

---

# **PyRCS Documentation**

***Release 0.2.8***

**Qian Fu**

**Mar 23, 2021**



---

## Contents

---

<b>1</b>	<b>Installation</b>	<b>1</b>
<b>2</b>	<b>Quick start</b>	<b>3</b>
2.1	Get location codes . . . . .	3
2.2	Get ELRs and mileages . . . . .	5
2.3	Get railway stations data . . . . .	6
<b>3</b>	<b>Modules</b>	<b>9</b>
3.1	line_data . . . . .	9
3.2	other_assets . . . . .	35
3.3	updater . . . . .	55
3.4	utils . . . . .	56
<b>4</b>	<b>License</b>	<b>69</b>
<b>5</b>	<b>Use of data</b>	<b>71</b>
<b>6</b>	<b>Acknowledgements</b>	<b>73</b>
	<b>Python Module Index</b>	<b>75</b>
	<b>Index</b>	<b>77</b>



# CHAPTER 1

---

## Installation

---

If you are using a [virtualenv](#), ensure that the virtualenv is activated.

To install the latest release of [pyrcs](#) at [PyPI](#) via [pip](#) on Windows Command Prompt (CMD) or Linux/Unix terminal.

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

If you would like to try the more recent version under development, install it from GitHub

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

To test if [pyrcs](#) is correctly installed, try importing the package from an interpreter shell:

```
>>> import pyrcs
>>> pyrcs.__version__ # Check the current release
0.2.8
```

---

### Note:

- To ensure you get the most recent version, it is always recommended to add `--upgrade` (or `-U`) to `pip install`.
  - [pyrcs](#) has not yet been tested with Python 2. For users who have installed both Python 2 and 3, it would be recommended to replace `pip` with `pip3`. But you are more than welcome to volunteer testing the package with Python 2 and any issues should be logged/reported onto the web page of “[Issues](#)”.
  - For more general instructions, check the web page of “[Installing Packages](#)”.
-



To demonstrate how PyRCS works, this part of the documentation provides a quick guide and a few examples as follows:

- *Get location codes: CRS, NLC, TIPLOC and STANOX*
  - *Location codes for a given initial letter*
  - *All available location codes*
- *Get ELRs and mileages*
  - *ELR codes*
  - *Mileage files*
- *Get railway stations data*

For more details and examples, feel free to check *Modules*.

## 2.1 Get location codes

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

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

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

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

---

**Note:** An alternative way of creating the instance is through `pyrcs.line_data.LineData()`:

```
>>> from pyrcs import LineData

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

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

---

### 2.1.1 For a given initial letter

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

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

`location_codes_a` is a dictionary (in `dict` type), with the following keys:

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

Their corresponding values are

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

### 2.1.2 For all location codes

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

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

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

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

Their corresponding values are

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



## 2.2 Get ELRs and mileages

To get ELRs (Engineer's Line References) and mileages, use the class `pyrcs.line_data.ELRMileages()`:

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

>>> em = ELRMileages()
```

### 2.2.1 Get ELR codes

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

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

The keys of `elrs_a` include:

- 'A'
- 'Last updated date'

Their corresponding values are

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

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

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

The keys of `elrs_data` include:

- 'ELRs'
- 'Latest update date'

Their corresponding values are

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

### 2.2.2 Get mileage files

To get detailed mileage data for a given ELR, for example, [AAM](#), use the method `:ref:.fetch_mileage_file()` `<em-fetch-mileage-file>`, which returns a dictionary as well:

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

The keys of `em_amm` include:

- 'ELR'
- 'Line'
- 'Sub-Line'
- 'AAM'

- 'Notes'

Their corresponding values are

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

## 2.3 Get railway stations data

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

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

---

**Note:** Alternatively, the instance `stn` can also be defined in the following way, where `oa` contains all classes under the category of [other assets](#).

```
>>> from pyrcs import OtherAssets
>>> oa = OtherAssets()
>>> stn = oa.Stations
```

---

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

```
>>> railway_station_data_a = stn.collect_railway_station_data_by_initial('A
↪')
```

The keys of `railway_station_data_a` include:

- 'A'
- 'Last updated date'

The corresponding values are

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

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

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

The keys of `railway_station_data` include:

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

Their corresponding values are

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



### 3.1 line\_data

Classes for collecting line data.

#### 3.1.1 LocationIdentifiers

A class for collecting CRS, NLC, TIPLOC and STANOX codes.

<code>LocationIdentifiers.amendment_to_location_names_dict()</code>	Create a replacement dictionary for location name amendments.
<code>LocationIdentifiers.parse_additional_note_page(...)</code>	Parse addition note page.
<code>LocationIdentifiers.collect_multiple_station_codes_explanatory_note(...)</code>	Collect note about CRS code from source web page.
<code>LocationIdentifiers.fetch_multiple_station_codes_explanatory_note(...)</code>	Fetch multiple station codes explanatory note from local backup.
<code>LocationIdentifiers.collect_other_systems_codes(...)</code>	Collect data of the other systems codes from source web page.
<code>LocationIdentifiers.fetch_other_systems_codes(...)</code>	Fetch data of the other systems codes from local backup.
<code>LocationIdentifiers.collect_location_codes_by_initial(initial)</code>	Collect CRS, NLC, TIPLOC, STANME and STANOX codes for the given initial letter.
<code>LocationIdentifiers.fetch_location_codes(...)</code>	Fetch CRS, NLC, TIPLOC, STANME and STANOX codes from local backup.
<code>LocationIdentifiers.make_location_codes_dictionary(keys)</code>	Make a dict/dataframe for location code data for the given keys

**class** `pyrcs.line_data.LocationIdentifiers` (*data\_dir=None, update=False*)

A class for collecting CRS, NLC, TIPLOC and STANOX codes.

#### Parameters

- **data\_dir** (*str, None*) – name of data directory, defaults to `None`
- **update** (*bool*) – whether to check on update and proceed to update the package data,

defaults to `False`

**Example:**

```
from pyrcs.line_data import LocationIdentifiers

lid = LocationIdentifiers()

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

print(lid.SourceURL)
# http://www.railwaycodes.org.uk/crs/CRS0.shtm
```

**static amendment\_to\_location\_names\_dict()**

Create a replacement dictionary for location name amendments.

**Returns** dictionary of regular-expression amendments to location names

**Return type** `str`

**Example:**

```
from pyrcs.line_data import LocationIdentifiers

lid = LocationIdentifiers()

location_name_amendment_dict = lid.amendment_to_location_names_
    ↪dict()
```

**static parse\_additional\_note\_page(note\_url, parser='lxml')**

Parse addition note page.

