all contributors put in making the site and data available are fully credited.

Contributors

- Qian Fu
- Firtun

Python Module Index

p

Index

G

H

I

K

L

M

N

O

P

S

T

U

V

W

Y