

PyRCS

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

Release 0.3.3

Qian Fu

Birmingham Centre for Railway Research and Education

School of Engineering

University of Birmingham

First release: August 2019

Last updated: November 2022

© Copyright 2019-2022, Qian Fu

Table of Contents

1	About PyRCS	1
2	Installation	2
3	Sub-packages and modules	3
3.1	Sub-packages	3
3.1.1	line_data	3
	elr_mileage	4
	elec	14
	loc_id	28
	lor_code	39
	line_name	47
	trk_diagr	51
	bridge	55
3.1.2	other_assets	58
	sig_box	59
	tunnel	73
	viaduct	78
	station	82
	depot	89
	feature	101
3.2	Modules	114
3.2.1	parser	114
	Preprocess contents	114
	Extract information	117
3.2.2	converter	124
	Convert mileage data	124
	Convert other data	130
3.2.3	collector	132
	LineData	132
	OtherAssets	134
3.2.4	utils	137
	Validate inputs	137
	Print messages	140
	Save and retrieve pre-packed data	143

4	License	148
5	Use of data	149
6	Acknowledgement	150
7	Tutorial	151
7.1	Location identifiers	151
7.1.1	Location identifiers given a specific initial letter	152
7.1.2	All available location identifiers	153
7.2	ELRs and mileages	154
7.2.1	Engineer's Line References (ELRs)	154
7.2.2	Mileage file of a given ELR	156
7.3	Railway station data	157
7.3.1	Railway station locations given a specific initial letter	158
7.3.2	All available railway station locations	159
	Python Module Index	161
	Index	162

Chapter 1

About PyRCS

PyRCS is an open-source Python package for collecting and handling various codes (used in different UK rail industry systems), which are made available from [Railway Codes](#) website. This tool is intended for those, such as researchers and practitioners, who are the website users or work with the UK's railway codes by using Python programming language. It can facilitate access to, and manipulation of, the relevant data.

The *installation* of PyRCS includes a set of pre-packed data. When users request data of a category that is specified on the [Railway Codes](#) website, the pre-packed data of the category is loaded by default. Beyond that, it also offers capabilities to directly access the most up-to-date data on the data source website, and update the relevant pre-packed data as well.

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: 0.3.3

Chapter 3

Sub-packages and modules

3.1 Sub-packages

<i>line_data</i>	A sub-package for collecting codes of <i>line data</i> .
<i>other_assets</i>	A sub-package of modules for collecting codes of <i>other assets</i> .

3.1.1 line_data

A sub-package for collecting codes of *line data*.

(See also *LineData*.)

Sub-modules

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

elr_mileage

Collect [Engineer's Line References \(ELRs\)](#).

Class

<code>ELRMileages</code> ([data_dir, update, verbose])	A class for collecting data of Engineer's Line References (ELRs) .
--	--

ELRMileages

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

A class for collecting data of [Engineer's Line References \(ELRs\)](#).

Parameters

- **data_dir** (*str* or *None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to True

Variables

- **catalogue** (*dict*) – catalogue of the data
- **last_updated_date** (*str*) – last update date
- **data_dir** (*str*) – path to the data directory
- **current_data_dir** (*str*) – path to the current data directory

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

<i>KEY</i>	Key of the <code>dict</code> -type data
<i>KEY_TO_LAST_UPDATED_DATE</i>	Key of the data of the last updated date
<i>NAME</i>	Name of the data
<i>URL</i>	URL of the main web page of the data

ELRMileages.KEY

`ELRMileages.KEY = 'ELRs and mileages'`

Key of the `dict`-type data

ELRMileages.KEY_TO_LAST_UPDATED_DATE

`ELRMileages.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'`

Key of the data of the last updated date

ELRMileages.NAME

`ELRMileages.NAME = "Engineer's Line References (ELRs)"`

Name of the data

ELRMileages.URL

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

URL of the main web page of the data

Methods

<i>collect_elr_by_initial</i> (initial[, update, ...])	Collect Engineer's Line References (ELRs) for a given initial letter from source web page.
<i>collect_mileage_file</i> (elr[, parsed, ...])	Collect mileage file for the given ELR from source web page.
<i>fetch_elr</i> ([update, dump_dir, verbose])	Fetch data of ELRs and their associated mileages.
<i>fetch_mileage_file</i> (elr[, update, dump_dir, ...])	Fetch the mileage file for a given ELR.
<i>get_conn_mileages</i> (start_elr, end_elr[, update])	Get a connection point between two ELR-and-mileage pairs.
<i>search_conn</i> (start_elr, start_em, end_elr, end_em)	Search for connection between two ELR-and-mileage pairs.

ELRMileages.collect_elr_by_initial

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

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

Parameters

- **initial** (*str*) – initial letter of an ELR, e.g. 'a', 'z'
- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to True

Returns

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

Return type

dict

Examples:

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

>>> em = ELRMileages()

>>> elrs_a_codes = em.collect_elr_by_initial(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_by_initial(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_it=False, verbose=False)`

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

Parameters

- **elr** (*str*) – ELR, e.g. 'CJD', 'MLA', 'FED'
- **parsed** (*bool*) – whether to parse the scraped mileage data
- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **dump_it** (*bool*) – whether to save the collected data as a pickle file, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

mileage file for the given *elr*

Return type

dict

Note:

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

Examples:

```
>>> from pyrcs.line_data import ELRMileages # from pyrcs import ELRMileages
>>> em = ELRMileages()
>>> cjd_mileage_file = em.collect_mileage_file(elr='CJD')
To collect mileage file of "CJD"
? [No]|Yes: yes
>>> type(cjd_mileage_file)
dict
>>> list(cjd_mileage_file.keys())
['ELR', 'Line', 'Sub-Line', 'Mileage', 'Notes']
>>> cjd_mileage_file['Mileage']
Mileage ... Link_1_Mile_Chain
0    0.0000 ...
1    0.0528 ...          91.48
2    1.1540 ...
```

(continues on next page)

(continued from previous page)

```

3   2.0000 ...
4   2.1562 ...           0.00
5   6.0022 ...
6   8.0308 ...
7  10.0748 ...
8  12.0968 ...
9  14.0968 ...
10 16.1452 ...
11 19.1408 ...
12 19.1540 ...
13 23.0770 ...
14 26.1078 ...
15 28.1276 ...
16 32.1188 ...
17 32.1188 ...
18 38.1276 ...
19 43.0572 ...
20 46.0704 ...
21 49.1188 ...
22 49.1320 ...           0.00
23 55.1606 ...
24 64.0594 ...
[25 rows x 8 columns]

>>> gam_mileage_file = em.collect_mileage_file(elr='GAM')
To collect mileage file of "GAM"
? [No]|Yes: yes
>>> 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]

>>> sld_mileage_file = em.collect_mileage_file(elr='SLD')
To collect mileage file of "SLD"
? [No]|Yes: yes
>>> sld_mileage_file['Mileage']
Mileage Mileage_Note Miles_Chains ... Link_1 Link_1_ELR Link_1_Mile_Chain
0  31.0088                31.04 ...   MVN2       MVN2
1  31.0682                31.31 ...   None
2  31.1474                31.67 ...   None
3  32.1078                32.49 ...   None
4  32.1232                32.56 ...   None
[5 rows x 8 columns]

>>> elr_mileage_file = em.collect_mileage_file(elr='ELR')
To collect mileage file of "ELR"
? [No]|Yes: yes
>>> elr_mileage_file['Mileage']
Mileage Mileage_Note ... Link_1_ELR Link_1_Mile_Chain
0  122.0044                ...   GRS3
1  122.0682                ...           0.00
2  122.0726                ...   SPI       0.00
3  122.0836                ...
4  124.0792                ...
5  127.1716 Approximate ...
6  128.0088                ...

```

(continues on next page)

(continued from previous page)

```

7  128.0154      ...      MAB
8  130.0946      ...
9  133.1254      ...
10 134.1694      ...
11 138.0770      ...
12 139.1694      ...      MAB      149.43
13 140.1122      ...      LOB      150.13
14 141.0000      ...
15 143.0792      ...
16 143.0792      ...
17 145.1078      ...
18 146.0594      ...
19 147.1650      ...
20 148.1166      ...
21 149.1452      ...
22 150.1056      ...
23 151.1606      ...
24 154.0088      ...
25 154.0704      ...
26 154.1078      ...
27 154.1628      ...
28 154.1650      ...      MAC3      109.53
[29 rows x 8 columns]
```

ELRMileages.fetch_elr

`ELRMileages.fetch_elr(update=False, dump_dir=None, verbose=False)`

Fetch data of ELRs and their associated mileages.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

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

Return type

dict

Examples:

```

>>> from pyrcs.line_data import ELRMileages # from pyrcs import ELRMileages

>>> em = ELRMileages()

>>> elrs_codes = em.fetch_elr()
>>> type(elrs_codes)
```

(continues on next page)

(continued from previous page)

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

>>> em.KEY
'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)`

Fetch the mileage file for a given ELR.

Parameters

- **elr** (*str*) – elr: ELR, e.g. 'CJD', 'MLA', 'FED'
- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

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

Return type

dict

Examples:

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

>>> em = ELRMileages()

>>> # Get the mileage file of 'AAL' (Now 'NAJ3')
>>> aal_mileage_file = em.fetch_mileage_file(elr='AAL')
>>> 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')
>>> 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]

```

ELRMileages.get_conn_mileages

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

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

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

Note: This function may not be able to find the connection for every pair of ELRs. See [Example 2](#) below.

Parameters

- `start_elr (str)` – start ELR
- `end_elr (str)` – end ELR
- `update (bool)` – whether to do an update check (for the package data), defaults to `False`
- `kwargs` – [optional] parameters of the method
`ELRMileages.fetch_mileage_file()`

Returns

connection ELR and mileages between the given `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()
```

(continues on next page)

(continued from previous page)

```
>>> 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)

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

Parameters

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

Returns

connection (<end mileage of the start ELR>, <start mileage of the end 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
```

(continues on next page)

(continued from previous page)

```
'0.0396'
>>> elr_2_orig
'84.1364'
```

elec

Collect [section codes for overhead line electrification \(OLE\) installations](#).

Class

<i>Electrification</i> ([data_dir, update, verbose])	A class for collecting section codes for overhead line electrification (OLE) installations .
--	--

Electrification

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

A class for collecting [section codes for overhead line electrification \(OLE\) installations](#).

Parameters

- **data_dir** (*str* or *None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to True

Variables

- **catalogue** (*dict*) – catalogue of the data
- **last_updated_date** (*str*) – last update date
- **data_dir** (*str*) – path to the data directory
- **current_data_dir** (*str*) – 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

<i>KEY</i>	Key of the <i>dict</i> -type data
<i>KEY_TO_ENERGY_TARIFF_ZONES</i>	Key of the dict-type data of the ' <i>UK railway electrification tariff zones</i> '
<i>KEY_TO_INDEPENDENT_LINES</i>	Key of the dict-type data of the ' <i>independent lines</i> '
<i>KEY_TO_LAST_UPDATED_DATE</i>	Key of the data of the last updated date
<i>KEY_TO_NATIONAL_NETWORK</i>	Key of the dict-type data of the ' <i>national network</i> '
<i>KEY_TO_OHNS</i>	Key of the dict-type data of the ' <i>overhead line electrification neutral sections (OHNS)</i> '
<i>NAME</i>	Name of the data
<i>URL</i>	URL of the main web page of the data

Electrification.KEY

`Electrification.KEY = 'Electrification'`

Key of the *dict*-type data

Electrification.KEY_TO_ENERGY_TARIFF_ZONES

`Electrification.KEY_TO_ENERGY_TARIFF_ZONES = 'National network energy tariff zones'`

Key of the dict-type data of the '*UK railway electrification tariff zones*'

Electrification.KEY_TO_INDEPENDENT_LINES

`Electrification.KEY_TO_INDEPENDENT_LINES = 'Independent lines'`

Key of the dict-type data of the '*independent lines*'

Electrification.KEY_TO_LAST_UPDATED_DATE

`Electrification.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'`

Key of the data of the last updated date

Electrification.KEY_TO_NATIONAL_NETWORK

```
Electrification.KEY_TO_NATIONAL_NETWORK = 'National network'
```

Key of the dict-type data of the *'national network'*

Electrification.KEY_TO_OHNS

```
Electrification.KEY_TO_OHNS = 'National network neutral sections'
```

Key of the dict-type data of the *'overhead line electrification neutral sections (OHNS)'*

Electrification.NAME

```
Electrification.NAME = 'Section codes for overhead line electrification (OLE) installations'
```

Name of the data

Electrification.URL

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

URL of the main web page of the data

Methods

<code>collect_etz_codes</code> ([confirmation_required, ...])	Collect OLE section codes for national network energy tariff zones from source web page.
<code>collect_indep_lines_codes</code> ([...])	Collect OLE section codes for independent lines from source web page.
<code>collect_national_network_codes</code> ([...])	Collect OLE section codes for national network from source web page.
<code>collect_ohns_codes</code> ([confirmation_required, ...])	Collect codes for overhead line electrification neutral sections (OHNS) from source web page.
<code>fetch_codes</code> ([update, dump_dir, verbose])	Fetch OLE section codes listed in the Electrification catalogue.
<code>fetch_etz_codes</code> ([update, dump_dir, verbose])	Fetch OLE section codes for national network energy tariff zones .
<code>fetch_indep_lines_codes</code> ([update, dump_dir, ...])	Fetch OLE section codes for independent lines .
<code>fetch_national_network_codes</code> ([update, ...])	Fetch OLE section codes for national network .
<code>fetch_ohns_codes</code> ([update, dump_dir, verbose])	Fetch codes for overhead line electrification neutral sections (OHNS).
<code>get_indep_line_catalogue</code> ([update, verbose])	Get a catalogue for independent lines .

Electrification.collect_etz_codes

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

Collect OLE section codes for [national network energy tariff zones](#) from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

OLE section codes for national network energy tariff zones

Return type

dict or None

Examples:

```
>>> from pyrcs.line_data import Electrification # from pyrcs import Electrification
>>> elec = Electrification()
```

(continues on next page)

(continued from previous page)

```

>>> rail_etz_codes = elec.collect_etz_codes()
To collect section codes for OLE installations: national network energy tariff zones
? [No]|Yes: yes
>>> type(rail_etz_codes)
dict
>>> list(rail_etz_codes.keys())
['National network energy tariff zones', 'Last updated date']

>>> elec.KEY_TO_ENERGY_TARIFF_ZONES
'National network energy tariff zones'

>>> rail_etz_codes_dat = rail_etz_codes[elec.KEY_TO_ENERGY_TARIFF_ZONES]
>>> 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_indep_lines_codes

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

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

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

OLE section codes for independent lines

Return type

dict or None

Examples:

```

>>> from pyrcs.line_data import Electrification # from pyrcs import Electrification

>>> elec = Electrification()

>>> indep_lines_codes = elec.collect_indep_lines_codes()
To collect section codes for OLE installations: independent lines
? [No]|Yes: yes
>>> 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)
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.collect_national_network_codes

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

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

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

OLE section codes for National network

Return type

dict or None

Examples:

```
>>> from pyrcs.line_data import Electrification # from pyrcs import Electrification
>>> elec = Electrification()
>>> nn_codes = elec.collect_national_network_codes()
To collect section codes for OLE installations: national network
? [No]|Yes: yes
>>> 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
```

(continues on next page)

(continued from previous page)

```

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*)

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

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

OHNS codes

Return type

dict or None

Examples:

```

>>> from pyrcs.line_data import Electrification # from pyrcs import Electrification

>>> elec = Electrification()

>>> ohl_ns_codes = elec.collect_ohns_codes()
To collect section codes for OLE installations: national network neutral sections
? [No]|Yes: yes
>>> 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']
  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

```

(continues on next page)

(continued from previous page)

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*)

Fetch OLE section codes listed in the [Electrification](#) catalogue.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str or None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

section codes for overhead line electrification (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)

Fetch OLE section codes for national network energy tariff zones.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

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_ENERGY_TARIFF_ZONES
'National network energy tariff zones'

>>> rail_etz_codes_dat = rail_etz_codes[elec.KEY_TO_ENERGY_TARIFF_ZONES]
>>> 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
```

(continues on next page)

(continued from previous page)

12	WA	West Anglia †
13	WC	West Coast/North West

Electrification.fetch_indep_lines_codes

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

Fetch OLE section codes for **independent lines**.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str or None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

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_indep_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)
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]',
```

(continues on next page)

(continued from previous page)

```

'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*)

Fetch OLE section codes for [national network](#).