**Parameters**

- **note\_url** (*str*) – URL link of the target web page
- **parser** (*str*) – the parser to use for `bs4.BeautifulSoup`, defaults to `'lxml'`

**Returns** parsed texts

**Return type** `list`

**Example:**

```
from pyrcs.line_data import LocationIdentifiers

lid = LocationIdentifiers()

note_url = locid.HomeURL + '/crs/CRS2.shtm'
parser = 'lxml'

parsed_note = lid.parse_additional_note_page(note_url, parser)
```

**collect\_multiple\_station\_codes\_explanatory\_note(confirmation\_required=True, verbose=False)**

Collect note about CRS code from source web page.

**Parameters**

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

**Returns** data of multiple station codes explanatory note

**Return type** dict, None

**Example:**

```
from pyrcs.line_data import LocationIdentifiers

lid = LocationIdentifiers()

confirmation_required = True

explanatory_note = lid.collect_multiple_station_codes_explanatory_
    ↪note(confirmation_required)
# To collect multiple station codes explanatory note? [No]/Yes:
# >? yes

print(explanatory_note)
# {'Last updated date': <date>,
#  'Multiple station codes explanatory note': <codes>,
#  'Notes': <notes>}
```

**fetch\_multiple\_station\_codes\_explanatory\_note** (*update=False, pickle\_it=False,*  
*data\_dir=None, ver-*  
*bose=False)*

Fetch multiple station codes explanatory note from local backup.

**Parameters**

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

**Returns** data of multiple station codes explanatory note

**Return type** dict

**Example:**

```
from pyrcs.line_data import LocationIdentifiers

lid = LocationIdentifiers()

update = False
pickle_it = False
data_dir = None
verbose = True

explanatory_note = lid.fetch_multiple_station_codes_explanatory_
    ↪note(update, pickle_it, data_dir, verbose)

print(explanatory_note)
# {'Last updated date': <date>,
#  'Multiple station codes explanatory note': <codes>,
#  'Notes': <notes>}
```

**collect\_other\_systems\_codes** (*confirmation\_required=True, verbose=False*)

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

**Parameters**

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

**Returns** codes of other systems

**Return type** dict, None

**Example:**

```
from pyrcs.line_data import LocationIdentifiers

lid = LocationIdentifiers()

confirmation_required = True
verbose = True

other_systems_codes = lid.collect_other_systems_codes(confirmation_
→required, verbose)
# To collect additional CRS note? [No]/Yes: >? yes

print(other_systems_codes)
# {<system name>: <codes>,
# ...}
```

**fetch\_other\_systems\_codes** (*update=False, pickle\_it=False, data\_dir=None, ver-*  
*bose=False*)

Fetch data of the other systems codes from local backup.

**Parameters**

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

**Returns** codes of other systems

**Return type** dict

**Example:**

```
from pyrcs.line_data import LocationIdentifiers

lid = LocationIdentifiers()

update = False
pickle_it = False
data_dir = None
verbose = True

other_systems_codes = lid.fetch_other_systems_codes(update, pickle_
→it, data_dir, verbose)
```

(continues on next page)



(continued from previous page)

**collect\_location\_codes\_by\_initial** (*initial*, *update=False*, *verbose=False*)Collect CRS, NLC, TIPLOC, STANME and STANOX codes for the given *initial* letter.**Parameters**

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

**Returns** data of location codes for the given *initial* letter; and date of when the data was last updated**Return type** dict**Example:**

```
from pyrcs.line_data import LocationIdentifiers

lid = LocationIdentifiers()

initial = 'a'
location_codes_a = lid.collect_location_codes_by_initial(initial)

print(location_codes_a)
# {'A': <codes>,
#   'Additional notes': <notes>,
#   'Last updated date': <date>}
```

**fetch\_location\_codes** (*update=False*, *pickle\_it=False*, *data\_dir=None*, *verbose=False*)

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

**Parameters**

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

**Returns** data of location codes and date of when the data was last updated**Return type** dict**Example:**

```
from pyrcs.line_data import LocationIdentifiers

lid = LocationIdentifiers()

update = False
pickle_it = False
data_dir = None
```

(continues on next page)

(continued from previous page)

```
location_codes = lid.fetch_location_codes(update, pickle_it, data_
    ↪dir)

print(location_codes)
# {'Location codes': <codes>,
#   'Other systems': <codes>,
#   'Additional notes': <notes>,
#   'Latest update date': <date>}
```

**make\_location\_codes\_dictionary**(*keys*, *initials=None*, *drop\_duplicates=False*,  
*as\_dict=False*, *main\_key=None*, *save\_it=False*,  
*data\_dir=None*, *update=False*, *verbose=False*)

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

#### Parameters

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

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

**Return type** dict, pandas.DataFrame, *None*

#### Examples:

```
from pyrcs.line_data import LocationIdentifiers

lid = LocationIdentifiers()

drop_duplicates = False
save_it = False
data_dir = None
update = False

keys = 'STANOX'
initials = None
as_dict = False
main_key = None
stanox_dictionary = lid.make_location_codes_dictionary(keys, ↪
    ↪initials, drop_duplicates,
                                                    as_dict, ↪
    ↪main_key, save_it, data_dir,
```

(continues on next page)

(continued from previous page)

```

                                                                    update)
print(stanox_dictionary)

keys = ['STANOX', 'TIPLOC']
initials = 'a'
as_dict = False
main_key = None
stanox_dictionary = lid.make_location_codes_dictionary(keys,
↳initials, drop_duplicates,
                                                                    as_dict,
↳main_key, save_it, data_dir,
                                                                    update)
print(stanox_dictionary)

keys = ['STANOX', 'TIPLOC']
initials = 'b'
as_dict = True
main_key = 'Data'
stanox_dictionary = lid.make_location_codes_dictionary(keys,
↳initials, drop_duplicates,
                                                                    as_dict,
↳main_key, save_it, data_dir,
                                                                    update)
print(stanox_dictionary)

```

### 3.1.2 Electrification

A class for collecting codes associated with British railway overhead electrification installations.

<i>Electrification.</i> <i>collect_codes_for_national_network(...)</i>	Collect OLE section codes for National network from source web page.
<i>Electrification.</i> <i>fetch_codes_for_national_network(...)</i>	Fetch OLE section codes for National network from local backup.
<i>Electrification.</i> <i>get_names_of_independent_lines()</i>	Get names of independent lines.
<i>Electrification.</i> <i>collect_codes_for_independent_lines(...)</i>	Collect OLE section codes for independent lines from source web page.
<i>Electrification.</i> <i>fetch_codes_for_independent_lines(...)</i>	Fetch OLE section codes for independent lines from local backup.
<i>Electrification.</i> <i>collect_codes_for_ohns(...)</i>	Collect codes for overhead line electrification neutral sections (OHNS) from source web page.
<i>Electrification.</i> <i>fetch_codes_for_ohns(...)</i>	Fetch codes for overhead line electrification neutral sections (OHNS) from local backup.
<i>Electrification.</i> <i>collect_codes_for_energy_tariff_zones(...)</i>	Collect OLE section codes for national network energy tariff zones from source web page.
<i>Electrification.</i> <i>fetch_codes_for_energy_tariff_zones(...)</i>	Fetch OLE section codes for national network energy tariff zones from source web page.
<i>Electrification.</i> <i>fetch_electrification_codes(...)</i>	Fetch OLE section codes in the electrification catalogue.