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str or None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

OLE section codes for National network

Return type

dict or `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']

```

(continues on next page)

(continued from previous page)

```

>>> 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*)

Fetch codes for **overhead line electrification neutral sections** (OHNS).

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str or None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

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_indep_line_catalogue

Electrification.get_indep_line_catalogue(update=False, verbose=False)

Get a catalogue for independent lines.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

a list of independent line names

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_indep_line_catalogue()

```

(continues on next page)

(continued from previous page)

```
>>> indep_line_cat.head()
      Feature ...
0      Beamish Tramway ...
1      Birkenhead Tramway ...
2      Black Country Living Museum ...
3      Blackpool Tramway ...
4  Brighton and Rottingdean Seashore Electric Rai... ...
[5 rows x 3 columns]
```

loc_id

Collect CRS, NLC, TIPLOC and STANOX codes.

Class

<code>LocationIdentifiers</code> ([<code>data_dir</code> , <code>update</code> , <code>verbose</code>])	A class for collecting data of location identifiers (including other systems' station codes).
--	--

LocationIdentifiers

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

A class for collecting data of [location identifiers](#) (including [other systems'](#) station codes).

Parameters

- `data_dir` (*str* or *None*) – name of data directory, defaults to *None*
- `update` (*bool*) – whether to do an update check (for the package data), defaults to *False*
- `verbose` (*bool* or *int*) – whether to print relevant information in console, defaults to *True*

Variables

- `catalogue` (*dict*) – catalogue of the data
- `last_updated_date` (*str*) – last updated date
- `data_dir` (*str*) – path to the data directory
- `current_data_dir` (*str*) – path to the current data directory

Examples:

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

>>> lid = LocationIdentifiers()
```

(continues on next page)

(continued from previous page)

```
>>> lid.NAME
'CRS, NLC, TIPLoc and STANOX codes'

>>> lid.URL
'http://www.railwaycodes.org.uk/crs/crs0.shtm'
```

Attributes

<i>KEY</i>	Key of the dict-type data
<i>KEY_TO_ADDITIONAL_NOTES</i>	Key of the dict-type data of <i>additional notes</i>
<i>KEY_TO_LAST_UPDATED_DATE</i>	Key of the data of the last updated date
<i>KEY_TO_MSCEN</i>	Key of the dict-type data of the ' <i>multiple station codes explanatory note</i> '
<i>KEY_TO_OTHER_SYSTEMS</i>	Key of the dict-type data of the ' <i>other systems</i> '
<i>NAME</i>	Name of the data
<i>URL</i>	URL of the main web page of the data

LocationIdentifiers.KEY

LocationIdentifiers.KEY = 'LocationID'
Key of the dict-type data

LocationIdentifiers.KEY_TO_ADDITIONAL_NOTES

LocationIdentifiers.KEY_TO_ADDITIONAL_NOTES = 'Additional notes'
Key of the dict-type data of *additional notes*

LocationIdentifiers.KEY_TO_LAST_UPDATED_DATE

LocationIdentifiers.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'
Key of the data of the last updated date

LocationIdentifiers.KEY_TO_MSCEN

LocationIdentifiers.KEY_TO_MSCEN = 'Multiple station codes explanatory note'
Key of the dict-type data of the '*multiple station codes explanatory note*'

LocationIdentifiers.KEY_TO_OTHER_SYSTEMS

`LocationIdentifiers.KEY_TO_OTHER_SYSTEMS = 'Other systems'`

Key of the dict-type data of the *'other systems'*

LocationIdentifiers.NAME

`LocationIdentifiers.NAME = 'CRS, NLC, TIPLOC and STANOX codes'`

Name of the data

LocationIdentifiers.URL

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

URL of the main web page of the data

Methods

<code>cleanse_mult_alt_codes(data)</code>	Cleanse multiple alternatives for every code column.
<code>collect_codes_by_initial(initial[, update, ...])</code>	Collect CRS, NLC, TIPLOC, STANME and STANOX codes for a given initial letter.
<code>collect_explanatory_note([...])</code>	Collect note about CRS code from source web page.
<code>collect_other_systems_codes([...])</code>	Collect data of other systems' station codes from source web page.
<code>fetch_codes([update, dump_dir, verbose])</code>	Fetch CRS, NLC, TIPLOC, STANME and STANOX codes and other systems' station codes.
<code>fetch_explanatory_note([update, dump_dir, ...])</code>	Fetch multiple station codes explanatory note.
<code>fetch_other_systems_codes([update, ...])</code>	Fetch data of other systems' station codes.
<code>get_code_notes(data)</code>	Get notes for every code column.
<code>make_xref_dict(keys[, initials, main_key, ...])</code>	Make a dict/dataframe for location code data for the given keys.
<code>parse_location_name(data)</code>	Parse the location names of the preprocessed data.
<code>parse_stanox_note(data)</code>	Parse the note for STANOX.

LocationIdentifiers.cleanse_mult_alt_codes

static LocationIdentifiers.cleanse_mult_alt_codes(*data*)

Cleanse multiple alternatives for every code column.

Parameters

data (*pandas.DataFrame*) – preprocessed data of the location codes

Returns

cleansed data of the location codes where multiple alternatives are replicated

Return type

pandas.DataFrame

LocationIdentifiers.collect_codes_by_initial

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

Collect [CRS](#), [NLC](#), [TIPLOC](#), [STANME](#) and [STANOX](#) codes for a given initial letter.

Parameters

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

Returns

data of locations beginning with the given initial letter 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_a_codes = lid.collect_codes_by_initial(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()

```

(continues on next page)

(continued from previous page)

```

                                Location CRS ... STANME_Note STANOX_Note
0                                A1 ...
1                A463 Traded In ...
2  A483 Road Scheme Supervisors Closed ...
3                                Aachen ...
4                AA Holidays S524 ...
[5 rows x 12 columns]

```

LocationIdentifiers.collect_explanatory_note

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

Collect note about CRS code from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to False

Returns

data of multiple station codes explanatory note

Return type

dict or None

Examples:

```

>>> from pyrcs.line_data import LocationIdentifiers
>>> # from pyrcs import LocationIdentifiers

>>> lid = LocationIdentifiers()

>>> exp_note = lid.collect_explanatory_note()
To collect data of Multiple station codes explanatory note
? [No]|Yes: yes
>>> 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

```

(continues on next page)

(continued from previous page)

2	Heworth	HEW	HEZ	
3	Highbury & Islington	HHY	HII	XHZ
4	Lichfield Trent Valley	LTV	LIF	

LocationIdentifiers.collect_other_systems_codes

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

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

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to False

Returns

codes of other systems

Return type

dict or 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)
collections.defaultdict
>>> list(os_codes_dat.keys())
['C  ras Iompair   ireann (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)`

Fetch CRS, NLC, TIPLOC, STANME and STANOX codes and other systems' station codes.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str or None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

data of 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())
['LocationID', '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		A1		...	
1		A463 Traded In		...	
2	A483 Road Scheme Supervisors Closed			...	
3		Aachen		...	
4		AA Holidays S524		...	

```
[5 rows x 12 columns]
```

LocationIdentifiers.fetch_explanatory_note

`LocationIdentifiers.fetch_explanatory_note(update=False, dump_dir=None, verbose=False)`

Fetch multiple station codes explanatory note.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str or None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

data of 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_explanatory_note()
>>> 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*)

Fetch data of *other systems'* station codes.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to *False*
- **dump_dir** (*str* or *None*) – pathname of a directory where the data file is dumped, defaults to *None*
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to *False*

Returns

codes of 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())
['C  ras Iompair   ireann (Republic of Ireland)',
 'Crossrail',
 'Croydon Tramlink',
 'Docklands Light Railway',
 'Manchester Metrolink',
 'Translink (Northern Ireland)',
 'Tyne & Wear Metro']
```

LocationIdentifiers.get_code_notes

LocationIdentifiers.get_code_notes(*data*)

Get notes for every code column.

Parameters

data (*pandas.DataFrame*) – preprocessed data of the location codes

LocationIdentifiers.make_xref_dict

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

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

Parameters

- **keys** (*str or list*) – one or a sublist of ['CRS', 'NLC', 'TIPLOC', 'STANOX', 'STANME']
- **initials** (*str or list or None*) – one or a sequence of initials for which the codes are used, defaults to None
- **main_key** (*str or None*) – key of the returned dictionary (when *as_dict=True*), defaults to None
- **as_dict** (*bool*) – whether to return a dictionary, defaults to False
- **drop_duplicates** (*bool*) – whether to drop duplicates, defaults to False
- **dump_it** (*bool*) – whether to save the location codes dictionary, defaults to False
- **dump_dir** (*str or None*) – pathname of a directory where the data file is dumped, defaults to None
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict or *pandas.DataFrame* or None

Examples:

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

>>> lid = LocationIdentifiers()

>>> stanox_dictionary = lid.make_xref_dict(keys='STANOX')
>>> type(stanox_dictionary)
```

(continues on next page)

(continued from previous page)

```

pandas.core.frame.DataFrame
>>> stanox_dictionary.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', ''),
 ('23490', 'BABWTHL'),
 ('38306', 'BACHE'),
 ('66021', 'BADESCL'),
 ('81003', 'BADMTN')]

```

LocationIdentifiers.parse_location_name

LocationIdentifiers.parse_location_name(*data*)

Parse the location names of the preprocessed data.

Parameters

data (*pandas.DataFrame*) – preprocessed data of the location codes

LocationIdentifiers.parse_stanox_note

LocationIdentifiers.parse_stanox_note(*data*)

Parse the note for STANOX.

Parameters

data (*pandas.DataFrame*) – preprocessed data of the location codes

lor_code

Collect [Line of Route \(LOR/PRIDE\)](#) codes.

Class

LOR ([<i>data_dir</i> , <i>update</i> , <i>verbose</i>])	A class for collecting data of Line of Route (LOR/PRIDE) .
--	--

LOR

class pyrcs.line_data.lor_code.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* or *None*) – name of data directory, defaults to *None*
- **update** (*bool*) – whether to do an update check (for the package data), defaults to *False*
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to *True*

Variables

- **catalogue** (*dict*) – catalogue of the data
- **last_updated_date** (*str*) – last updated date
- **data_dir** (*str*) – path to the data directory
- **current_data_dir** (*str*) – path to the current data directory
- **valid_prefixes** (*list*) – 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

<i>KEY</i>	Key of the dict-type data
<i>KEY_ELC</i>	Key of the dict-type data of <i>ELR/LOR converter</i>
<i>KEY_P</i>	Key of the dict-type data of prefixes
<i>KEY_TO_LAST_UPDATED_DATE</i>	Key of the data of the last updated date
<i>NAME</i>	Name of the data
<i>SHORT_NAME</i>	Short name of the data
<i>URL</i>	URL of the main web page of the data

LOR.KEY

`LOR.KEY = 'LOR'`

Key of the dict-type data

LOR.KEY_ELC

`LOR.KEY_ELC = 'ELR/LOR converter'`

Key of the dict-type data of *ELR/LOR converter*

LOR.KEY_P

`LOR.KEY_P = 'Key to prefixes'`

Key of the dict-type data of prefixes

LOR.KEY_TO_LAST_UPDATED_DATE

```
LOR.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'
```

Key of the data of the last updated date

LOR.NAME

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

Name of the data

LOR.SHORT_NAME

```
LOR.SHORT_NAME = 'Line of Route (LOR/PRIDE) codes'
```

Short name of the data

LOR.URL

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

URL of the main web page of the data

Methods

<code>collect_codes_by_prefix(prefix[, update, ...])</code>	Collect PRIDE/LOR codes by a given prefix.
<code>collect_elr_lor_converter([...])</code>	Collect ELR/LOR converter from source web page.
<code>fetch_codes([update, dump_dir, verbose])</code>	Fetch data of PRIDE/LOR codes .
<code>fetch_elr_lor_converter([update, dump_dir, ...])</code>	Fetch data of ELR/LOR converter .
<code>get_keys_to_prefixes([prefixes_only, ...])</code>	Get the keys to PRIDE/LOR code prefixes.
<code>get_page_urls([update, verbose])</code>	Get URLs to PRIDE/LOR codes with different prefixes.

LOR.collect_codes_by_prefix

```
LOR.collect_codes_by_prefix(prefix, update=False, verbose=False)
```

Collect [PRIDE/LOR codes](#) by a given prefix.

Parameters

- **prefix** (*str*) – prefix of LOR codes

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

LOR codes for the given prefix

Return type

dict or `None`

Examples:

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

>>> lor = LOR()

>>> lor_codes_cy = lor.collect_codes_by_prefix(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_by_prefix(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_by_prefix(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()
```

(continues on next page)

(continued from previous page)

```

    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)`

Collect [ELR/LOR converter](#) from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

data of ELR/LOR converter

Return type

dict or 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/prideIn.s...
3  ABE1 ... http://www.railwaycodes.org.uk/pride/prideIn.s...
4  ABE2 ... http://www.railwaycodes.org.uk/pride/prideIn.s...
[5 rows x 6 columns]

```

LOR.fetch_codes

`LOR.fetch_codes(update=False, dump_dir=None, verbose=False)`

Fetch data of PRIDE/LOR codes.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

LOR codes

Return type

dict

Examples:

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

>>> lor = LOR()

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

>>> cy_codes = l_codes['CY']
>>> type(cy_codes)
dict
>>> list(cy_codes.keys())
['CY', 'Notes', 'Last updated date']
>>> cy_codes['CY']
  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)
dict
>>> list(xr_codes.keys())
['XR', 'Last updated date']
>>> xr_codes_ = xr_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'].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.fetch_elr_lor_converter

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

Fetch data of ELR/LOR converter.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **dump_dir** (*str* or *None*) – pathname of a directory where the data file is dumped, defaults to None
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to False

Returns

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_lor_conv_data = elr_lor_conv['ELR/LOR converter']
>>> type(elr_lor_conv_data)
pandas.core.frame.DataFrame
>>> elr_lor_conv_data.head()
   ELR  ... LOR_URL
0  AAV  ... http://www.railwaycodes.org.uk/pride/pridesw.s...
```

(continues on next page)

(continued from previous page)

```

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, verbose=False)`

Get the keys to PRIDE/LOR code prefixes.

Parameters

- **prefixes_only** (*bool*) – whether to get only prefixes, defaults to `True`
- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `True`

Returns

keys to LOR code prefixes

Return type

list or dict or `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, verbose=False)`

Get URLs to [PRIDE/LOR codes](#) with different prefixes.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `True`

Returns

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

Return type

list or `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'
```

line_name

Collect [railway line names](#).

Class

<code>LineNames</code> (<code>[data_dir, update, verbose]</code>)	A class for collecting data of railway line names .
---	---

LineNames

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

A class for collecting data of [railway line names](#).

Parameters

- **data_dir** (*str or None*) – name of data directory, defaults to `None`
- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`

- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to True

Variables

- **catalogue** (*dict*) – catalogue of the data
- **last_updated_date** (*str*) – last update date
- **data_dir** (*str*) – path to the data directory
- **current_data_dir** (*str*) – 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

<i>KEY</i>	Key of the <i>dict</i> -type data
<i>KEY_TO_LAST_UPDATED_DATE</i>	Key of the data of the last updated date
<i>NAME</i>	Name of the data
<i>URL</i>	URL of the main web page of the data

LineNames.KEY

```
LineNames.KEY = 'Line names'
```

Key of the *dict*-type data

LineNames.KEY_TO_LAST_UPDATED_DATE

```
LineNames.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'
```

Key of the data of the last updated date

LineNames.NAME

LineNames.NAME = 'Railway line names'

Name of the data

LineNames.URL

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

URL of the main web page of the data

Methods

<code>collect_codes</code> ([confirmation_required, verbose])	Collect data of railway line names from source web page.
<code>fetch_codes</code> ([update, dump_dir, verbose])	Fetch data of railway line names .

LineNames.collect_codes

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

Collect data of [railway line names](#) from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict or 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())
```

(continues on next page)

(continued from previous page)

```
['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]
```

LineNames.fetch_codes

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

Fetch data of [railway line names](#).