**class** pyrcs.line\_data.**Electrification** (*data\_dir=None, update=False*)

A class for collecting codes associated with British railway overhead electrification installations.

#### Parameters

- **data\_dir** (*str, None*) – name of data directory, defaults to None

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`

Example:

```
from pyrcs.line_data import Electrification

elec = Electrification()

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

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

**collect\_codes\_for\_national\_network** (*confirmation\_required=True, verbose=False*)

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

#### Parameters

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

**Returns** OLE section codes for National network

**Return type** dict, None

Example:

```
from pyrcs.line_data import Electrification

elec = Electrification()

confirmation_required = True

national_network_ole = elec.collect_codes_for_national_
↳ network(confirmation_required)
# To collect section codes for OLE installations: national network?
↳ [No] | Yes:
# >? yes

print(national_network_ole)
# {'National network': <code>,
#   'Last updated date': <date>}
```

**fetch\_codes\_for\_national\_network** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)

Fetch OLE section codes for National network from local backup.

#### Parameters

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

**Returns** OLE section codes for National network

**Return type** dict, None

**Example:**

```
from pyrcs.line_data import Electrification

elec = Electrification()

update = False
pickle_it = False
data_dir = None

national_network_ole = elec.fetch_codes_for_national_
↳network(update, pickle_it, data_dir)

print(national_network_ole)
# {'National network': <code>,
#  'Last updated date': <date>}
```

**get\_names\_of\_independent\_lines()**

Get names of independent lines.

**Returns** a list of independent line names

**Return type** list

**Example:**

```
from pyrcs.line_data import Electrification

elec = Electrification()

line_names = elec.get_names_of_independent_lines()

print(line_names)
# a list of independent line names
```

**collect\_codes\_for\_independent\_lines** (*confirmation\_required=True, verbose=False*)

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

**Parameters**

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

**Returns** OLE section codes for independent lines

**Return type** dict, None

**Example:**

```
from pyrcs.line_data import Electrification

elec = Electrification()

confirmation_required = True

independent_lines_ole = elec.collect_codes_for_independent_
↳lines(confirmation_required)
```

(continues on next page)

(continued from previous page)

```
# To collect section codes for OLE installations: independent_
↪lines? [No]/Yes:
# >? yes

print(independent_lines_ole)
# {'Independent lines': <codes>,
#   'Last updated date': <date>}
```

**fetch\_codes\_for\_independent\_lines** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)  
Fetch OLE section codes for independent lines from local backup.

**Parameters**

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

**Returns** OLE section codes for independent lines

**Return type** dict

**Example:**

```
from pyrcs.line_data import Electrification

elec = Electrification()

update = False
pickle_it = False
data_dir = None

independent_lines_ole = elec.fetch_codes_for_independent_
↪lines(update, pickle_it, data_dir)

print(independent_lines_ole)
# {'Independent lines': <codes>,
#   'Last updated date': <date>}
```

**collect\_codes\_for\_ohns** (*confirmation\_required=True, verbose=False*)  
Collect codes for overhead line electrification neutral sections (OHNS) from source web page.

**Parameters**

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

**Returns** OHNS codes

**Return type** dict, None

**Example:**

```

from pyrcs.line_data import Electrification

elec = Electrification()

confirmation_required = True

ohns_codes = elec.collect_codes_for_ohns(confirmation_required)
# To collect section codes for OLE installations: national network,
# neutral sections? [No]|Yes:
# >? yes

print(ohns_codes)
# {'National network neutral sections': <codes>,
#  'Last updated date': <date>}

```

**fetch\_codes\_for\_ohns** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)  
 Fetch codes for overhead line electrification neutral sections (OHNS) from local backup.

#### Parameters

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

**Returns** OHNS codes

**Return type** dict

**Example:**

```

from pyrcs.line_data import Electrification

elec = Electrification()

update = False
pickle_it = False
data_dir = None

ohns_codes = elec.fetch_codes_for_ohns(update, pickle_it, data_dir)

print(ohns_codes)
# {'National network neutral sections': <codes>,
#  'Last updated date': <date>}

```

**collect\_codes\_for\_energy\_tariff\_zones** (*confirmation\_required=True, verbose=False*)  
 Collect OLE section codes for national network energy tariff zones from source web page.

#### Parameters

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

**Returns** OLE section codes for national network energy tariff zones

**Return type** dict, None

**Example:**

```
from pyrcs.line_data import Electrification

elec = Electrification()

confirmation_required = True

etz_ole = elec.collect_codes_for_energy_tariff_zones(confirmation_
↪required)
# To collect section codes for OLE installations: national network
↪energy tariff zones? [No]|Yes:
# >? yes

print(etz_ole)
# {'National network energy tariff zones': <codes>,
#   'Last updated date': <date>}
```

**fetch\_codes\_for\_energy\_tariff\_zones** (*update=False*, *pickle\_it=False*,  
*data\_dir=None*, *verbose=False*)  
Fetch OLE section codes for national network energy tariff zones from source web page.

#### Parameters

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

**Returns** OLE section codes for national network energy tariff zones

**Return type** dict

**Example:**

```
from pyrcs.line_data import Electrification

elec = Electrification()

update = False
pickle_it = False
data_dir = None

etz_ole = elec.fetch_codes_for_energy_tariff_zones(update, pickle_
↪it, data_dir)

print(etz_ole)
# {'National network energy tariff zones': <codes>,
#   'Last updated date': <date>}
```

**fetch\_electrification\_codes** (*update=False*, *pickle\_it=False*, *data\_dir=None*, *ver-*  
*bose=False*)  
Fetch OLE section codes in the electrification catalogue.

#### Parameters



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

**Returns** section codes for overhead line electrification (OLE) installations

**Return type** dict

**Example:**

```
from pyrcs.line_data import Electrification

elec = Electrification()

update = False
pickle_it = False
data_dir = None

ole_section_codes = elec.fetch_electrification_codes(update,
    pickle_it, data_dir)

print(ole_section_codes)
# {'Electrification': <codes>,
#  'Latest update date': <date>}
```

### 3.1.3 ELRMileages

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

<code>ELRMileages.identify_multiple_measures</code>	Identify the scraped data of mileage file if it has multiple measures and, if so, preprocess it.
<code>ELRMileages.parse_mileage_data</code>	Parse scraped data of mileage file.
<code>ELRMileages.collect_elr_by_initial</code>	Collect Engineer's Line References (ELRs) for the given initial letter from source web page.
<code>ELRMileages.fetch_elr</code>	Fetch ELRs and mileages from local backup.
<code>ELRMileages.collect_mileage_file_by_elr</code>	Collect mileage file for the given ELR from source web page.
<code>ELRMileages.fetch_mileage_file</code>	Fetch mileage file for the given ELR from local backup.
<code>ELRMileages.search_conn</code>	Search for connection between two ELR-and-mileage pairs.
<code>ELRMileages.get_conn_mileages</code>	Get to end and start mileages for StartELR and EndELR, respectively, for the connection point

**class** `pyrcs.line_data.ELRMileages` (*data\_dir=None*, *update=False*)

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

**Parameters**

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

Example:

```
from pyrcs.line_data import ELRMileages

em = ELRMileages()

print(em.Name)
# ELRs and mileages

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

**static identify\_multiple\_measures** (*mileage\_data*)

Identify the scraped data of mileage file if it has multiple measures and, if so, preprocess it.

**Parameters** *mileage\_data* – scraped raw mileage file from source web page

**Type** pandas.DataFrame

**parse\_mileage\_data** (*mileage\_data*)

Parse scraped data of mileage file.

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

**Returns** parsed data of mileage file

**Return type** pandas.DataFrame

**collect\_elr\_by\_initial** (*initial*, *update=False*, *verbose=False*)

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

**Parameters**

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

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

**Return type** dict

Example:

```
from pyrcs.line_data import ELRMileages

em = ELRMileages()

initial = 'a'
update = False

elrs_a = em.collect_elr_by_initial(initial, update)

print(elrs_a)
# {'A': <codes>,
#   'Last updated date': <date>}
```

**fetch\_elr** (*update=False*, *pickle\_it=False*, *data\_dir=None*, *verbose=False*)

Fetch ELRs and mileages from local backup.

**Parameters**

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

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

**Return type** dict

**Example:**

```
from pyrcs.line_data import ELRMileages

em = ELRMileages()

update = False
pickle_it = False
data_dir = None

elrs_data = em.fetch_elr(update, pickle_it, data_dir)

print(elrs_data)
# {'ELRs': <codes>,
#  'Latest update date': <date>}
```

**collect\_mileage\_file\_by\_elr** (*elr*, *parsed=True*, *confirmation\_required=True*,  
*pickle\_it=False*, *verbose=False*)

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

**Parameters**

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

**Returns** mileage file for the given *elr*

**Return type** dict

---

**Note:**

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

---

**Examples:**

```
from pyrcs.line_data import ELRMileages

em = ELRMileages()

parsed = True
confirmation_required = True
pickle_it = False

elr = 'CJD'
mileage_file = em.collect_mileage_file_by_elr(elr, parsed,
    ↳confirmation_required, pickle_it)
# To collect mileage file for "CJD"? [No]|Yes:
# >? yes
print(mileage_file)
# {'ELR': 'CJD',
#  'Line': 'Challoch Junction to Dumfries Line',
#  'Sub-Line': '',
#  'CJD': <codes>,
#  'Notes': <notes>}

elr = 'GAM'
mileage_file = em.collect_mileage_file_by_elr(elr, parsed,
    ↳confirmation_required, pickle_it)
# To collect mileage file of "GAM"? [No]|Yes:
# >? yes
print(mileage_file)
# {'ELR': 'GAM',
#  'Line': 'Gartness Branch (LMS)',
#  'Sub-Line': '',
#  'GAM': <codes>,
#  'Notes': ''}

elr = 'SLD'
mileage_file = em.collect_mileage_file_by_elr(elr, parsed,
    ↳confirmation_required, pickle_it)
# To collect mileage file of "SLD"? [No]|Yes:
# >? yes
print(mileage_file)
# {'ELR': 'SLD',
#  'Line': 'Stainland Branch',
#  'Sub-Line': '',
#  'SLD': <codes>,
#  'Notes': ''}

elr = 'ZZD2'
mileage_file = em.collect_mileage_file_by_elr(elr, parsed,
    ↳confirmation_required, pickle_it)
# To collect mileage file of "ZZD2"? [No]|Yes:
# >? yes
print(mileage_file)
# {'ELR': 'ZZD2',
#  'Line': 'Gartsherrie Freightliner Depot Sidings',
#  'Sub-Line': '',
#  'ZZD2': <codes>,
#  'Notes': ''}

elr = 'WHG?'
```

(continues on next page)

(continued from previous page)

```

mileage_file = em.collect_mileage_file_by_elr(elr, parsed,
↪confirmation_required, pickle_it)
# To collect mileage file of "WHG"? [No]|Yes:
# >? yes
print(mileage_file)
# {'ELR': 'WHG',
#  'Line': 'West Hartlepool Goods Branch',
#  'Sub-Line': '',
#  'WHG': '<codes>',
#  'Notes': ''}

elr = 'ELR'
mileage_file = em.fetch_mileage_file(elr, update, pickle_it, data_
↪dir)
# To collect mileage file of "ELR"? [No]|Yes:
# >? yes
print(mileage_file)
# {'ELR': 'ELR',
#  'Line': 'Maryhill Park Junction to Anniesland Line',
#  'Sub-Line': '',
#  'MLA': '<codes>',
#  'Notes': '<notes>'}

```

**fetch\_mileage\_file** (elr, update=False, pickle\_it=False, data\_dir=None, verbose=False)

Fetch mileage file for the given ELR from local backup.

#### Parameters

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

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

**Return type** dict

**Example:**

```

from pyrcs.line_data import ELRMileages

em = ELRMileages()

update = False
pickle_it = False
data_dir = None

elr = 'MLA'
mileage_file = em.fetch_mileage_file(elr, update, pickle_it, data_
↪dir)

print(mileage_file)
# {'ELR': 'MLA',

```

(continues on next page)

(continued from previous page)

```
# 'Line': 'Maryhill Park Junction to Anniesland Line',
# 'Sub-Line': '',
# 'MLA': <codes>,
# 'Notes': <notes>}
```

**static search\_conn**(start\_elr, start\_em, end\_elr, end\_em)

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

**Parameters**

- **start\_elr**(str) – start ELR
- **start\_em**(pandas.DataFrame) – mileage file of the start ELR
- **end\_elr**(str) – end ELR
- **end\_em**(pandas.DataFrame) – mileage file of the end ELR