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

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

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
```

(continues on next page)

(continued from previous page)

```
'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]
```

trk_diagr

Collect British railway track diagrams.

Class

<i>TrackDiagrams</i> ([data_dir, update, verbose])	A class for collecting data of British railway track diagrams.
--	--

TrackDiagrams

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

A class for collecting data of British railway track diagrams.

Parameters

- **data_dir** (*str* or *None*) – name of data directory, defaults to *None*
- **update** (*bool*) – whether to do an update check (for the package data), defaults to *False*
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to *True*

Variables

- **catalogue** (*dict*) – catalogue of the data
- **last_updated_date** (*str*) – last updated date
- **data_dir** (*str*) – path to the data directory
- **current_data_dir** (*str*) – 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

<i>KEY</i>	Key of the <i>dict</i> -type data
<i>KEY_TO_LAST_UPDATED_DATE</i>	Key of the data of the last updated date
<i>NAME</i>	Name of the data
<i>URL</i>	URL of the main web page of the data

TrackDiagrams.KEY

TrackDiagrams.KEY = 'Track diagrams'

Key of the *dict*-type data

TrackDiagrams.KEY_TO_LAST_UPDATED_DATE

TrackDiagrams.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'

Key of the data of the last updated date

TrackDiagrams.NAME

TrackDiagrams.NAME = 'Railway track diagrams'

Name of the data

TrackDiagrams.URL

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

URL of the main web page of the data

Methods

<code>collect_catalogue</code> ([confirmation_required, ...])	Collect catalogue of sample railway track diagrams from source web page.
<code>fetch_catalogue</code> ([update, dump_dir, verbose])	Fetch the catalogue of railway track diagrams.

TrackDiagrams.collect_catalogue

`TrackDiagrams.collect_catalogue(confirmation_required=True, verbose=False)`

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

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict or 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
>>> 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...

TrackDiagrams.fetch_catalogue

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

Fetch the catalogue of railway track diagrams.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str or None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

catalogue of sample railway track diagrams and date of when the catalogue 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()
```

	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...

bridge

Collect data of British railway bridges.

Class

<code>Bridges([data_dir, verbose])</code>	A class for collecting data of railway bridges.
---	---

Bridges

class `pyrcs.line_data.bridge.Bridges(data_dir=None, verbose=True)`

A class for collecting data of railway bridges.

Parameters

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

Variables

- `catalogue` (*dict*) – catalogue of the data
- `last_updated_date` (*str*) – last update date
- `data_dir` (*str*) – path to the data directory
- `current_data_dir` (*str*) – 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

<code>KEY</code>	Key of the <i>dict</i> -type data
<code>KEY_TO_LAST_UPDATED_DATE</code>	Key of the data of the last updated date
<code>NAME</code>	Name of the data
<code>URL</code>	URL of the main web page of the data

Bridges.KEY

`Bridges.KEY = 'Bridges'`
Key of the `dict`-type data

Bridges.KEY_TO_LAST_UPDATED_DATE

`Bridges.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'`
Key of the data of the last updated date

Bridges.NAME

`Bridges.NAME = 'Railway bridges'`
Name of the data

Bridges.URL

`Bridges.URL = 'http://www.railwaycodes.org.uk/bridges/bridges0.shtm'`
URL of the main web page of the data

Methods

<code>collect_codes</code> ([confirmation_required, verbose])	Collect codes of <code>railway bridges</code> from source web page.
<code>fetch_codes</code> ([update, dump_dir, verbose])	Fetch codes of <code>railway bridges</code> .

Bridges.collect_codes

`Bridges.collect_codes(confirmation_required=True, verbose=False)`
Collect codes of `railway bridges` from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

data of railway bridges and date of when the data was last updated

Return type

`dict` or `None`

Examples:

```

>>> from pyrcs.line_data import Bridges # from pyrcs import Bridges

>>> bdg = Bridges()

>>> bdg_codes = bdg.collect_codes()
To collect data of railway bridges
? [No]|Yes: yes
>>> 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'}
```

Bridges.fetch_codes

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

Fetch codes of **railway bridges**.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

data of railway bridges and date of when the data was last updated

Return type

dict or `None`

Examples:

```

>>> from pyrcs.line_data import Bridges # from pyrcs import Bridges

>>> bdg = Bridges()

>>> bdg_codes = bdg.fetch_codes()

>>> 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'}

```

3.1.2 other_assets

A sub-package of modules for collecting codes of [other assets](#).

(See also [OtherAssets](#).)

Sub-modules

<i>sig_box</i>	Collect data of signal box prefix codes .
<i>tunnel</i>	Collect data of railway tunnel lengths .
<i>viaduct</i>	Collect codes of railway viaducts .
<i>station</i>	Collect railway station data .
<i>depot</i>	Collect data of depot codes .
<i>feature</i>	Collect codes of infrastructure features.

sig_box

Collect data of [signal box prefix codes](#).

Class

<code>SignalBoxes</code> ([data_dir, update, verbose])	A class for collecting data of signal box prefix codes .
--	--

SignalBoxes

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

A class for collecting data of [signal box prefix codes](#).

Parameters

- **data_dir** (*str* or *None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to True

Variables

- **catalogue** (*dict*) – catalogue of the data
- **last_updated_date** (*str*) – last updated date
- **data_dir** (*str*) – path to the data directory
- **current_data_dir** (*str*) – 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

<i>KEY</i>	Key of the dict -type data
<i>KEY_TO_BELL_CODES</i>	Key of the dict-type data of ' <i>bell codes</i> '
<i>KEY_TO_IRELAND</i>	Key of the dict-type data of ' <i>Ireland</i> '
<i>KEY_TO_LAST_UPDATED_DATE</i>	Key of the data of the last updated date
<i>KEY_TO_NON_NATIONAL_RAIL</i>	Key of the dict-type data of ' <i>non-national rail</i> '
<i>KEY_TO_WRMASD</i>	Key of the dict-type data of ' <i>WR (Western region) MAS (multiple aspect signalling) dates</i> '
<i>NAME</i>	Name of the data
<i>URL</i>	URL of the main web page of the data

SignalBoxes.KEY

`SignalBoxes.KEY = 'Signal boxes'`

Key of the [dict](#)-type data

SignalBoxes.KEY_TO_BELL_CODES

`SignalBoxes.KEY_TO_BELL_CODES = 'Bell codes'`

Key of the dict-type data of '*bell codes*'

SignalBoxes.KEY_TO_IRELAND

`SignalBoxes.KEY_TO_IRELAND = 'Ireland'`

Key of the dict-type data of '*Ireland*'

SignalBoxes.KEY_TO_LAST_UPDATED_DATE

`SignalBoxes.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'`

Key of the data of the last updated date

SignalBoxes.KEY_TO_NON_NATIONAL_RAIL

`SignalBoxes.KEY_TO_NON_NATIONAL_RAIL = 'Non-National Rail'`

Key of the dict-type data of '*non-national rail*'

SignalBoxes.KEY_TO_WRMASD

```
SignalBoxes.KEY_TO_WRMASD = 'WR MAS dates'
```

Key of the dict-type data of 'WR (Western region) MAS (multiple aspect signalling) dates'

SignalBoxes.NAME

```
SignalBoxes.NAME = 'Signal box prefix codes'
```

Name of the data

SignalBoxes.URL

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

URL of the main web page of the data

Methods

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

SignalBoxes.collect_bell_codes

`SignalBoxes.collect_bell_codes(confirmation_required=True, verbose=False)`

Collect data of **bell codes** from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

bell codes for the signal box prefix codes

Return type

dict or 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)
collections.OrderedDict
>>> 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
```

(continues on next page)

(continued from previous page)

```

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)`

Collect data of [Irish signal cabin prefix codes](#) from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to False

Returns

signal box prefix codes of Ireland

Return type

dict or 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)`

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

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

signal box prefix codes of non-national rail

Return type

dict or None

Examples:

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

>>> sb = SignalBoxes()

>>> 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())
```

(continues on next page)

(continued from previous page)

```

['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, update=False, verbose=False)`

Collect signal box prefix codes beginning with a given initial letter from source web page.

Parameters

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

Returns

data of signal box prefix codes beginning with the given initial letter and 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  ...

```

(continues on next page)

(continued from previous page)

```

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*)

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

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict or 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)
collections.defaultdict
>>> list(sb_wr_mas_dates_dat.keys())
['Paddington-Hayes',
 'Birmingham',
 'Plymouth',
 'Reading-Hayes',
 'Newport Multiple Aspect Signalling',
 'Old Oak Common (original scheme)',
 'Port Talbot Multiple Aspect Signalling',

```

(continues on next page)

(continued from previous page)

```

'Reading Multiple Aspect Signalling',
'Original Barry amalgamation',
'Cornwall',
'Cardiff Multiple Aspect Signalling',
'Central Wales',
'Gloucester Multiple Aspect Signalling',
'Swindon Multiple Aspect Signalling',
'Bristol Division (miscellaneous schemes)',
'Old Oak Common (new panel)',
'Western Valleys',
'London Division (miscellaneous schemes)',
'Cardiff Valleys',
'Newport Extension',
'Barry centralisation',
'Slough/Reading (developments)',
'Bristol Multiple Aspect Signalling',
'Port Talbot Multiple Aspect Signalling (extensions and developments)',
'Miscellaneous',
'Old Oak Common (rationalisation)',
'Centralisation schemes',
'Bristol (developments)',
'Devon',
'Didcot/Swindon/Bristol reversible working',
'Reading West extension',
'Cardiff-Carmarthen-Whitland']

>>> 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*)

Fetch data of bell codes.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – name of package data folder, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

data of bell codes

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)
collections.OrderedDict
>>> 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)

Fetch data of [Irish signal cabin prefix codes](#).

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to *False*
- **dump_dir** (*str* or *None*) – name of package data folder, defaults to *None*
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to *False*

Returns

signal box prefix codes of Ireland

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)

Fetch signal box prefix codes of **non-national rail**.**Parameters**

- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **dump_dir** (*str or None*) – name of package data folder, defaults to None
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

signal box prefix codes of non-national rail

Return type

dict

Examples:

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

>>> sb = SignalBoxes()
```

(continues on next page)

(continued from previous page)

```

>>> 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]
>>> 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(update=False, dump_dir=None, verbose=False)

Fetch data of signal box prefix codes.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **dump_dir** (*str* or *None*) – name of package data folder, defaults to None

- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

data of location codes and 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_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  ...  16 February 1992  Nuneaton (NN)
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)`

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

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **dump_dir** (*str or None*) – name of package data folder, defaults to None
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

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

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
>>> 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)
collections.defaultdict
>>> list(sb_wr_mas_dates_dat.keys())
['Paddington-Hayes',
 'Birmingham',
 'Plymouth',
 'Reading-Hayes',
 'Newport Multiple Aspect Signalling',
 'Old Oak Common (original scheme)',
 'Port Talbot Multiple Aspect Signalling',
 'Reading Multiple Aspect Signalling',
 'Original Barry amalgamation',
 'Cornwall',
 'Cardiff Multiple Aspect Signalling',
 'Central Wales',
 'Gloucester Multiple Aspect Signalling',
 'Swindon Multiple Aspect Signalling',
 'Bristol Division (miscellaneous schemes)',
 'Old Oak Common (new panel)',
 'Western Valleys',
 'London Division (miscellaneous schemes)',
 'Cardiff Valleys',
 'Newport Extension',
 'Barry centralisation',
 'Slough/Reading (developments)',
 'Bristol Multiple Aspect Signalling',
 'Port Talbot Multiple Aspect Signalling (extensions and developments)',
 'Miscellaneous',
 'Old Oak Common (rationalisation)',
 'Centralisation schemes',
 'Bristol (developments)',
 'Devon',
 'Didcot/Swindon/Bristol reversible working',
 'Reading West extension',
 'Carmarthen-Whitland']

>>> 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

tunnel

Collect data of [railway tunnel lengths](#).

Class

<code>Tunnels</code> ([data_dir, update, verbose])	A class for collecting data of railway tunnel lengths .
--	---

Tunnels

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

A class for collecting data of [railway tunnel lengths](#).

Parameters

- **data_dir** (*str* or *None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to True

Variables

- **catalogue** (*dict*) – catalogue of the data
- **last_updated_date** (*str*) – last updated date
- **data_dir** (*str*) – path to the data directory
- **current_data_dir** (*str*) – 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

<i>KEY</i>	Key of the dict -type data
<i>KEY_TO_LAST_UPDATED_DATE</i>	Key of the data of the last updated date
<i>NAME</i>	Name of the data
<i>URL</i>	URL of the main web page of the data

Tunnels.KEY

`Tunnels.KEY = 'Tunnels'`
Key of the [dict](#)-type data

Tunnels.KEY_TO_LAST_UPDATED_DATE

`Tunnels.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'`
Key of the data of the last updated date

Tunnels.NAME

`Tunnels.NAME = 'Railway tunnel lengths'`
Name of the data

Tunnels.URL

`Tunnels.URL = 'http://www.railwaycodes.org.uk/tunnels/tunnels0.shtm'`
URL of the main web page of the data

Methods

<i>collect_codes_by_page</i> (page_no[, update, verbose])	Collect data of railway tunnel lengths for a page number from source web page.
<i>fetch_codes</i> ([update, dump_dir, verbose])	Fetch data of railway tunnel lengths .
<i>parse_length</i> (x)	Parse data in 'Length' column, i.e. convert miles/yards to metres.

Tunnels.collect_codes_by_page

`Tunnels.collect_codes_by_page(page_no, update=False, verbose=False)`

Collect data of [railway tunnel lengths](#) for a page number from source web page.