**Returns** connection, in the form (<end mileage of the start ELR>, <start mileage of the end ELR>)**Return type** tuple**Example:**

```
from pyrcs.line_data import ELRMileages

em = ELRMileages()

start_elr = 'AAM'
start_mileage_file = em.collect_mileage_file_by_elr(start_elr)
# To collect mileage file of "AAM"? [No]/Yes:
# >? yes
start_em = start_mileage_file['Mileage']

end_elr = 'ANZ'
end_mileage_file = em.collect_mileage_file_by_elr(end_elr)
# To collect mileage file of "ANZ"? [No]/Yes:
# >? yes
end_em = end_mileage_file['Mileage']

start_dest_mileage, end_orig_mileage = em.search_conn(start_elr,
↪start_em, end_elr, end_em)
print(start_dest_mileage)
# 0.0396
print(end_orig_mileage)
# 84.1364
```

**get\_conn\_mileages**(start\_elr, end\_elr, update=False, pickle\_mileage\_file=False, data\_dir=None, verbose=False)

Get to end and start mileages for StartELR and EndELR, respectively, for the connection point

**Parameters**

- **start\_elr**(str) – start ELR
- **end\_elr**(str) – end ELR
- **update**(bool) – whether to check on update and proceed to update the package data, defaults to False
- **pickle\_mileage\_file**(bool) – whether to replace the current mileage file with newly collected data, defaults to False

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

**Returns** connection ELR and mileages between the given *start\_elr* and *end\_elr*

**Return type** tuple

**Example:**

```
from pyrcs.line_data import ELRMileages

em = ELRMileages()

update = False
pickle_mileage_file = False
data_dir = None
verbose = True

start_elr = 'NAY'
end_elr = 'LTN2'
start_dest_mileage, conn_elr, conn_orig_mileage, conn_dest_mileage,
↪ end_orig_mileage = em.get_conn_mileages(start_
↪ elr, end_elr, update, pickle_mileage_file, data_dir)

print(start_dest_mileage)
# 5.1606
print(conn_elr)
# NOL
print(conn_orig_mileage)
# 5.1606
print(conn_dest_mileage)
# 0.0638
print(end_orig_mileage)
# 123.1320

start_elr = 'MAC3'
end_elr = 'DBP1'
start_dest_mileage, conn_elr, conn_orig_mileage, conn_dest_mileage,
↪ end_orig_mileage = em.get_conn_mileages(start_
↪ elr, end_elr, update, pickle_mileage_file, data_dir)
# ''
```

### 3.1.4 LineNames

A class for collecting British railway line names.

<code>LineNames.collect_line_names([...])</code>	Collect data of railway line names from source web page.
<code>LineNames.fetch_line_names([update, ...])</code>	Fetch data of railway line names from local backup.

**class** `pyrcs.line_data.LineNames` (*data\_dir=None*, *update=False*)

A class for collecting British railway line names.

**Parameters**

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

defaults to `False`

**Example:**

```
from pyrcs.line_data import LineNames

ln = LineNames()

print(ln.Name)
# Railway line names

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

**collect\_line\_names** (*confirmation\_required=True, verbose=False*)

Collect data of railway line names from source web page.

**Parameters**

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

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

**Return type** dict, None

**Example:**

```
from pyrcs.line_data import LineNames

ln = LineNames()

confirmation_required = True

line_names_data = ln.collect_line_names(confirmation_required)
# To collect British railway line names? [No]/Yes:
# >? yes

print(line_names_data)
# {'Line names': <code>,
#  'Last updated date': <date>}
```

**fetch\_line\_names** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)

Fetch data of railway line names from local backup.

**Parameters**

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

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

**Return type** dict

**Example:**



```

from pyrcs.line_data import LineNames

ln = LineNames()

update = False
pickle_it = False
data_dir = None

line_names_data = ln.fetch_line_names(update, pickle_it, data_dir)

print(line_names_data)
# {'Line names': <code>,
#  'Last updated date': <date>}

```

### 3.1.5 LOR

A class for collecting line of route (LOR) codes.

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

**class** `pyrcs.line_data.LOR` (*data\_dir=None, update=False*)

A class for collecting line of route (LOR) codes.

#### Parameters

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

#### Example:

```

from pyrcs.line_data import LOR

lor = LOR()

print(lor.Name)
# Line of Route (LOR/PRIDE) codes

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

```

**get\_keys\_to\_prefixes** (*prefixes\_only=True, update=False, verbose=False*)

Get key to LOR code prefixes.

#### Parameters

- **prefixes\_only** (*bool*) – whether to get only prefixes, defaults to True

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

**Returns** keys to LOR code prefixes

**Return type** list, dict

**Examples:**

```
from pyrcs.line_data import LOR

lor = LOR()

prefixes_only = True
keys_to_prefixes = lor.get_keys_to_prefixes(prefixes_only)
print(keys_to_prefixes)
# ['CY', 'EA', 'GW', 'LN', 'MD', 'NW', 'NZ', 'SC', 'SO', 'SW', 'XR
→']

prefixes_only = False
keys_to_prefixes = lor.get_keys_to_prefixes(prefixes_only)
print(keys_to_prefixes)
# {'Key to prefixes': <data frame of keys to prefixes>,
#  'Last update date': <date>}
```

**get\_lor\_page\_urls** (*update=False, verbose=False*)

Get URLs to LOR codes with different prefixes.

**Parameters**

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

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

**Return type** list

**Example:**

```
from pyrcs.line_data import LOR

lor = LOR()

lor_page_urls = lor.get_lor_page_urls()
print(lor_page_urls)
# <a list of URLs>
```

**update\_catalogue** (*confirmation\_required=True, verbose=False*)

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

**Parameters**

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

**collect\_lor\_codes\_by\_prefix** (*prefix, update=False, verbose=False*)

Collect LOR codes by a given prefix.

**Parameters**

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

**Returns** LOR codes for the given prefix

**Return type** dict, None

**Examples:**

```
from pyrcs.line_data import LOR

lor = LOR()

prefix = 'CY'
update = False
lor_codes_cy = lor.collect_lor_codes_by_prefix(prefix, update)
print(lor_codes_cy)
# {'CY': <codes>,
#   'Notes': <notes>,
#   'Last updated date': <date>}

prefix = 'NW'
update = False
lor_codes_nw = lor.collect_lor_codes_by_prefix(prefix, update)
print(lor_codes_nw)
# {'NW/NZ': <codes>,
#   'Notes': <codes>,
#   'Last updated date': <date>}

prefix = 'EA'
update = True
lor_codes_ea = lor.collect_lor_codes_by_prefix(prefix, update)
print(lor_codes_ea)
# {'EA': <codes>,
#   'Last updated date': <date>}
```

**fetch\_lor\_codes** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)

Fetch LOR codes from local backup.

**Parameters**

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

**Returns** LOR codes

**Return type** dict

**Example:**