Parameters

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

Returns

data of tunnel lengths on page `page_no` and 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_by_page(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_by_page(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']
```

(continues on next page)

(continued from previous page)

```

>>> 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]

```

Tunnels.fetch_codes

`Tunnels.fetch_codes(update=False, dump_dir=None, verbose=False)`

Fetch data of [railway tunnel lengths](#).

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – name of a folder where the pickle file is to be saved, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

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

Return type

dict

Examples:

```

>>> from pyrcs.other_assets import Tunnels # from pyrcs import Tunnels

```

(continues on next page)

(continued from previous page)

```

>>> tunl = Tunnels()

>>> tunl_len_codes = tunl.fetch_codes()
>>> type(tunl_len_codes)
dict
>>> list(tunl_len_codes.keys())
['Tunnels', 'Last updated date']

>>> 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

```

(continues on next page)

(continued from previous page)

2	A720		...	145.3896
3	A9	Aberdeen line	...	141.7320
4	A9	Perth line	...	146.3040
[5 rows x 8 columns]				

Tunnels.parse_length

static Tunnels.parse_length(*x*)

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

Parameters

x (*str* or *None*) – raw length data

Returns

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

Return type

tuple

Examples:

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

>>> tunl = Tunnels()

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

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

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

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

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

viaduct

Collect codes of [railway viaducts](#).

Class

<code>Viaducts</code> ([<code>data_dir</code> , <code>update</code> , <code>verbose</code>])	A class for collecting codes of railway viaducts .
--	--

Viaducts

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

A class for collecting codes of [railway viaducts](#).

Parameters

- **data_dir** (*str* or *None*) – name of data directory, defaults to *None*
- **update** (*bool*) – whether to do an update check (for the package data), defaults to *False*
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to *True*

Variables

- **catalogue** (*dict*) – catalogue of the data
- **last_updated_date** (*str*) – last updated date
- **data_dir** (*str*) – path to the data directory
- **current_data_dir** (*str*) – 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

<code>KEY</code>	Key of the dict -type data
<code>KEY_TO_LAST_UPDATED_DATE</code>	Key of the data of the last updated date
<code>NAME</code>	Name of the data
<code>URL</code>	URL of the main web page of the data

Viaducts.KEY

```
Viaducts.KEY = 'Viaducts'
```

Key of the `dict`-type data

Viaducts.KEY_TO_LAST_UPDATED_DATE

```
Viaducts.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'
```

Key of the data of the last updated date

Viaducts.NAME

```
Viaducts.NAME = 'Railway viaducts'
```

Name of the data

Viaducts.URL

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

URL of the main web page of the data

Methods

<code>collect_codes_by_page</code> (page_no[, update, verbose])	Collect data of <code>railway viaducts</code> for a given page number from source web page.
<code>fetch_codes</code> ([update, dump_dir, verbose])	Fetch data of <code>railway viaducts</code> .

Viaducts.collect_codes_by_page

```
Viaducts.collect_codes_by_page(page_no, update=False, verbose=False)
```

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

Parameters

- **page_no** (*int* or *str*) – page number; valid values include 1, 2, 3, 4, 5, and 6
- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

data of railway viaducts on page `page_no` and 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_by_page(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)
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(update=False, dump_dir=None, verbose=False)`

Fetch data of [railway viaducts](#).

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – name of a folder where the pickle file is to be saved, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

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

Return type

dict

Examples:

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

>>> vdct = Viaducts()
```

(continues on next page)

(continued from previous page)

```

>>> 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)',
 '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]

```

station

Collect [railway station data](#).

Class

<code>Stations([data_dir, update, verbose])</code>	A class for collecting railway station data .
--	---

Stations

class `pyrcs.other_assets.station.Stations`(*data_dir=None, update=False, verbose=True*)
 A class for collecting [railway station data](#).

Parameters

- **data_dir** (*str* or *None*) – name of data directory, defaults to *None*
- **update** (*bool*) – whether to do an update check (for the package data), defaults to *False*

- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to True

Variables

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

Examples:

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

>>> stn = Stations()

>>> stn.NAME
'Railway station data'

>>> stn.URL
'http://www.railwaycodes.org.uk/stations/station0.shtm'
```

Attributes

<i>KEY</i>	Key of the <i>dict</i> -type data
<i>KEY_TO_LAST_UPDATED_DATE</i>	Key of the data of the last updated date
<i>KEY_TO_STN</i>	Key of the dict-type data of ' <i>Mileages, operators and grid coordinates</i> '
<i>NAME</i>	Name of the data
<i>URL</i>	URL of the main web page of the data

Stations.KEY

`Stations.KEY = 'Stations'`

Key of the *dict*-type data

Stations.KEY_TO_LAST_UPDATED_DATE

`Stations.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'`

Key of the data of the last updated date

Stations.KEY_TO_STN

`Stations.KEY_TO_STN = 'Mileages, operators and grid coordinates'`

Key of the dict-type data of *'Mileages, operators and grid coordinates'*

Stations.NAME

`Stations.NAME = 'Railway station data'`

Name of the data

Stations.URL

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

URL of the main web page of the data

Methods

<code>check_row_spans(dat)</code>	Check data where there are row spans.
<code>collect_locations_by_initial(initial[, ...])</code>	Collect data of railway station locations (mileages, operators and grid coordinates) for a given initial letter.
<code>fetch_locations([update, dump_dir, verbose])</code>	Fetch data of railway station locations (mileages, operators and grid coordinates).
<code>get_catalogue([update, verbose])</code>	Get catalogue of railway station data.
<code>parse_coordinates_columns(dat)</code>	Parse 'Degrees Longitude' and 'Degrees Latitude' of the station locations data.
<code>parse_elr_mileage_columns(dat)</code>	Parse 'ELR' and 'Mileage' of the station locations data.
<code>parse_owner_and_operator_columns(dat)</code>	Parse 'Owner' and 'Operator' of the station locations data.
<code>parse_station_column(dat)</code>	Parse 'Station' of the station locations data.

Stations.check_row_spans

`static Stations.check_row_spans(dat)`

Check data where there are row spans.

Parameters

`dat` (*pandas.DataFrame*) – preprocessed data of the station locations

Returns

data with row spans (if any)

Return type

pandas.DataFrame

Stations.collect_locations_by_initial`Stations.collect_locations_by_initial(initial, update=False, verbose=False)`

Collect [data of railway station locations](#) (mileages, operators and grid coordinates) for a given initial letter.

Parameters

- **initial** (*str*) – initial letter of locations of the railway station data
- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

data of railway station locations beginning with the given initial letter 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_by_initial(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',
```

(continues on next page)

(continued from previous page)

```

'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_locations

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

Fetch [data of railway station locations](#) (mileages, operators and grid coordinates).

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str or None*) – name of a folder where the pickle file is to be saved, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

data of railway station locations 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_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'

```

(continues on next page)

(continued from previous page)

```

>>> 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

```

Stations.get_catalogue

`Stations.get_catalogue(update=False, verbose=False)`

Get catalogue of railway station data.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

catalogue of railway station data

Return type

dict

Examples:

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

>>> stn = Stations()

>>> stn_data_cat = stn.get_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.parse_coordinates_columns

static Stations.parse_coordinates_columns(*dat*)

Parse 'Degrees Longitude' and 'Degrees Latitude' of the station locations data.

Parameters

dat (*pandas.DataFrame*) – preprocessed data of the station locations

Returns

data with parsed coordinates

Return type

pandas.DataFrame

Stations.parse_elr_mileage_columns

static Stations.parse_elr_mileage_columns(*dat*)

Parse 'ELR' and 'Mileage' of the station locations data.

Parameters

dat (*pandas.DataFrame*) – preprocessed data of the station locations

Returns

data with parsed 'ELR' and 'Mileage'

Return type

pandas.DataFrame

Stations.parse_owner_and_operator_columns

`Stations.parse_owner_and_operator_columns(dat)`

Parse 'Owner' and 'Operator' of the station locations data.

Parameters

dat (*pandas.DataFrame*) – preprocessed data of the station locations

Returns

data with parsed information of owners and operators

Return type

pandas.DataFrame

Stations.parse_station_column

static `Stations.parse_station_column(dat)`

Parse 'Station' of the station locations data.

Parameters

dat (*pandas.DataFrame*) – preprocessed data of the station locations

Returns

data with parsed station names and their corresponding CRS

Return type

pandas.DataFrame

Tests:

```
x = 'Hythe Road / [CRS awaited] '
x = 'Heathrow Junction [sometimes referred to as Heathrow Interchange]
→[no CRS?] ' /_
```

depot

Collect data of [depot codes](#).

Class

<code>Depots([data_dir, update, verbose])</code>	A class for collecting data of depot codes .
--	--

Depots

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

A class for collecting data of [depot codes](#).

Parameters

- **data_dir** (*str* or *None*) – name of data directory, defaults to *None*
- **update** (*bool*) – whether to do an update check (for the catalogue data), defaults to *False*
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to *True*

Variables

- **catalogue** (*dict*) – catalogue of the data
- **last_updated_date** (*str*) – last updated date
- **data_dir** (*str*) – path to the data directory
- **current_data_dir** (*str*) – 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

<i>KEY</i>	Key of the dict -type data
<i>KEY_TO_1950_SYSTEM</i>	Key of the dict-type data of 1950 system (pre-TOPS) codes
<i>KEY_TO_GWR</i>	Key of the dict-type data of GWR codes
<i>KEY_TO_LAST_UPDATED_DATE</i>	Key of the data of the last updated date
<i>KEY_TO_PRE_TOPS</i>	Key of the dict-type data of four digit pre-TOPS codes
<i>KEY_TO_TOPS</i>	Key of the dict-type data of two character TOPS codes
<i>NAME</i>	Name of the data
<i>URL</i>	URL of the main web page of the data

Depots.KEY

`Depots.KEY = 'Depots'`

Key of the dict-type data

Depots.KEY_TO_1950_SYSTEM

`Depots.KEY_TO_1950_SYSTEM = '1950 system (pre-TOPS) codes'`

Key of the dict-type data of 1950 system (pre-TOPS) codes

Depots.KEY_TO_GWR

`Depots.KEY_TO_GWR = 'GWR codes'`

Key of the dict-type data of GWR codes

Depots.KEY_TO_LAST_UPDATED_DATE

`Depots.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'`

Key of the data of the last updated date

Depots.KEY_TO_PRE_TOPS

`Depots.KEY_TO_PRE_TOPS = 'Four digit pre-TOPS codes'`

Key of the dict-type data of four digit pre-TOPS codes

Depots.KEY_TO_TOPS

`Depots.KEY_TO_TOPS = 'Two character TOPS codes'`

Key of the dict-type data of two character TOPS codes

Depots.NAME

`Depots.NAME = 'Depot codes'`

Name of the data

Depots.URL

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

URL of the main web page of the data

Methods

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

Depots.collect_1950_system_codes

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

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

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to `True`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

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

Return type

dict or `None`

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)`

Collect [Great Western Railway \(GWR\) depot codes](#) from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict or 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

```

(continues on next page)

(continued from previous page)

```

>>> 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)`

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

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

data of four-digit pre-TOPS codes and date of when the data was last updated

Return type

dict or 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)

```

(continues on next page)

(continued from previous page)

```
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*)

Collect **two-character TOPS codes** from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict or 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'
```

(continues on next page)

(continued from previous page)

```

>>> 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)`

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

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

data of 1950 system (pre-TOPS) codes and date of when the data was 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()
>>> 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

```

(continues on next page)

(continued from previous page)

```
>>> 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)

Fetch data of depot codes.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **dump_dir** (*str or None*) – pathname of a directory where the data file is dumped, defaults to None
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

data of depot codes and date of when the data was 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',
 'Four digit pre-TOPS codes',
 'GWR codes',
 'Two character TOPS codes']
```

(continues on next page)

(continued from previous page)

```

>>> 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)`

Fetch data of [Great Western Railway \(GWR\)](#) depot codes.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

data of GWR depot codes and date of when the data was 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())

```

(continues on next page)

(continued from previous page)

```

['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.fetch_pre_tops_codes

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

Fetch data of four-digit pre-TOPS codes.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

data of four-digit pre-TOPS codes and date of when the data was 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())

```

(continues on next page)

(continued from previous page)

```
['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.fetch_tops_codes

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

Fetch data of **two-character TOPS codes**.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str or None*) – pathname of a directory where the data file is dumped, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

data of two-character TOPS codes and date of when the data was 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]
```

(continues on next page)

(continued from previous page)

```
>>> 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]
```

feature

Collect codes of infrastructure features.

This category includes:

- [HABD and WILD](#)
- [Water troughs](#)
- [Telegraph codes](#)
- [Driver/guard buzzer codes](#)

Class

<i>Features</i> ([data_dir, update, verbose])	A class for collecting codes of several infrastructure features.
---	--

Features

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

A class for collecting codes of several infrastructure features.

Parameters

- **data_dir** (*str* or *None*) – name of data directory, defaults to None
- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to True

Variables

- **catalogue** (*dict*) – catalogue of the data
- **last_updated_date** (*str*) – last updated date
- **data_dir** (*str*) – path to the data directory

- `current_data_dir` (*str*) – path to the current data directory

Examples:

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

>>> feats = Features()

>>> feats.NAME
'Infrastructure features'
```

Attributes

<code>KEY</code>	Key of the <code>dict</code> -type data
<code>KEY_TO_BUZZER</code>	Key of the dict-type data of 'buzzer codes'
<code>KEY_TO_HABD_WILD</code>	Key of the dict-type data of 'HABD' and 'WILD'
<code>KEY_TO_LAST_UPDATED_DATE</code>	Key of the data of the last updated date
<code>KEY_TO_TELEGRAPH</code>	Key of the dict-type data of 'telegraph codes'
<code>KEY_TO_TROUGH</code>	Key of the dict-type data of 'water troughs'
<code>NAME</code>	Name of the data

Features.KEY

`Features.KEY = 'Features'`

Key of the `dict`-type data

Features.KEY_TO_BUZZER

`Features.KEY_TO_BUZZER = 'Buzzer codes'`

Key of the dict-type data of 'buzzer codes'

Features.KEY_TO_HABD_WILD

`Features.KEY_TO_HABD_WILD = 'HABD and WILD'`

Key of the dict-type data of 'HABD' and 'WILD'

Features.KEY_TO_LAST_UPDATED_DATE

```
Features.KEY_TO_LAST_UPDATED_DATE = 'Last updated date'
```

Key of the data of the last updated date

Features.KEY_TO_TELEGRAPH

```
Features.KEY_TO_TELEGRAPH = 'Telegraphic codes'
```

Key of the dict-type data of *'telegraph codes'*

Features.KEY_TO_TROUGH

```
Features.KEY_TO_TROUGH = 'Water troughs'
```

Key of the dict-type data of *'water troughs'*

Features.NAME

```
Features.NAME = 'Infrastructure features'
```

Name of the data

Methods

<code>collect_buzzer_codes([...])</code>	Collect data of <i>buzzer codes</i> from source web page.
<code>collect_habds_and_wilds([...])</code>	Collect codes of <i>HABDs</i> and <i>WILDs</i> from source web page.
<code>collect_telegraph_codes([...])</code>	Collect data of <i>telegraph code words</i> from source web page.
<code>collect_water_troughs([...])</code>	Collect codes of <i>water troughs</i> locations from source web page.
<code>fetch_buzzer_codes([update, dump_dir, verbose])</code>	Fetch data of <i>buzzer codes</i> .
<code>fetch_codes([update, dump_dir, verbose])</code>	Fetch codes of infrastructure features.
<code>fetch_habds_and_wilds([update, dump_dir, ...])</code>	Fetch codes of <i>HABDs</i> and <i>WILDs</i> .
<code>fetch_telegraph_codes([update, dump_dir, ...])</code>	Fetch data of <i>telegraph code words</i> .
<code>fetch_water_troughs([update, dump_dir, verbose])</code>	Fetch codes of <i>water troughs</i> locations.

Features.collect_buzzer_codes

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

Collect data of **buzzer codes** from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict or None

Examples:

```
>>> from pyrcs.other_assets import Features # from pyrcs import Features
>>> feats = Features()
>>> buz_codes = feats.collect_buzzer_codes()
To collect data of Buzzer codes
? [No]|Yes: yes
>>> type(buz_codes)
dict
>>> list(buz_codes.keys())
['Buzzer codes', 'Last updated date']

>>> feats.KEY_TO_BUZZER
'Buzzer codes'

>>> buz_codes_dat = buz_codes[feats.KEY_TO_BUZZER]
>>> 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
```

Features.collect_habds_and_wilds

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

Collect codes of HABDs and WILDs from source web page.

Note:

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

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict or None

Examples:

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

>>> feats = Features()

>>> hw_codes = feats.collect_habds_and_wilds()
To collect data of HABD and WILD
? [No]|Yes: yes
>>> type(hw_codes)
dict
>>> list(hw_codes.keys())
['HABD and WILD', 'Last updated date']

>>> feats.KEY_TO_HABD_WILD
'HABD and WILD'

>>> hw_codes_dat = hw_codes[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  ...
0  BAG2  ...
```

Notes

(continues on next page)

(continued from previous page)

```

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]

```

Features.collect_telegraph_codes

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

Collect data of **telegraph code words** from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict or None

Examples:

```

>>> from pyrcs.other_assets import Features # from pyrcs import Features
>>> feats = Features()
>>> tel_codes = feats.collect_telegraph_codes()
To collect data of Telegraphic codes
? [No]|Yes: yes
>>> type(tel_codes)
dict
>>> list(tel_codes.keys())
['Telegraphic codes', 'Last updated date']
>>> feats.KEY_TO_TELEGRAPH
'Telegraphic codes'
>>> tel_codes_dat = tel_codes[feats.KEY_TO_TELEGRAPH]

```

(continues on next page)

(continued from previous page)

```

>>> 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...

```

Features.collect_water_troughs

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

Collect codes of [water troughs locations](#) from source web page.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

data of water trough locations, and date of when the data was last updated

Return type

dict or None

Examples:

```

>>> from pyrcs.other_assets import Features # from pyrcs import Features

>>> feats = Features()

>>> wt_codes = feats.collect_water_troughs()
To collect data of Water troughs

```

(continues on next page)

(continued from previous page)

```

? [No] | Yes: yes
>>> type(wt_codes)
dict
>>> list(wt_codes.keys())
['Water troughs', 'Last updated date']

>>> feats.KEY_TO_TROUGH
'Water troughs'

>>> wt_codes_dat = wt_codes[feats.KEY_TO_TROUGH]
>>> 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]

```

Features.fetch_buzzer_codes

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

Fetch data of **buzzer codes**.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str or None*) – name of a folder where the pickle file is to be saved, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

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

Return type

dict

Examples:

```

>>> from pyrcs.other_assets import Features # from pyrcs import Features

>>> feats = Features()

>>> buz_codes = feats.fetch_buzzer_codes()
>>> type(buz_codes)
dict
>>> list(buz_codes.keys())
['Buzzer codes', 'Last updated date']

```

(continues on next page)

(continued from previous page)

```

>>> feats.KEY_TO_BUZZER
'Buzzer codes'

>>> buz_codes_dat = buz_codes[feats.KEY_TO_BUZZER]
>>> 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

```

Features.fetch_codes

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

Fetch codes of infrastructure features.

Including:

- [HABD and WILD](#)
- [Water troughs](#)
- [Telegraph codes](#)
- [Driver/guard buzzer codes](#)

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **dump_dir** (*str* or *None*) – name of a folder where the pickle file is to be saved, defaults to None
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict

Examples:

```

>>> from pyrcs.other_assets import Features # from pyrcs import Features
>>> feats = Features()

```

(continues on next page)

(continued from previous page)

```

>>> 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
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]

```

Features.fetch_habds_and_wilds

Features.fetch_habds_and_wilds(update=False, dump_dir=None, verbose=False)

Fetch codes of HABDs and WILDs.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **dump_dir** (*str* or *None*) – name of a folder where the pickle file is to be saved, defaults to None
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Features # from pyrcs import Features
>>> feats = Features()
>>> hw_codes = feats.fetch_habds_and_wilds()
>>> type(hw_codes)
dict
>>> list(hw_codes.keys())
['HABD and WILD', 'Last updated date']
>>> feats.KEY_TO_HABD_WILD
'HABD and WILD'
>>> hw_codes_dat = hw_codes[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']
```

(continues on next page)

(continued from previous page)

```
>>> 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]
```

Features.fetch_telegraph_codes

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

Fetch data of telegraph code words.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str* or *None*) – name of a folder where the pickle file is to be saved, defaults to `None`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Returns

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

Return type

dict

Examples:

```
>>> from pyrcs.other_assets import Features # from pyrcs import Features
>>> feats = Features()
>>> tel_codes = feats.fetch_telegraph_codes()
>>> type(tel_codes)
dict
>>> list(tel_codes.keys())
['Telegraphic codes', 'Last updated date']
>>> feats.KEY_TO_TELEGRAPH
'Telegraphic codes'
>>> tel_codes_dat = tel_codes[feats.KEY_TO_TELEGRAPH]
>>> type(tel_codes_dat)
dict
>>> list(tel_codes_dat.keys())
['Official codes', 'Unofficial codes']
>>> tel_official_codes = tel_codes_dat['Official codes']
```

(continues on next page)

(continued from previous page)

```

>>> 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...

```

Features.fetch_water_troughs

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

Fetch codes of [water troughs locations](#).

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **dump_dir** (*str or None*) – name of a folder where the pickle file is to be saved, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

data of water trough locations, and date of when the data was last updated

Return type

dict

Examples:

```

>>> from pyrcs.other_assets import Features # from pyrcs import Features

>>> feats = Features()

>>> wt_codes = feats.fetch_water_troughs()
>>> type(wt_codes)
dict
>>> list(wt_codes.keys())
['Water troughs', 'Last updated date']

```

(continues on next page)

(continued from previous page)

```

>>> feats.KEY_TO_TROUGH
'Water troughs'

>>> wt_codes_dat = wt_codes[feats.KEY_TO_TROUGH]
>>> 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]

```

3.2 Modules

<i>parser</i>	Parse web-page contents.
<i>converter</i>	Change data into a desired form.
<i>collector</i>	Collect data of railway codes.
<i>utils</i>	Provide a number of helper functions.

3.2.1 parser

Parse web-page contents.

Preprocess contents

<i>parse_tr</i> (trs, ths[, sep, as_dataframe])	Parse a list of parsed HTML <tr> elements.
<i>parse_table</i> (source[, parser, as_dataframe])	Parse HTML <tr> elements for creating a data frame.
<i>parse_date</i> (str_date[, as_date_type])	Parse a date.

parse_tr

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

Parse a list of parsed HTML <tr> elements.

See also [PT-1].

Parameters

- **trs** (*bs4.ResultSet* or *list*) – contents under <tr> tags of a web page

- **ths** (*list or bs4.element.Tag*) – list of column names (usually under a `<th>` tag) of a requested table
- **sep** (*str or None*) – separator that replaces the one in the raw data
- **as_dataframe** (*bool*) – whether to return the parsed data in tabular form

Returns

a list of lists that each comprises a row of the requested table

Return type

`pandas.DataFrame` or `List[list]`

Example:

```
>>> 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(markup=source.content, features='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)`

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

Parameters

- **source** (*requests.Response*) – response object to connecting a URL to request a table
- **parser** (*str*) – 'html.parser' (default), 'html5lib' or 'lxml'
- **as_dataframe** (*bool*) – whether to return the parsed data in tabular form

Returns

a list of lists each comprising a row of the requested table (see also `pyrcs.utils.parse_tr()`) and a list of column names of the requested table

Return type

tuple[list, list] or pandas.DataFrame or 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)

Parse a date.

Parameters

- **str_date** (*str*) – string-type date
- **as_date_type** (*bool*) – whether to return the date as `datetime.date`, defaults to False

Returnsparsed date as a string or `datetime.date`**Return type**str or `datetime.date`**Examples:**

```
>>> from pyrcs.parser import parse_date

>>> str_date_dat = '2020-01-01'

>>> parsed_date_dat = parse_date(str_date_dat)
>>> parsed_date_dat
'2020-01-01'

>>> parsed_date_dat = parse_date(str_date_dat, as_date_type=True)
>>> parsed_date_dat
datetime.date(2020, 1, 1)
```

Extract information

<code>get_site_map([update, ...])</code>	Fetch the site map from the package data.
<code>get_last_updated_date(url[, parsed, ...])</code>	Get last update date.
<code>get_financial_year(date)</code>	Convert calendar year of a given date to Network Rail financial year.
<code>get_catalogue(url[, update, ...])</code>	Get the catalogue for a class.
<code>get_category_menu(url[, update, ...])</code>	Get a menu of the available classes.
<code>get_page_catalogue(url[, head_tag_name, ...])</code>	Get the catalogue of the main page of a data cluster.
<code>get_heading_text(heading_tag[, elem_tag_name])</code>	Get the text of a given heading tag.
<code>get_page_catalogue(url[, head_tag_name, ...])</code>	Get the catalogue of the main page of a data cluster.
<code>get_hypertext(hypertext_tag[, ...])</code>	Get text that is with a hyperlink.
<code>get_introduction(url[, delimiter, verbose])</code>	Get contents of the Introduction page.

get_site_map

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

Fetch the [site map](#) from the package data.

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

dictionary of site map data

Return type

`collections.OrderedDict` or `None`

Examples:

```
>>> from pyrcs.parser import get_site_map

>>> site_map_dat = get_site_map()

>>> type(site_map_dat)
collections.OrderedDict
>>> list(site_map_dat.keys())
['Home',
 'Line data',
 'Other assets',
```

(continues on next page)

(continued from previous page)

```

    "Legal/financial" lists',
    'Miscellaneous']
>>> site_map_dat['Home']
{'index.shtml': '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)`

Get last update date.

Parameters

- **url** (*str*) – URL link of a requested web page
- **parsed** (*bool*) – whether to reformat the date, defaults to `True`
- **as_date_type** (*bool*) – whether to return the date as `datetime.date`, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns

date of when the specified web page was last updated

Return type

`str` or `datetime.date` or `None`

Examples:

```

>>> from pyrcs.parser import get_last_updated_date

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

>>> last_upd_date = get_last_updated_date(url_a, parsed=True, as_date_type=True)
>>> type(last_upd_date)
datetime.date

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

get_financial_year

`pyrcs.parser.get_financial_year(date)`

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

Parameters

date (*datetime.datetime*) – date

Returns

Network Rail financial year of the given date

Return type

int

Example:

```
>>> 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, confirmation_required=True, json_it=True, verbose=False)`

Get the catalogue for a class.

Parameters

- **url** (*str*) – URL of the main page of a data cluster
- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **json_it** (*bool*) – whether to save the catalogue as a JSON file, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

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

Return type

dict or 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']

>>> line_data_cat = get_catalogue(url='http://www.railwaycodes.org.uk/linedatamenu.shtm')
>>> type(line_data_cat)
dict
>>> list(line_data_cat.keys())
['ELRs and mileages',
 'Electrification masts and related features',
 'CRS, NLC, TIPLOC and STANOX Codes',
 'Line of Route (LOR/PRIDE) codes',
 'Line names',
 'Track diagrams']

```

get_category_menu

`pyrcs.parser.get_category_menu(url, update=False, confirmation_required=True, json_it=True, verbose=False)`

Get a menu of the available classes.

Parameters

- **url** (*str*) – URL of the menu page
- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **json_it** (*bool*) – whether to save the catalogue as a .json file, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

a category menu

Return type

dict or None

Example:

```
>>> from pyrcs.parser import get_category_menu

>>> menu = get_category_menu('http://www.railwaycodes.org.uk/linedatamenu.shtml')

>>> type(menu)
dict
>>> list(menu.keys())
['Line data']
```

get_page_catalogue

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

Get the catalogue of the main page of a data cluster.

Parameters

- **url** (*str*) – URL of the main page of a data cluster
- **head_tag_name** (*str*) – tag name of the feature list at the top of the page, defaults to 'nav'
- **head_tag_txt** (*str*) – text that is contained in the head_tag, defaults to 'Jump to: '
- **feature_tag_name** (*str*) – tag name of the headings of each feature, defaults to 'h3'
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to False

Returns

catalogue of the main page of a data cluster

Return type

pandas.DataFrame

Example:

```
>>> 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.shtml'

>>> 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...	...

(continues on next page)

(continued from previous page)

```

..                               ... ..
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')`

Get the text of a given heading tag.

Parameters

- **heading_tag** (*bs4.element.Tag*) – tag of a heading
- **elem_tag_name** (*str*) – tag name of an element in the heading_tag, defaults to 'em'

Returns

cleansed text of the given 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)`

Get text that is with a hyperlink.

Parameters

- **hypertext_tag** (*bs4.element.Tag* or *bs4.element.PageElement*) – tag of hypertext (i.e. text that is with a hyperlink)
- **hyperlink_tag_name** (*str*) –
- **md_style** (*bool*) – whether to return the obtained hypertext in markdown style, defaults to True

Returns

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', verbose=True)`

Get contents of the Introduction page.

Parameters

- **url** (*str*) – URL of a web page (usually the main page of a data cluster)
- **delimiter** (*str*) – delimiter used for separating paragraphs, defaults to '\n'

- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to True

Returns

introductory texts on the given web page

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..."
```

3.2.2 converter

Change data into a desired form.

Convert mileage data

<i>fix_mileage</i> (mileage)	Fix mileage data (associated with an ELR).
<i>yard_to_mileage</i> (yard[, as_str])	Convert yards to Network Rail mileages.
<i>mileage_to_yard</i> (mileage)	Convert Network Rail mileages to yards.
<i>mile_chain_to_mileage</i> (mile_chain)	Convert mileage data in the form '<miles>.<chains>' to Network Rail mileage.
<i>mileage_to_mile_chain</i> (mileage)	Convert Network Rail mileage to the form '<miles>.<chains>'.
<i>mile_yard_to_mileage</i> (mile, yard[, as_numeric])	Convert mile and yard to Network Rail mileage.
<i>mileage_str_to_num</i> (mileage)	Convert string-type Network Rail mileage to numerical-type one.
<i>mileage_num_to_str</i> (mileage)	Convert numerical-type Network Rail mileage to string-type one.
<i>shift_mileage_by_yard</i> (mileage, shift_yards)	Shift Network Rail mileage by given yards.

fix_mileage

`pyrcs.converter.fix_mileage(mileage)`

Fix mileage data (associated with an ELR).

Parameters

mileage (*str or float or None*) – Network Rail mileage

Returns

fixed mileage data in the conventional format used by Network Rail

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)`

Convert yards to Network Rail mileages.

Parameters

- **yard** (*int or float or None*) – yard data
- **as_str** (*bool*) – whether to return as a string value, defaults to True

Returns

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

Return type

str or float

Examples:

```
>>> from pyrcs.converter import yard_to_mileage

>>> mileage_dat = yard_to_mileage(yard=396)
>>> mileage_dat
'0.0396'
```

(continues on next page)

(continued from previous page)

```
>>> mileage_dat = yard_to_mileage(yard=396, as_str=False)
>>> mileage_dat
0.0396

>>> mileage_dat = yard_to_mileage(yard=None)
>>> 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)`

Convert Network Rail mileages to yards.

Parameters

mileage (*float or int or str*) – Network Rail mileage

Returns

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)`

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

Parameters

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

Returns

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

Return type

`str`

Examples:

```
>>> from pyrcs.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)`

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

Parameters

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

Returns

data presented in the form '<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)
```

(continues on next page)

(continued from previous page)

```
>>> 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)`

Convert mile and yard to Network Rail mileage.

Parameters

- **mile** (*float or int*) – mile
- **yard** (*float or int*) – yard
- **as_numeric** (*bool*) – whether to return a numeric value, defaults to True

Returns

Network Rail mileage

Return type

str or 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)`

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

Parameters

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

Returns

numerical-type Network Rail 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)`

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

Parameters

mileage (*float or None*) – numerical-type Network Rail mileage

Returns

string-type Network Rail mileage in the form '<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)`

Shift Network Rail mileage by given yards.