```
from pyrcs.line_data import LOR

lor = LOR()

update = False
pickle_it = False
data_dir = None

lor_codes_data = lor.fetch_lor_codes(update, pickle_it, data_dir)

print(lor_codes_data)
# {'LOR': <codes>,
#   'Last_updated_date': <date>}
```

**collect\_elr\_lor\_converter** (*confirmation\_required=True, verbose=False*)

Collect ELR/LOR converter from source web page.

**Parameters**

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

**Returns** data of ELR/LOR converter

**Return type** dict, None

**Example:**

```
from pyrcs.line_data import LOR

lor = LOR()

confirmation_required = True

elr_lor_converter = lor.collect_elr_lor_converter(confirmation_
↪required)
# To collect "ELR/LOR converter"? [No]|Yes:
# >? yes

print(elr_lor_converter)
# {'ELR/LOR converter': <codes>,
#   'Last updated date': <date>}
```

**fetch\_elr\_lor\_converter** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)

Fetch ELR/LOR converter from local backup.

**Parameters**

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

**Returns** data of ELR/LOR converter

**Return type** dict

**Example:**

```
from pyrcs.line_data import LOR

lor = LOR()

update = False
pickle_it = False
data_dir = None

elr_lor_converter = lor.fetch_elr_lor_converter(update, pickle_it,
↪data_dir)

print(elr_lor_converter)
# {'ELR/LOR converter': <codes>,
#   'Last updated date': <date>}
```

### 3.1.6 TrackDiagrams

A class for collecting a catalogue of some sample British railway track diagrams.

<code>TrackDiagrams.</code>	Collect catalogue of sample railway track diagrams
<code>collect_sample_track_diagrams_catalogue</code>	(from source web page).
<code>TrackDiagrams.</code>	Fetch catalogue of sample railway track diagrams
<code>fetch_sample_track_diagrams_catalogue</code>	(from local backup).

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

A class for collecting British railway track diagrams.

#### Parameters

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

**Example:**

```
from pyrcs.line_data import TrackDiagrams

td = TrackDiagrams()

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

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

**collect\_sample\_track\_diagrams\_catalogue** (`confirmation_required=True`, `verbose=False`)

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

#### Parameters

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

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

**Return type** dict, None

**Example:**

```
from pyrcs.line_data import TrackDiagrams

td = TrackDiagrams()

confirmation_required = True

track_diagrams_catalogue = td.collect_sample_track_diagrams_
    ↪ catalogue(confirmation_required)
# To collect the catalogue of sample track diagrams? [No]/Yes:
# >? yes

print(track_diagrams_catalogue)
# {'Track diagrams': <code>,
#   'Last updated date': <date>}
```

**fetch\_sample\_track\_diagrams\_catalogue** (*update=False*, *pickle\_it=False*,  
*data\_dir=None*, *verbose=False*)  
Fetch catalogue of sample railway track diagrams from local backup.

**Parameters**

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

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

**Return type** dict

**Example:**

```
from pyrcs.line_data import TrackDiagrams

td = TrackDiagrams()

update = False
pickle_it = False
data_dir = None

track_diagrams_cat = td.fetch_sample_track_diagrams_
    ↪ catalogue(update, pickle_it, data_dir)

print(track_diagrams_cat)
# {'Track diagrams': <code>,
#   'Last updated date': <date>}
```

Table 7 – continued from previous page

<i>electrification</i>	Collecting section codes for OLE installations.
<i>elrs_mileages</i>	Collecting Engineer's Line References (ELRs) codes.
<i>line_names</i>	Collecting British railway line names.
<i>lor_codes</i>	Collecting line of route (LOR) codes.
<i>track_diagrams</i>	Collecting British railway track diagrams.

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

**Parameters** `update` (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`

**Example:**

```
from pyrcs import LineData

ld = LineData()

# To get location codes
lid = ld.LocationIdentifiers

# location codes that start with 'A'
location_codes_a = lid.collect_location_codes_by_initial('A')
```

## 3.2 other\_assets

Classes for collecting other assets data.

### 3.2.1 Depots

A class for collecting depot codes.

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

**class** `pyrcs.other_assets.Depots` (*data\_dir=None, update=False*)

A class for collecting depot codes.

**Parameters**

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

**Example:**

```
from pyrcs.other_assets import Depots

depots = Depots()

print(depots.Name)
# Depot codes

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

**collect\_two\_char\_tops\_codes** (*confirmation\_required=True*, *verbose=False*)  
Collect two-character TOPS codes from source web page.

**Parameters**

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

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

**Return type** dict, *None*

**Example:**

```
from pyrcs.other_assets import Depots

depots = Depots()

confirmation_required = True

two_char_tops_codes_data = depots.collect_two_char_tops_
    ↪codes(confirmation_required)
# To collect data of two character TOPS codes? [No]|Yes:
# >? yes

print(two_char_tops_codes_data)
# {'Two character TOPS codes': <codes>,
#   'Last updated date': <date>}
```

**fetch\_two\_char\_tops\_codes** (*update=False*, *pickle\_it=False*, *data\_dir=None*, *verbose=False*)  
Fetch two-character TOPS codes from local backup.

**Parameters**

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



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

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

**Return type** dict

**Example:**

```
from pyrcs.other_assets import Depots

depots = Depots()

update = False
pickle_it = False
data_dir = None

two_char_tops_codes_data = depots.fetch_two_char_tops_codes(update,
    ↪ pickle_it, data_dir)

print(two_char_tops_codes_data)
# {'Two character TOPS codes': <codes>,
#   'Last updated date': <date>}
```

**collect\_four\_digit\_pre\_tops\_codes** (*confirmation\_required=True, verbose=False*)  
Collect four-digit pre-TOPS codes from source web page.

**Parameters**

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

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

**Return type** dict, None

**Example:**

```
from pyrcs.other_assets import Depots

depots = Depots()

confirmation_required = True

four_digit_pre_tops_codes = depots.collect_four_digit_pre_tops_
    ↪ codes(confirmation_required)
# To collect data of four digit pre-TOPS codes? [No]/Yes:
# >? yes

print(four_digit_pre_tops_codes)
# {'Four digit pre-TOPS codes': <codes>,
#   'Last updated date': <date>}
```

**fetch\_four\_digit\_pre\_tops\_codes** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)  
Fetch four-digit pre-TOPS codes from local backup.

**Parameters**

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`

- **pickle\_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data\_dir** (*str*, *None*) – name of package data folder, defaults to `None`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

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

**Return type** dict

**Example:**

```
from pyrcs.other_assets import Depots

depots = Depots()

update = False
pickle_it = False
data_dir = None

four_digit_pretops_codes = depots.fetch_four_digit_pre_tops_
    ↪codes(update, pickle_it, data_dir)

print(four_digit_pretops_codes)
# {'Four digit pre-TOPS codes': <codes>,
#  'Last updated date': <date>}
```

**collect\_1950\_system\_codes** (*confirmation\_required=True*, *verbose=False*)

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

**Parameters**

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

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

**Return type** dict, `None`

**Example:**