Parameters

- **mileage** (*float or int or str*) – mileage (associated with an ELR) used by Network Rail
- **shift_yards** (*int or float*) – yards by which the given mileage is shifted
- **as_numeric** (*bool*) – whether to return a numeric type result, defaults to True

Returns

shifted mileage

Return type

float or 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
```

Convert other data

<code>fix_stanox(stanox)</code>	Fix the format of a given STANOX (station number) code.
<code>kilometer_to_yard(km)</code>	Make kilometer-to-yard conversion.

fix_stanox

`pyrcs.converter.fix_stanox(stanox)`

Fix the format of a given STANOX (station number) code.

Parameters

stanox (*str or int or None*) – STANOX code

Returns

standard STANOX code

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)`

Make kilometer-to-yard conversion.

Parameters

km (*int or float or None*) – kilometer

Returns

yard

Return type

float

Example:

```
>>> from pyrcs.converter import kilometer_to_yard

>>> kilometer_to_yard(1)
1093.6132983377079
```

3.2.3 collector

Collect data of railway codes.

The current release only includes [line data](#) and [other assets](#).

<code>LineData([update, verbose])</code>	A class representation of all modules of the subpackage <code>line_data</code> for collecting line data .
<code>OtherAssets([update, verbose])</code>	A class representation of all modules of the subpackage <code>other_assets</code> for collecting other assets .

LineData

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

A class representation of all modules of the subpackage `line_data` for collecting [line data](#).

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to True

Variables

- **connected** (*bool*) – whether the Internet / the website can be connected
- **catalogue** (*dict*) – catalogue of the data
- **ELRMileages** (*object*) – instance of the class `ELRMileages`
- **Electrification** (*object*) – instance of the class `Electrification`
- **LocationIdentifiers** (*object*) – instance of the class `LocationIdentifiers`
- **LOR** (*object*) – instance of the class `LOR`
- **LineNames** (*object*) – instance of the class `LineNames`
- **TrackDiagrams** (*object*) – instance of the class `TrackDiagrams`
- **Bridges** (*object*) – instance of the class `Bridges`

Examples:

```
>>> from pyrcs import LineData
>>> ld = LineData()
>>> # To get data of location codes
>>> location_codes = ld.LocationIdentifiers.fetch_codes()
>>> type(location_codes)
```

(continues on next page)

(continued from previous page)

```
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

<i>NAME</i>	Name of data
<i>URL</i>	URL of the main web page of the data

LineData.NAME

LineData.NAME = 'Line data'

Name of data

LineData.URL

`LineData.URL = 'http://www.railwaycodes.org.uk/linedatamenu.shtm'`

URL of the main web page of the data

Methods

<code>update([confirmation_required, verbose, ...])</code>	Update pre-packed of the line data .
--	--

LineData.update

`LineData.update(confirmation_required=True, verbose=False, interval=5, init_update=False)`

Update pre-packed of the [line data](#).

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **interval** (*int*) – time gap (in seconds) between the updating of different classes, defaults to `5`
- **init_update** (*bool*) – whether to update the data for instantiation of each subclass, defaults to `False`

Example:

```
>>> from pyrcs.collector import LineData
>>> ld = LineData()
>>> ld.update(verbose=True)
```

OtherAssets

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

A class representation of all modules of the subpackage [other_assets](#) for collecting [other assets](#).

Parameters

- **update** (*bool*) – whether to do an update check (for the package data), defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `True`

Variables

- **connected** (*bool*) – whether the Internet / the website can be connected
- **catalogue** (*dict*) – catalogue of the data
- **SignalBoxes** (*object*) – instance of the class SignalBoxes
- **Tunnels** (*object*) – instance of the class Tunnels
- **Viaducts** (*object*) – instance of the class Viaducts
- **Stations** (*object*) – instance of the class Stations
- **Depots** (*object*) – instance of the class Depots
- **Features** (*object*) – instance of the class Features

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  ...  16 February 1992  Nuneaton (NN)
```

(continues on next page)

(continued from previous page)

```

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

<i>NAME</i>	Name of data
<i>URL</i>	URL of the main web page of the data

OtherAssets.NAME

```
OtherAssets.NAME = 'Other assets'
```

Name of data

OtherAssets.URL

```
OtherAssets.URL = 'http://www.railwaycodes.org.uk/otherassetsmenu.shtm'
```

URL of the main web page of the data

Methods

<i>update</i> ([confirmation_required, verbose, ...])	Update pre-packed data of the <i>other assets</i> .
---	---

OtherAssets.update

```
OtherAssets.update(confirmation_required=True, verbose=False, interval=5, init_update=False)
```

Update pre-packed data of the *other assets*.

Parameters

- **confirmation_required** (*bool*) – whether to confirm before proceeding, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **interval** (*int*) – time gap (in seconds) between the updating of different classes, defaults to 5
- **init_update** (*bool*) – whether to update the data for instantiation of each subclass, defaults to False

Example:

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

>>> oa = OtherAssets()

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

3.2.4 utils

Provide a number of helper functions.

Validate inputs

<code>is_home_connectable()</code>	Check whether the Railway Codes website is connectable.
<code>is_str_float(x)</code>	Check if a string-type variable can express a float-type value.
<code>validate_initial(x[, as_is])</code>	Get a valid initial letter as an input.
<code>validate_page_name(cls, page_no, valid_page_no)</code>	Get a valid page name.
<code>collect_in_fetch_verbose(data_dir, verbose)</code>	Create a new parameter that indicates whether to print relevant information in console (used only if it is necessary to re-collect data when trying to fetch the data).
<code>fetch_all_verbose(data_dir, verbose)</code>	Create a new parameter that indicates whether to print relevant information in console (used only when trying to fetch all data of a cluster).

is_home_connectable

`pyrcs.utils.is_home_connectable()`

Check whether the Railway Codes website is connectable.

Returns

whether the Railway Codes website is connectable

Return type

bool

Example:

```
>>> from pyrcs.utils import is_home_connectable

>>> is_home_connectable()
True
```

is_str_float

`pyrcs.utils.is_str_float(x)`

Check if a string-type variable can express a float-type value.

Parameters

x (*str*) – a string-type variable

Returns

whether `str_val` can express a float value

Return type

`bool`

Examples:

```
>>> from pyrcs.utils import is_str_float
>>> is_str_float('')
False
>>> is_str_float('a')
False
>>> is_str_float('1')
True
>>> is_str_float('1.1')
True
```

validate_initial

`pyrcs.utils.validate_initial(x, as_is=False)`

Get a valid initial letter as an input.

Parameters

- **x** (*str*) – any string variable (which is supposed to be an initial letter)
- **as_is** (*bool*) – whether to return the validated letter as is the input

Returns

validated initial letter

Return type

`str`

Examples:

```
>>> from pyrcs.utils import validate_initial
>>> validate_initial('x')
'X'
```

(continues on next page)

(continued from previous page)

```
>>> validate_initial('x', as_is=True)
'x'

>>> validate_initial('xyz')
AssertionError: `x` must be a single letter.
```

validate_page_name

`pyrcs.utils.validate_page_name(cls, page_no, valid_page_no)`

Get a valid page name.

Parameters

- `cls` (*any*) – instance of a class
- `page_no` (*int* or *str*) – page number
- `valid_page_no` (*set* or *list* or *tuple*) – all valid page numbers

Returns

validated page name of the given `cls`

Return type

`str`

See also:

- Examples for the methods `Tunnels.collect_codes_by_page()` and `Tunnels.collect_codes_by_page()`.

collect_in_fetch_verbose

`pyrcs.utils.collect_in_fetch_verbose(data_dir, verbose)`

Create a new parameter that indicates whether to print relevant information in console (used only if it is necessary to re-collect data when trying to fetch the data).

Parameters

- `data_dir` (*str* or *None*) – name of a folder where the pickle file is to be saved
- `verbose` (*bool* or *int*) – whether to print relevant information in console

Returns

whether to print relevant information in console when collecting data

Return type

`bool` or `int`

Example:

```
>>> from pyrcs.utils import collect_in_fetch_verbose

>>> collect_in_fetch_verbose(data_dir="data", verbose=True)
False
```

fetch_all_verbose

`pyrcs.utils.fetch_all_verbose(data_dir, verbose)`

Create a new parameter that indicates whether to print relevant information in console (used only when trying to fetch all data of a cluster).

Parameters

- **data_dir** (*str* or *None*) – name of a folder where the pickle file is to be saved
- **verbose** (*bool* or *int*) – whether to print relevant information in console

Returns

whether to print relevant information in console when collecting data

Return type

bool or int

Example:

```
>>> from pyrcs.utils import fetch_all_verbose

>>> fetch_all_verbose(data_dir="data", verbose=True)
False
```

Print messages

<code>confirm_msg(data_name)</code>	Create a confirmation message (for data collection).
<code>print_collect_msg(data_name, verbose, ..., end))</code>	Print a message about the status of collecting data.
<code>print_conn_err([verbose])</code>	Print a message about unsuccessful attempts to establish a connection to the Internet.
<code>format_err_msg(e)</code>	Format an error message.
<code>print_inst_conn_err([update, verbose, e])</code>	Print a message about unsuccessful attempts to establish a connection to the Internet (for an instance of a class).
<code>print_void_msg(data_name, verbose)</code>	Print a message about the status of collecting data (only when the data collection process fails).

confirm_msg

`pyrcs.utils.confirm_msg(data_name)`

Create a confirmation message (for data collection).

Parameters

data_name (*str*) – name of data, e.g. "Railway Codes"

Returns

a confirmation message

Return type

str

Example:

```
>>> from pyrcs.utils import confirm_msg

>>> msg = confirm_msg(data_name="Railway Codes")
>>> print(msg)
To collect data of Railway Codes
?
```

print_collect_msg

`pyrcs.utils.print_collect_msg(data_name, verbose, confirmation_required, end=' ... ')`

Print a message about the status of collecting data.

Parameters

- **data_name** (*str*) – name of the data being collected
- **verbose** (*bool* or *int*) – whether to print relevant information in console
- **confirmation_required** (*bool*) – whether to confirm before proceeding
- **end** (*str*) – string appended after the last value, defaults to " ... ".

Example:

```
>>> from pyrcs.utils import print_collect_msg

>>> print_collect_msg("Railway Codes", verbose=2, confirmation_required=False)
Collecting the data of "Railway Codes" ...
```


print_conn_err

`pyrcs.utils.print_conn_err(verbose=False)`

Print a message about unsuccessful attempts to establish a connection to the Internet.

Parameters

verbose (*bool* or *int*) – whether to print relevant information in console, defaults to `False`

Example:

```
>>> from pyrcs.utils import print_conn_err

>>> # If Internet connection is ready, nothing would be printed
>>> print_conn_err(verbose=True)
```

format_err_msg

`pyrcs.utils.format_err_msg(e)`

Format an error message.

Parameters

e (*Exception* or *None*) – subclass of `Exception`

Returns

an error message

Return type

`str`

print_inst_conn_err

`pyrcs.utils.print_inst_conn_err(update=False, verbose=False, e=None)`

Print a message about unsuccessful attempts to establish a connection to the Internet (for an instance of a class).

Parameters

- **update** (*bool*) – mostly complies with `update` in a parent function that uses this function, defaults to `False`
- **verbose** (*bool* or *int*) – whether to print relevant information in console, defaults to `False`
- **e** (*Exception* or *None*) – error message

Example:

```
>>> from pyrcs.utils import print_inst_conn_err

>>> print_inst_conn_err(verbose=True)
The Internet connection is not available.
```

print_void_msg

`pyrcs.utils.print_void_msg(data_name, verbose)`

Print a message about the status of collecting data (only when the data collection process fails).

Parameters

- **data_name** (*str*) – name of the data being collected
- **verbose** (*bool* or *int*) – whether to print relevant information in console

Example:

```
>>> from pyrcs.utils import print_void_msg

>>> print_void_msg(data_name="Railway Codes", verbose=True)
No data of "Railway Codes" has been freshly collected.
```

Save and retrieve pre-packed data

<code>init_data_dir</code> (cls, data_dir, category[, cluster])	Specify an initial data directory for (an instance of) a class for a data cluster.
<code>make_file_pathname</code> (cls, data_name[, ext, ...])	Make a pathname for saving data as a file of a certain format.
<code>fetch_location_names_errata</code> ([k, regex, ...])	Create a dictionary for rectifying location names.
<code>save_data_to_file</code> (cls, data, data_name, ext)	Save the collected data as a file, depending on the given parameters.
<code>fetch_data_from_file</code> (cls, method, data_name, ...)	Fetch/load desired data from a backup file, depending on the given parameters.

init_data_dir

`pyrcs.utils.init_data_dir(cls, data_dir, category, cluster=None, **kwargs)`

Specify an initial data directory for (an instance of) a class for a data cluster.

Parameters

- **cls** (*object*) – (an instance of) a class for a certain data cluster
- **data_dir** (*str* or *None*) – name of a folder where the pickle file is to be saved
- **category** (*str*) – name of a data category, e.g. "line-data"
- **cluster** (*str* or *None*) – replacement for `cls.KEY`
- **kwargs** – [optional] parameters of the function `cd_data()`

Returns

pathnames of a default data directory and a current data directory

Return type

tuple[str, os.PathLike[str]]

Example:

```
>>> from pyrcs.utils import init_data_dir
>>> from pyrcs.line_data import Bridges
>>> import os

>>> bridges = Bridges()

>>> dat_dir, current_dat_dir = init_data_dir(bridges, data_dir="data", category="line-data")
>>> os.path.relpath(dat_dir)
'data'
>>> os.path.relpath(current_dat_dir)
'data'
```

make_file_pathname

pyrcs.utils.make_file_pathname(*cls*, *data_name*, *ext*='.pickle', *data_dir*=None)

Make a pathname for saving data as a file of a certain format.

Parameters

- **cls** (*object*) – (an instance of) a class for a certain data cluster
- **data_name** (*str*) – key to the dict-type data of a certain code cluster
- **ext** (*str*) – file extension, defaults to ".pickle"
- **data_dir** (*str or None*) – name of a folder where the data is saved, defaults to None

Returns

a pathname for saving the data

Return type

str

Example:

```
>>> from pyrcs.utils import make_file_pathname
>>> from pyrcs.line_data import Bridges
>>> import os

>>> bridges = Bridges()

>>> example_pathname = make_file_pathname(bridges, data_name="example-data", ext=".pickle")
>>> os.path.relpath(example_pathname)
'pyrcs\data\line-data\bridges\example-data.pickle'
```

fetch_location_names_errata

```
pyrcs.utils.fetch_location_names_errata(k=None, regex=False, as_dataframe=False,
                                         column_name=None)
```

Create a dictionary for rectifying location names.

Parameters

- **k** (*str or int or float or bool or None*) – key of the created dictionary, defaults to None
- **regex** (*bool*) – whether to create a dictionary for replacement based on regular expressions, defaults to False
- **as_dataframe** (*bool*) – whether to return the created dictionary as a pandas.DataFrame, defaults to False
- **column_name** (*str or list or None*) – (if as_dataframe=True) column name of the errata data as a dataframe

Returns

dictionary for rectifying location names

Return type

dict or 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-

save_data_to_file

`pyrcs.utils.save_data_to_file(cls, data, data_name, ext, dump_dir=None, verbose=False, **kwargs)`

Save the collected data as a file, depending on the given parameters.

Parameters

- `cls` (*object*) – (an instance of) a class for a certain data cluster
- `data` (*pandas.DataFrame or list or dict*) – data collected for a certain cluster
- `data_name` (*str*) – key to the dict-type data of a certain cluster
- `ext` (*bool or str*) – whether to save the data as a file, or file extension
- `dump_dir` (*str or None*) – pathname of a directory where the data file is to be dumped, defaults to `None`
- `verbose` (*bool or int*) – whether to print relevant information in console, defaults to `False`
- `kwargs` – [optional] parameters of the function `pyhelpers.store.save_data()`

fetch_data_from_file

`pyrcs.utils.fetch_data_from_file(cls, method, data_name, ext, update, dump_dir, verbose, data_dir=None, save_data_kwargs=None, **kwargs)`

Fetch/load desired data from a backup file, depending on the given parameters.

Parameters

- `cls` (*object*) – (an instance of) a class for a certain data cluster
- `method` (*str*) – name of a method of the `cls`, which is used for collecting the data
- `data_name` (*str*) – key to the dict-type data of a certain cluster
- `ext` (*bool or str*) – whether to save the data as a file, or file extension
- `update` (*bool*) – whether to do an update check (for the package data), defaults to `False`
- `dump_dir` (*str or os.PathLike[str] or None*) – pathname of a directory where the data file is to be dumped, defaults to `None`
- `verbose` (*bool or int*) – whether to print relevant information in console
- `data_dir` (*str or os.PathLike[str] or None*) – pathname of a directory where the data is fetched, defaults to `None`
- `save_data_kwargs` (*dict or None*) – equivalent of `kwargs` used by the function `pyrcs.utils.save_data_to_file()`, defaults to `None`
- `kwargs` (*Any*) – [optional] parameters of the `cls.``method``` being called

Returns

data fetched for the desired cluster

Return type

dict or None

Chapter 4

License

PyRCS is licensed under [GNU General Public License v3](#) or later (GPLv3+).

Chapter 5

Use of data

For the use of the data pre-packed with, and collected by, PyRCS, please refer to this link: <http://www.railwaycodes.org.uk/misc/contributing.shtm>

Chapter 6

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.

Chapter 7

Tutorial

To demonstrate how PyRCS works, this brief tutorial provides a quick guide with examples of getting the following three categories of codes, which are frequently used in the railway system in the UK:

- [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).

7.1 Location identifiers

The location identifiers, including CRS, NLC, TIPLOC and STANOX codes, are categorised as [line data](#) on the [Railway Codes](#) website. To get these codes via PyRCS, we can use the class `LocationIdentifiers`, which is contained in the sub-package `line_data`. Let's firstly import the class and create an instance:

```
>>> 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'
```

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

```
>>> from pyrcs.collector import LineData # from pyrcs import LineData

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

>>> lid.NAME == lid_.NAME
True
```

Note:

- The instance `ld` refers to all classes under the category of [line data](#).
- Here `lid_` is equivalent to `lid`.

7.1.1 Location identifiers given a specific initial letter

Now we can get the codes (in a `pandas.DataFrame` type) for all locations beginning with a given letter, by using the method `LocationIdentifiers.collect_codes_by_initial()`. For example, to get the codes for locations whose names begin with 'A' (or 'a'):

```
>>> loc_a_codes = lid.collect_codes_by_initial(initial='a') # The input is case-insensitive
>>> type(loc_a_codes)
dict
>>> list(loc_a_codes.keys())
['A', 'Additional notes', 'Last updated date']
```

As demonstrated above, `loc_a_codes` is a [dictionary](#) (in `dict` type), which has the following *keys*:

- 'A'
- 'Additional 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 presented on the web page [Locations beginning A](#));
- `loc_a_codes['Additional notes']` - Additional information on the web page (if available);
- `loc_a_codes['Last updated date']` - The date when the web page [Locations beginning A](#) 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.head()
   Location CRS  ... STANME_Note STANOX_Note
0           A1  ...
1   A463 Traded In  ...
2  A483 Road Scheme Supervisors Closed  ...
3           Aachen  ...
4   AA Holidays S524  ...

[5 rows x 12 columns]

>>> print("Last updated date: {}".format(loc_a_codes_dat['Last updated date']))
```

7.1.2 All available location identifiers

In addition to the 'A' group of locations, we can use the method `LocationIdentifiers.fetch_codes()` to get the codes of all locations (with the initial letters ranging from 'A' to 'Z') available in this category:

```
>>> loc_codes = lid.fetch_codes()
>>> type(loc_codes)
dict
>>> list(loc_codes.keys())
['LocationID', 'Other systems', 'Additional notes', 'Last updated date']
```

`loc_codes` is also in a [dictionary](#), of which the *keys* are as follows:

- 'LocationID'
- 'Other systems'
- 'Additional notes'
- 'Latest update date'

The corresponding *values* are:

- `loc_codes['LocationID']` - CRS, NLC, TIPLOC and STANOX codes for all locations available on the relevant web pages ranging from 'A' to 'Z';
- `loc_codes['Other systems']` - Relevant codes of the [Other systems](#);
- `loc_codes['Additional notes']` - Additional notes and information (if available);
- `loc_codes['Latest update date']` - The latest 'Last updated date' among all initial-specific codes.

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

```
>>> lid.KEY
'LocationID'

>>> loc_codes_dat = loc_codes[lid.KEY] # loc_codes['LocationID']
>>> type(loc_codes_dat)
pandas.core.frame.DataFrame
>>> loc_codes_dat.head()

   Location CRS  ... STANME_Note STANOX_Note
0              A1  ...
1      A463 Traded In  ...
2  A483 Road Scheme Supervisors Closed  ...
3              Aachen  ...
4      AA Holidays S524  ...

[5 rows x 12 columns]

>>> # Relevant codes of the 'Other systems'
>>> lid.KEY_TO_OTHER_SYSTEMS
'Other systems'
>>> os_codes_dat = loc_codes[lid.KEY_TO_OTHER_SYSTEMS]
>>> type(os_codes_dat)
collections.defaultdict
```

(continues on next page)

(continued from previous page)

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

>>> # Take 'Crossrail' as an example
>>> 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

```

[5 rows x 5 columns]

```

7.2 ELRs and mileages

Engineer's Line References (ELRs) are also frequently seen among various data in Britain's railway system. To get the codes of ELRs (and their associated mileage files), we can use the class `ELRMileages`:

```
>>> 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'
```

7.2.1 Engineer's Line References (ELRs)

Similar to the location identifiers, the codes of ELRs on the [Railway Codes](#) website are also alphabetically arranged given their initial letters. We can use the method `ELRMileages.collect_elr_by_initial()` to get the data of ELRs which begin with a specific initial letter. Let's take 'A' as an example:

```
>>> elrs_a_codes = em.collect_elr_by_initial(initial='a') # Data of ELRs beginning with 'A'
>>> type(elrs_a_codes)
dict
>>> list(elrs_a_codes.keys())
['A', 'Last updated date']
```

`elrs_a_codes` is a [dictionary](#) and has the following *keys*:

- 'A'
- 'Last updated date'

The corresponding *values* are:

- `elrs_a_codes['A']` - Data of ELRs that begin with 'A' (referring to the table presented on the web page [ELRs beginning with A](#));
- `elrs_a_codes['Last updated date']` - The date when the web page [ELRs beginning with A](#) was last updated.

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

```
>>> 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]

>>> print("Last updated date: {}".format(elrs_a_codes['Last updated date']))
```

To get the data of all ELRs (with the initial letters ranging from 'A' to 'Z') available in this category, we can use the method `ELRMileages.fetch_elr()`:

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

In like manner, `elrs_codes` is also a [dictionary](#), of which the *keys* are:

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

The corresponding *values* are:

- `elrs_codes['ELRs and mileages']` - Codes of all available ELRs (with the initial letters ranging from 'A' to 'Z');
- `elrs_codes['Latest update date']` - The latest 'Last updated date' among all the initial-specific codes.

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

```
>>> elrs_codes_dat = elrs_codes[em.KEY]
>>> type(elrs_codes_dat)
pandas.core.frame.DataFrame
>>> elrs_codes_dat.head()
```

(continues on next page)

(continued from previous page)

```

    ELR  ...      Notes
0  AAL  ...      Now NAJ3
1  AAM  ...  Formerly AML
2  AAV  ...
3  ABB  ...      Now AHB
4  ABB  ...

[5 rows x 5 columns]
```

7.2.2 Mileage file of a given ELR

Further to the codes of ELRs, each ELR is associated with a mileage file, which specifies the major mileages for the ELR. To get the mileage data, we can use the method `ELRMileages.fetch_mileage_file()`.

For example, let's try 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']
```

As demonstrated above, `amm_mileage_file` is a [dictionary](#) and has the following *keys*:

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

The corresponding *values* are:

- `amm_mileage_file['ELR']` - The given ELR, which, in this example, is 'AAM';
- `amm_mileage_file['Line']` - Name of the line associated with the given ELR;
- `amm_mileage_file['Sub-Line']` - Name of the sub line (if any) associated with the given ELR;
- `amm_mileage_file['Mileage']` - Major mileages for the given ELR;
- `amm_mileage_file['Notes']` - Additional information/notes (if any).

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

```

>>> amm_mileage_file['Line']
'Ashchurch and Malvern Line'

>>> amm_mileage_file['Mileage'].head()
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 rows x 11 columns]
```

7.3 Railway station data

The [railway station data](#) (including the station name, ELR, mileage, status, owner, operator, degrees of longitude and latitude, and grid reference) is categorised as one of the [other assets](#) on the [Railway Codes](#) website. To deal with data in this category, PyRCS offers a sub-package `other_assets`, from which we can use the contained class `Stations` to get the [railway station data](#):

Now let's import the class and create an instance of it:

```
>>> 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'
```

Note:

- Alternatively, the instance `stn` can also be defined through the class `OtherAssets`, which contains all classes under the category of [other assets](#) (see below).

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

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

>>> stn.NAME == stn_.NAME
True
```

Note:

- The instances `stn_` and `stn` are of the same class `Stations`.
-

7.3.1 Railway station locations given a specific initial letter

To get the location data of railway stations whose names start with a given letter, say 'A', we can use the method `Stations.collect_locations_by_initial()`:

```
>>> stn_loc_a_codes = stn.collect_locations_by_initial(initial='a')
>>> type(stn_loc_a_codes)
dict
>>> list(stn_loc_a_codes.keys())
['A', 'Last updated date']
```

As demonstrated above, the dictionary `stn_loc_a_codes` include the following *keys*:

- 'A'
- 'Last updated date'

The corresponding *values* are:

- `stn_loc_a_codes['A']` - Mileages, operators and grid coordinates of railway stations whose names begin with 'A' (referring to the table presented on the web page of [Stations beginning with A](#));
- `stn_loc_a_codes['Last updated date']` - The date when the web page [Stations beginning with A](#) was last updated.

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

```
>>> 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()
```

(continues on next page)

(continued from previous page)

	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

```
>>> print("Last updated date: {}".format(stn_loc_a_codes['Last updated date']))
```

7.3.2 All available railway station locations

To get the location data of all railway stations (with the initial letters ranging from 'A' to 'Z') available in this category, we can use the method `Stations.fetch_locations()`:

```
>>> 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` include the following *keys*:

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

The corresponding *values* are:

- `stn_loc_codes['Mileages, operators and grid coordinates']` - Location data of all railway stations available on the relevant web pages ranging from 'A' to 'Z';
- `stn_loc_codes['Latest update date']` - The latest 'Last updated date' among all initial-specific codes.

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

```
>>> 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',
```

(continues on next page)

(continued from previous page)

```
'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

>>> print("Last updated date: {}".format(stn_loc_data['Last updated date']))
```

This is the end of the *tutorial*.

Any issues regarding the use of the package are all welcome and should be logged/reported onto the [Bug Tracker](#).

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

Python Module Index

P

- `pyrcs`, 3
- `pyrcs.collector`, 132
- `pyrcs.converter`, 124
- `pyrcs.line_data`, 3
 - `pyrcs.line_data.bridge`, 55
 - `pyrcs.line_data.elec`, 14
 - `pyrcs.line_data.elr_mileage`, 4
 - `pyrcs.line_data.line_name`, 47
 - `pyrcs.line_data.loc_id`, 28
 - `pyrcs.line_data.lor_code`, 39
 - `pyrcs.line_data.trk_diagr`, 51
- `pyrcs.other_assets`, 58
 - `pyrcs.other_assets.depot`, 89
 - `pyrcs.other_assets.feature`, 101
 - `pyrcs.other_assets.sig_box`, 59
 - `pyrcs.other_assets.station`, 82
 - `pyrcs.other_assets.tunnel`, 73
 - `pyrcs.other_assets.viaduct`, 78
- `pyrcs.parser`, 114
- `pyrcs.utils`, 137

Index

B

Bridges (class in `pyrcs.line_data.bridge`), 55

C

`check_row_spans()` (`pyrcs.other_assets.station.Stations` static method), 84

`cleanse_mult_alt_codes()`
(`pyrcs.line_data.loc_id.LocationIdentifiers` static method), 31

`collect_1950_system_codes()`
(`pyrcs.other_assets.depot.Depots` method), 92

`collect_bell_codes()`
(`pyrcs.other_assets.sig_box.SignalBoxes` method), 62

`collect_buzzer_codes()` (`pyrcs.other_assets.feature.Features` method), 104

`collect_catalogue()`
(`pyrcs.line_data.trk_diagr.TrackDiagrams` method), 53

`collect_codes()` (`pyrcs.line_data.bridge.Bridges` method), 56

`collect_codes()` (`pyrcs.line_data.line_name.LineNames` method), 49

`collect_codes_by_initial()`
(`pyrcs.line_data.loc_id.LocationIdentifiers` method), 31

`collect_codes_by_page()` (`pyrcs.other_assets.tunnel.Tunnels` method), 75

`collect_codes_by_page()`
(`pyrcs.other_assets.viaduct.Viaducts` method), 80

`collect_codes_by_prefix()` (`pyrcs.line_data.lor_code.LOR` method), 41

`collect_elr_by_initial()`
(`pyrcs.line_data.elr_mileage.ELRMileages` method), 6

`collect_elr_lor_converter()`
(`pyrcs.line_data.lor_code.LOR` method), 43

`collect_etz_codes()` (`pyrcs.line_data.elec.Electrification` method), 17

`collect_explanatory_note()`
(`pyrcs.line_data.loc_id.LocationIdentifiers` method), 32

`collect_gwr_codes()` (`pyrcs.other_assets.depot.Depots` method), 93

`collect_habds_and_wilds()`
(`pyrcs.other_assets.feature.Features` method), 105

`collect_in_fetch_verbose()` (in module `pyrcs.utils`), 139

`collect_indep_lines_codes()`
(`pyrcs.line_data.elec.Electrification` method), 18

`collect_ireland_codes()`
(`pyrcs.other_assets.sig_box.SignalBoxes` method), 63

`collect_locations_by_initial()`
(`pyrcs.other_assets.station.Stations` method), 85

`collect_mileage_file()`
(`pyrcs.line_data.elr_mileage.ELRMileages` method), 7

`collect_national_network_codes()`
(`pyrcs.line_data.elec.Electrification` method), 20

`collect_non_national_rail_codes()`
(`pyrcs.other_assets.sig_box.SignalBoxes` method), 64

`collect_ohns_codes()` (`pyrcs.line_data.elec.Electrification` method), 21

`collect_other_systems_codes()`
(`pyrcs.line_data.loc_id.LocationIdentifiers` method), 33

`collect_pre_tops_codes()` (`pyrcs.other_assets.depot.Depots` method), 94

`collect_prefix_codes()`
(`pyrcs.other_assets.sig_box.SignalBoxes` method), 65

`collect_telegraph_codes()`
(`pyrcs.other_assets.feature.Features` method), 106

`collect_tops_codes()` (`pyrcs.other_assets.depot.Depots` method), 95

`collect_water_troughs()`
(`pyrcs.other_assets.feature.Features` method), 107

`collect_wr_mas_dates()`
(`pyrcs.other_assets.sig_box.SignalBoxes` method), 66

`confirm_msg()` (in module `pyrcs.utils`), 141

D

Depots (class in `pyrcs.other_assets.depot`), 90

E

Electrification (class in `pyrcs.line_data.elec`), 14

ELRMileages (class in `pyrcs.line_data.elr_mileage`), 4

F

Features (class in `pyrcs.other_assets.feature`), 101

`fetch_1950_system_codes()`
(`pyrcs.other_assets.depot.Depots` method), 96

`fetch_all_verbose()` (in module `pyrcs.utils`), 140

`fetch_bell_codes()` (`pyrcs.other_assets.sig_box.SignalBoxes` method), 67

`fetch_buzzer_codes()` (`pyrcs.other_assets.feature.Features` method), 108

`fetch_catalogue()` (`pyrcs.line_data.trk_diagr.TrackDiagrams` method), 54

`fetch_codes()` (`pyrcs.line_data.bridge.Bridges` method), 57

`fetch_codes()` (`pyrcs.line_data.elec.Electrification` method), 22

fetch_codes() (pyrcs.line_data.line_name.LineNames method), 50
 fetch_codes() (pyrcs.line_data.loc_id.LocationIdentifiers method), 34
 fetch_codes() (pyrcs.line_data.lor_code.LOR method), 44
 fetch_codes() (pyrcs.other_assets.depot.Depots method), 97
 fetch_codes() (pyrcs.other_assets.feature.Features method), 109
 fetch_codes() (pyrcs.other_assets.tunnel.Tunnels method), 76
 fetch_codes() (pyrcs.other_assets.viaduct.Viaducts method), 81
 fetch_data_from_file() (in module pyrcs.utils), 146
 fetch_elr() (pyrcs.line_data.elr_mileage.ELRMileages method), 9
 fetch_elr_lor_converter() (pyrcs.line_data.lor_code.LOR method), 45
 fetch_etz_codes() (pyrcs.line_data.elec.Electrification method), 23
 fetch_explanatory_note() (pyrcs.line_data.loc_id.LocationIdentifiers method), 35
 fetch_gwr_codes() (pyrcs.other_assets.depot.Depots method), 98
 fetch_habds_and_wilds() (pyrcs.other_assets.feature.Features method), 111
 fetch_indep_lines_codes() (pyrcs.line_data.elec.Electrification method), 24
 fetch_ireland_codes() (pyrcs.other_assets.sig_box.SignalBoxes method), 68
 fetch_location_names_errata() (in module pyrcs.utils), 145
 fetch_locations() (pyrcs.other_assets.station.Stations method), 86
 fetch_mileage_file() (pyrcs.line_data.elr_mileage.ELRMileages method), 10
 fetch_national_network_codes() (pyrcs.line_data.elec.Electrification method), 25
 fetch_non_national_rail_codes() (pyrcs.other_assets.sig_box.SignalBoxes method), 69
 fetch_ohns_codes() (pyrcs.line_data.elec.Electrification method), 26
 fetch_other_systems_codes() (pyrcs.line_data.loc_id.LocationIdentifiers method), 36
 fetch_pre_tops_codes() (pyrcs.other_assets.depot.Depots method), 99
 fetch_prefix_codes() (pyrcs.other_assets.sig_box.SignalBoxes method), 70
 fetch_telegraph_codes() (pyrcs.other_assets.feature.Features method), 112
 fetch_tops_codes() (pyrcs.other_assets.depot.Depots method), 100
 fetch_water_troughs() (pyrcs.other_assets.feature.Features method), 113
 fetch_wr_mas_dates() (pyrcs.other_assets.sig_box.SignalBoxes method), 71
 fix_mileage() (in module pyrcs.converter), 125
 fix_stanox() (in module pyrcs.converter), 131
 format_err_msg() (in module pyrcs.utils), 142