```
from pyrcs.other_assets import Depots

depots = Depots()

confirmation_required = True

system_1950_codes_data = depots.collect_1950_system_
    ↪codes(confirmation_required)
# To collect data of 1950 system (pre-TOPS) codes? [No]|Yes:
# >? yes

print(system_1950_codes_data)
# {'1950 system (pre-TOPS) codes': <codes>,
#  'Last updated date': <date>}
```

**fetch\_1950\_system\_codes** (*update=False*, *pickle\_it=False*, *data\_dir=None*, *verbose=False*)

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

**Parameters**

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

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

**Return type** dict

**Example:**

```
from pyrcs.other_assets import Depots

depots = Depots()

update = False
pickle_it = False
data_dir = None

system_1950_codes_data = depots.fetch_1950_system_codes(update,
    pickle_it, data_dir)

print(system_1950_codes_data)
# {'1950 system (pre-TOPS) codes': <codes>,
#  'Last updated date': <date>}
```

**collect\_gwr\_codes** (*confirmation\_required=True*, *verbose=False*)

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

**Parameters**

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

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

**Return type** dict, None

**Example:**

```
from pyrcs.other_assets import Depots

depots = Depots()

confirmation_required = True

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

print(gwr_codes_data)
# {'GWR codes': <codes>,
#  'Last updated date': <date>}
```

**fetch\_gwr\_codes** (*update=False*, *pickle\_it=False*, *data\_dir=None*, *verbose=False*)

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

**Parameters**

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

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

**Return type** dict

**Example:**

```
from pyrcs.other_assets import Depots

depots = Depots()

update = False
pickle_it = False
data_dir = None

gwr_codes_data = depots.fetch_gwr_codes(update, pickle_it, data_
    ↪dir)

print(gwr_codes_data)
# {'GWR codes': <codes>,
#  'Last updated date': <date>}
```

**fetch\_depot\_codes** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)

Fetch depots codes from local backup.

**Parameters**

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

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

**Return type** dict

**Example:**

```
from pyrcs.other_assets import Depots

depots = Depots()

update = False
pickle_it = False
data_dir = None

depot_codes = depots.fetch_depot_codes(update, pickle_it, data_dir)
```

(continues on next page)

(continued from previous page)

```
print(depot_codes)
# {'Depots': <codes>,
#   'Last updated date': <date>}
```

### 3.2.2 Features

A class for collecting infrastructure features, including OLE neutral sections, HABD and WILD, water troughs, telegraph codes and driver/guard buzzer codes.

<code>Features.decode_vulgar_fraction(x)</code>	Decode vulgar fraction.
<code>Features.parse_vulgar_fraction_in_length(x)</code>	Parse 'VULGAR FRACTION' for 'Length' of water trough locations.
<code>Features.collect_habds_and_wilds([...])</code>	Collect codes of hot axle box detectors (HABDs) and wheel impact load detectors (WILDs) from source web page.
<code>Features.fetch_habds_and_wilds([update, ...])</code>	Fetch codes of HABDs and WILDs from local backup.
<code>Features.collect_water_troughs([...])</code>	Collect codes of water troughs from source web page.
<code>Features.fetch_water_troughs([update, ...])</code>	Fetch codes of water troughs from local backup.
<code>Features.collect_telegraph_codes([...])</code>	Collect telegraph code words from source web page.
<code>Features.fetch_telegraph_codes([update, ...])</code>	Fetch telegraph code words from local backup.
<code>Features.collect_buzzer_codes([...])</code>	Collect buzzer codes from source web page.
<code>Features.fetch_buzzer_codes([update, ...])</code>	Fetch buzzer codes from local backup.
<code>Features.fetch_features_codes([update, ...])</code>	Fetch features codes from local backup.

**class** `pyrcs.other_assets.Features` (*data\_dir=None, update=False*)

A class for collecting infrastructure features, including OLE neutral sections, HABD and WILD, water troughs, telegraph codes and driver/guard buzzer codes.

#### Parameters

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

#### Example:

```
from pyrcs.other_assets import Features

features = Features()

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

**static** `decode_vulgar_fraction(x)`  
Decode vulgar fraction.

**parse\_vulgar\_fraction\_in\_length(x)**  
Parse 'VULGAR FRACTION' for 'Length' of water trough locations.

**collect\_habds\_and\_wilds** (*confirmation\_required=True, verbose=False*)  
Collect codes of hot axle box detectors (HABDs) and wheel impact load detectors (WILDs) from

source web page.

#### Parameters

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

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

**Return type** dict, None

#### Example:

```
from pyrcs.other_assets import Features

features = Features()

confirmation_required = True

habds_and_wilds_codes_data = features.collect_habds_and_
    ↪wilds(confirmation_required)
# To collect data of HABD and WILD? [No]|Yes:
# >? yes

print(habds_and_wilds_codes_data)
# {'HABD and WILD': <codes>,
#   'Last updated date': <date>}
```

**fetch\_habds\_and\_wilds** (*update=False*, *pickle\_it=False*, *data\_dir=None*, *verbose=False*)  
Fetch codes of HABDs and WILDs from local backup.

#### Parameters

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

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

**Return type** dict

#### Example:

```
from pyrcs.other_assets import Features

features = Features()

update = False
pickle_it = False
data_dir = None

habds_and_wilds_codes_data = features.fetch_habds_and_wilds(update,
    ↪ pickle_it, data_dir)
```

(continues on next page)

(continued from previous page)

```
print(habds_and_wilds_codes_data)
# {'HABD and WILD': <codes>,
#   'Last updated date': <date>}
```

**collect\_water\_troughs** (*confirmation\_required=True, verbose=False*)

Collect codes of water troughs from source web page.

#### Parameters

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

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

**Return type** dict, None

**Example:**

```
from pyrcs.other_assets import Features

features = Features()

confirmation_required = True

water_troughs_data = features.collect_water_troughs(confirmation_
↪required)
# To collect data of water troughs? [No]|Yes:
# >? yes

print(water_troughs_data)
# {'Water troughs': <codes>,
#   'Last updated date': <date>}
```

**fetch\_water\_troughs** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)

Fetch codes of water troughs from local backup.

#### Parameters

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

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

**Return type** dict

**Example:**

```
from pyrcs.other_assets import Features

features = Features()

update = False
```

(continues on next page)

(continued from previous page)

```
pickle_it = False
data_dir = None

water_troughs_data = features.fetch_water_troughs(update, pickle_
↪it, data_dir)

print(water_troughs_data)
# {'Water troughs': <codes>,
#   'Last updated date': <date>}
```

**collect\_telegraph\_codes** (*confirmation\_required=True, verbose=False*)

Collect telegraph code words from source web page.