G

get_catalogue() (in module pyrcs.parser), 119
 get_catalogue() (pyrcs.other_assets.station.Stations method), 87
 get_category_menu() (in module pyrcs.parser), 120
 get_code_notes() (pyrcs.line_data.loc_id.LocationIdentifiers method), 37
 get_conn_mileages() (pyrcs.line_data.elr_mileage.ELRMileages method), 12
 get_financial_year() (in module pyrcs.parser), 119
 get_heading_text() (in module pyrcs.parser), 122
 get_hypertext() (in module pyrcs.parser), 123
 get_indep_line_catalogue() (pyrcs.line_data.elec.Electrification method), 27
 get_introduction() (in module pyrcs.parser), 123
 get_keys_to_prefixes() (pyrcs.line_data.lor_code.LOR method), 46
 get_last_updated_date() (in module pyrcs.parser), 118
 get_page_catalogue() (in module pyrcs.parser), 121
 get_page_urls() (pyrcs.line_data.lor_code.LOR method), 47
 get_site_map() (in module pyrcs.parser), 117

I

init_data_dir() (in module pyrcs.utils), 143
 is_home_connectable() (in module pyrcs.utils), 137
 is_str_float() (in module pyrcs.utils), 138

K

KEY (pyrcs.line_data.bridge.Bridges attribute), 56
 KEY (pyrcs.line_data.elec.Electrification attribute), 15
 KEY (pyrcs.line_data.elr_mileage.ELRMileages attribute), 5
 KEY (pyrcs.line_data.line_name.LineNames attribute), 48
 KEY (pyrcs.line_data.loc_id.LocationIdentifiers attribute), 29
 KEY (pyrcs.line_data.lor_code.LOR attribute), 40
 KEY (pyrcs.line_data.trk_diagr.TrackDiagrams attribute), 52
 KEY (pyrcs.other_assets.depot.Depots attribute), 91
 KEY (pyrcs.other_assets.feature.Features attribute), 102
 KEY (pyrcs.other_assets.sig_box.SignalBoxes attribute), 60
 KEY (pyrcs.other_assets.station.Stations attribute), 83
 KEY (pyrcs.other_assets.tunnel.Tunnels attribute), 74
 KEY (pyrcs.other_assets.viaduct.Viaducts attribute), 80
 KEY_ELC (pyrcs.line_data.lor_code.LOR attribute), 40
 KEY_P (pyrcs.line_data.lor_code.LOR attribute), 40
 KEY_TO_1950_SYSTEM (pyrcs.other_assets.depot.Depots attribute), 91
 KEY_TO_ADDITIONAL_NOTES (pyrcs.line_data.loc_id.LocationIdentifiers attribute), 29
 KEY_TO_BELL_CODES (pyrcs.other_assets.sig_box.SignalBoxes attribute), 60
 KEY_TO_BUZZER (pyrcs.other_assets.feature.Features attribute), 102
 KEY_TO_ENERGY_TARIFF_ZONES (pyrcs.line_data.elec.Electrification attribute), 15
 KEY_TO_GWR (pyrcs.other_assets.depot.Depots attribute), 91
 KEY_TO_HABD_WILD (pyrcs.other_assets.feature.Features attribute), 102
 KEY_TO_INDEPENDENT_LINES (pyrcs.line_data.elec.Electrification attribute), 15

KEY_TO_IRELAND (*pyrcs.other_assets.sig_box.SignalBoxes attribute*), 60

KEY_TO_LAST_UPDATED_DATE (*pyrcs.line_data.bridge.Bridges attribute*), 56

KEY_TO_LAST_UPDATED_DATE (*pyrcs.line_data.elec.Electrification attribute*), 15

KEY_TO_LAST_UPDATED_DATE (*pyrcs.line_data.elr_mileage.ELRMileages attribute*), 5

KEY_TO_LAST_UPDATED_DATE (*pyrcs.line_data.line_name.LineNames attribute*), 48

KEY_TO_LAST_UPDATED_DATE (*pyrcs.line_data.loc_id.LocationIdentifiers attribute*), 29

KEY_TO_LAST_UPDATED_DATE (*pyrcs.line_data.lor_code.LOR attribute*), 41

KEY_TO_LAST_UPDATED_DATE (*pyrcs.line_data.trk_diagr.TrackDiagrams attribute*), 52

KEY_TO_LAST_UPDATED_DATE (*pyrcs.other_assets.depot.Depots attribute*), 91

KEY_TO_LAST_UPDATED_DATE (*pyrcs.other_assets.feature.Features attribute*), 103

KEY_TO_LAST_UPDATED_DATE (*pyrcs.other_assets.sig_box.SignalBoxes attribute*), 60

KEY_TO_LAST_UPDATED_DATE (*pyrcs.other_assets.station.Stations attribute*), 83

KEY_TO_LAST_UPDATED_DATE (*pyrcs.other_assets.tunnel.Tunnels attribute*), 74

KEY_TO_LAST_UPDATED_DATE (*pyrcs.other_assets.viaduct.Viaducts attribute*), 80

KEY_TO_MSCEM (*pyrcs.line_data.loc_id.LocationIdentifiers attribute*), 29

KEY_TO_NATIONAL_NETWORK (*pyrcs.line_data.elec.Electrification attribute*), 16

KEY_TO_NON_NATIONAL_RAIL (*pyrcs.other_assets.sig_box.SignalBoxes attribute*), 60

KEY_TO_OHNS (*pyrcs.line_data.elec.Electrification attribute*), 16

KEY_TO_OTHER_SYSTEMS (*pyrcs.line_data.loc_id.LocationIdentifiers attribute*), 30

KEY_TO_PRE_TOPS (*pyrcs.other_assets.depot.Depots attribute*), 91

KEY_TO_STN (*pyrcs.other_assets.station.Stations attribute*), 84

KEY_TO_TELEGRAPH (*pyrcs.other_assets.feature.Features attribute*), 103

KEY_TO_TOPS (*pyrcs.other_assets.depot.Depots attribute*), 91

KEY_TO_TROUGH (*pyrcs.other_assets.feature.Features attribute*), 103

KEY_TO_WRMASD (*pyrcs.other_assets.sig_box.SignalBoxes attribute*), 61

kilometer_to_yard() (*in module pyrcs.converter*), 131

L

LineData (*class in pyrcs.collector*), 132

LineNames (*class in pyrcs.line_data.line_name*), 47

LocationIdentifiers (*class in pyrcs.line_data.loc_id*), 28

LOR (*class in pyrcs.line_data.lor_code*), 39

M

make_file_pathname() (*in module pyrcs.utils*), 144

make_xref_dict() (*pyrcs.line_data.loc_id.LocationIdentifiers method*), 37

mile_chain_to_mileage() (*in module pyrcs.converter*), 127

mile_yard_to_mileage() (*in module pyrcs.converter*), 128

mileage_num_to_str() (*in module pyrcs.converter*), 129

mileage_str_to_num() (*in module pyrcs.converter*), 129

mileage_to_mile_chain() (*in module pyrcs.converter*), 127

mileage_to_yard() (*in module pyrcs.converter*), 126

module

- pyrcs, 3
- pyrcs.collector, 132
- pyrcs.converter, 124
- pyrcs.line_data, 3
- pyrcs.line_data.bridge, 55
- pyrcs.line_data.elec, 14
- pyrcs.line_data.elr_mileage, 4
- pyrcs.line_data.line_name, 47
- pyrcs.line_data.loc_id, 28
- pyrcs.line_data.lor_code, 39
- pyrcs.line_data.trk_diagr, 51
- pyrcs.other_assets, 58
- pyrcs.other_assets.depot, 89
- pyrcs.other_assets.feature, 101
- pyrcs.other_assets.sig_box, 59
- pyrcs.other_assets.station, 82
- pyrcs.other_assets.tunnel, 73
- pyrcs.other_assets.viaduct, 78
- pyrcs.parser, 114
- pyrcs.utils, 137

N

NAME (*pyrcs.collector.LineData attribute*), 133

NAME (*pyrcs.collector.OtherAssets attribute*), 136

NAME (*pyrcs.line_data.bridge.Bridges attribute*), 56

NAME (*pyrcs.line_data.elec.Electrification attribute*), 16

NAME (*pyrcs.line_data.elr_mileage.ELRMileages attribute*), 5

NAME (*pyrcs.line_data.line_name.LineNames attribute*), 49

NAME (*pyrcs.line_data.loc_id.LocationIdentifiers attribute*), 30

NAME (*pyrcs.line_data.lor_code.LOR attribute*), 41

NAME (*pyrcs.line_data.trk_diagr.TrackDiagrams attribute*), 52

NAME (*pyrcs.other_assets.depot.Depots attribute*), 91

NAME (*pyrcs.other_assets.feature.Features attribute*), 103

NAME (*pyrcs.other_assets.sig_box.SignalBoxes attribute*), 61

NAME (*pyrcs.other_assets.station.Stations attribute*), 84

NAME (*pyrcs.other_assets.tunnel.Tunnels attribute*), 74

NAME (*pyrcs.other_assets.viaduct.Viaducts attribute*), 80

O

OtherAssets (*class in pyrcs.collector*), 134

P

parse_coordinates_columns() (*pyrcs.other_assets.station.Stations static method*), 88

parse_date() (*in module pyrcs.parser*), 116

parse_elr_mileage_columns() (*pyrcs.other_assets.station.Stations static method*), 88

parse_length() (*pyrcs.other_assets.tunnel.Tunnels static method*), 78

[parse_location_name\(\)](#) (*pyrcs.line_data.loc_id.LocationIdentifiers method*), 38
[parse_owner_and_operator_columns\(\)](#) (*pyrcs.other_assets.station.Stations method*), 89
[parse_stanox_note\(\)](#) (*pyrcs.line_data.loc_id.LocationIdentifiers method*), 39
[parse_station_column\(\)](#) (*pyrcs.other_assets.station.Stations static method*), 89
[parse_table\(\)](#) (*in module pyrcs.parser*), 115
[parse_tr\(\)](#) (*in module pyrcs.parser*), 114
[print_collect_msg\(\)](#) (*in module pyrcs.utils*), 141
[print_conn_err\(\)](#) (*in module pyrcs.utils*), 142
[print_inst_conn_err\(\)](#) (*in module pyrcs.utils*), 142
[print_void_msg\(\)](#) (*in module pyrcs.utils*), 143
[pyrcs](#)
 [module](#), 3
[pyrcs.collector](#)
 [module](#), 132
[pyrcs.converter](#)
 [module](#), 124
[pyrcs.line_data](#)
 [module](#), 3
[pyrcs.line_data.bridge](#)
 [module](#), 55
[pyrcs.line_data.elec](#)
 [module](#), 14
[pyrcs.line_data.elr_mileage](#)
 [module](#), 4
[pyrcs.line_data.line_name](#)
 [module](#), 47
[pyrcs.line_data.loc_id](#)
 [module](#), 28
[pyrcs.line_data.lor_code](#)
 [module](#), 39
[pyrcs.line_data.trk_diagr](#)
 [module](#), 51
[pyrcs.other_assets](#)
 [module](#), 58
[pyrcs.other_assets.depot](#)
 [module](#), 89
[pyrcs.other_assets.feature](#)
 [module](#), 101
[pyrcs.other_assets.sig_box](#)
 [module](#), 59
[pyrcs.other_assets.station](#)
 [module](#), 82
[pyrcs.other_assets.tunnel](#)
 [module](#), 73
[pyrcs.other_assets.viaduct](#)
 [module](#), 78
[pyrcs.parser](#)
 [module](#), 114
[pyrcs.utils](#)
 [module](#), 137

S

[save_data_to_file\(\)](#) (*in module pyrcs.utils*), 146
[search_conn\(\)](#) (*pyrcs.line_data.elr_mileage.ELRMileages static method*), 13

[shift_mileage_by_yard\(\)](#) (*in module pyrcs.converter*), 130
[SHORT_NAME](#) (*pyrcs.line_data.lor_code.LOR attribute*), 41
[SignalBoxes](#) (*class in pyrcs.other_assets.sig_box*), 59
[Stations](#) (*class in pyrcs.other_assets.station*), 82

T

[TrackDiagrams](#) (*class in pyrcs.line_data.trk_diagr*), 51
[Tunnels](#) (*class in pyrcs.other_assets.tunnel*), 73

U

[update\(\)](#) (*pyrcs.collector.LineData method*), 134
[update\(\)](#) (*pyrcs.collector.OtherAssets method*), 136
[URL](#) (*pyrcs.collector.LineData attribute*), 134
[URL](#) (*pyrcs.collector.OtherAssets attribute*), 136
[URL](#) (*pyrcs.line_data.bridge.Bridges attribute*), 56
[URL](#) (*pyrcs.line_data.elec.Electrification attribute*), 16
[URL](#) (*pyrcs.line_data.elr_mileage.ELRMileages attribute*), 5
[URL](#) (*pyrcs.line_data.line_name.LineNames attribute*), 49
[URL](#) (*pyrcs.line_data.loc_id.LocationIdentifiers attribute*), 30
[URL](#) (*pyrcs.line_data.lor_code.LOR attribute*), 41
[URL](#) (*pyrcs.line_data.trk_diagr.TrackDiagrams attribute*), 52
[URL](#) (*pyrcs.other_assets.depot.Depots attribute*), 92
[URL](#) (*pyrcs.other_assets.sig_box.SignalBoxes attribute*), 61
[URL](#) (*pyrcs.other_assets.station.Stations attribute*), 84
[URL](#) (*pyrcs.other_assets.tunnel.Tunnels attribute*), 74
[URL](#) (*pyrcs.other_assets.viaduct.Viaducts attribute*), 80

V

[validate_initial\(\)](#) (*in module pyrcs.utils*), 138
[validate_page_name\(\)](#) (*in module pyrcs.utils*), 139
[Viaducts](#) (*class in pyrcs.other_assets.viaduct*), 79

Y

[yard_to_mileage\(\)](#) (*in module pyrcs.converter*), 125