**Parameters**

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

**Returns** data of telegraph code words, and date of when the data was last updated**Return type** dict, None**Example:**

```
from pyrcs.other_assets import Features

features = Features()

confirmation_required = True

telegraph_codes_data = features.collect_telegraph_
↪codes(confirmation_required)
# To collect data of telegraphic codes? [No]/Yes:
# >? yes

print(telegraph_codes_data)
# {'Telegraphic codes': <codes>,
#   'Last updated date': <date>}
```

**fetch\_telegraph\_codes** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)

Fetch telegraph code words from local backup.

**Parameters**

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

**Returns** data of telegraph code words, and date of when the data was last updated**Return type** dict**Example:**



```

from pyrcs.other_assets import Features

features = Features()

update = False
pickle_it = False
data_dir = None

telegraph_codes_data = features.fetch_telegraph_codes(update,
    ↪pickle_it, data_dir)

print(telegraph_codes_data)
# {'Telegraphic codes': <codes>,
#   'Last updated date': <date>}

```

**collect\_buzzer\_codes** (*confirmation\_required=True, verbose=False*)

Collect buzzer codes from source web page.

#### Parameters

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

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

**Return type** dict, None

**Example:**

```

from pyrcs.other_assets import Features

features = Features()

confirmation_required = True

buzzer_codes_data = features.collect_buzzer_codes(confirmation_
    ↪required)
# To collect data of buzzer codes? [No]/Yes:
# >? yes

print(buzzer_codes_data)
# {'Buzzer codes': <codes>,
#   'Last updated date': <date>}

```

**fetch\_buzzer\_codes** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)

Fetch buzzer codes from local backup.

#### Parameters

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

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

**Return type** dict

**Example:**

```
from pyrcs.other_assets import Features

features = Features()

update = False
pickle_it = False
data_dir = None

buzzer_codes_data = features.fetch_buzzer_codes(update, pickle_it,
↪data_dir)

print(buzzer_codes_data)
# {'Buzzer codes': <codes>,
#   'Last updated date': <date>}
```

**fetch\_features\_codes** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)

Fetch features codes from local backup.

**Parameters**

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

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

**Return type** dict

**Example:**

```
from pyrcs.other_assets import Features

features = Features()

update = False
pickle_it = False
data_dir = None

features_codes = features.fetch_features_codes(update, pickle_it,
↪data_dir)

print(features_codes)
# {'Features': <codes>,
#   'Last updated date': <date>}
```

### 3.2.3 SignalBoxes

A class for collecting signal box prefix codes.

<code>SignalBoxes.collect_signal_box_prefix_codes(initial)</code>	Collect (signal) box prefix codes for the given initial from source web page.
<code>SignalBoxes.fetch_signal_box_prefix_codes(initial)</code>	Fetch (signal) box prefix codes from local backup.
<code>SignalBoxes.collect_non_national_rail_signal_box_prefix_codes(initial)</code>	Collect (signal) box prefix codes of non-national rail from source web page.
<code>SignalBoxes.fetch_non_national_rail_signal_box_prefix_codes(initial)</code>	Fetch (signal) box prefix codes of non-national rail from local backup.

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

A class for collecting signal box prefix codes.

#### Parameters

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

#### Example:

```
from pyrcs.other_assets import SignalBoxes

sb = SignalBoxes()

print (sb.Name)
# Signal box prefix codes

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

**collect\_signal\_box\_prefix\_codes** (*initial, update=False, verbose=False*)

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

#### Parameters

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

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

**Return type** dict

#### Example:

```
from pyrcs.other_assets import SignalBoxes

sb = SignalBoxes()

update = False

initial = 'a'
signal_boxes_a = sb.collect_signal_box_prefix_codes(initial, update)

print (signal_boxes_a)
# {'A': <codes>,
#   'Last updated date': <date>}
```

**fetch\_signal\_box\_prefix\_codes** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)  
Fetch signal box prefix codes from local backup.

**Parameters**

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

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

**Return type** dict

**Example:**

```
from pyrcs.other_assets import SignalBoxes

sb = SignalBoxes()

update = False
pickle_it = False
data_dir = None

signal_box_prefix_codes = sb.fetch_signal_box_prefix_codes(update,
    ↪pickle_it, data_dir)

print(signal_box_prefix_codes)
# {'Signal boxes': <codes>,
#  'Latest update date': <date>}
```

**collect\_non\_national\_rail\_codes** (*confirmation\_required=True, verbose=False*)  
Collect signal box prefix codes of non-national rail from source web page.

**Parameters**

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

**Returns** signal box prefix codes of non-national rail

**Return type** dict, None

**Example:**

```
from pyrcs.other_assets import SignalBoxes

sb = SignalBoxes()

confirmation_required = True

non_national_rail_codes_data = sb.collect_non_national_rail_
    ↪codes(confirmation_required)
# To collect signal box data of non-national rail? [No]|Yes:
# >? yes
```

(continues on next page)

(continued from previous page)

```
print(non_national_rail_codes_data)
# {'Non-national rail': <codes>,
#   'Last updated date': <date>}
```

**fetch\_non\_national\_rail\_codes** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)

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

#### Parameters

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

**Returns** signal box prefix codes of non-national rail

**Return type** dict

#### Example:

```
from pyrcs.other_assets import SignalBoxes

sb = SignalBoxes()

update = False
pickle_it = False
data_dir = None

non_national_rail_codes_data = sb.fetch_non_national_rail_
    ↪codes(update, pickle_it, data_dir)

print(non_national_rail_codes_data)
# {'Non-national rail': <codes>,
#   'Last updated date': <date>}
```

## 3.2.4 Stations

A class for collecting railway station data.

<code>Stations.parse_current_operator(x)</code>	Parse 'Operator' column :param x: :return:
<code>Stations.collect_railway_station_data_station_data(station)</code>	Collect railway station data for the given initial letter.
<code>Stations.fetch_railway_station_data([...])</code>	Fetch railway station data from local backup.

**class** `pyrcs.other_assets.Stations` (*data\_dir=None, update=False*)

A class for collecting railway station data.

#### Parameters

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

Example:

```
from pyrcs.other_assets import Stations

stn = Stations()

print(stn.Name)
# Stations

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

**static parse\_current\_operator**(*x*)

Parse 'Operator' column :param x: :return:

**collect\_railway\_station\_data\_by\_initial**(*initial*, *update=False*, *verbose=False*)

Collect railway station data for the given initial letter.

#### Parameters

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

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

**Return type** dict

Example:

```
from pyrcs.other_assets import Stations

stn = Stations()

update = False

initial = 'a'
railway_station_data_a = stn.collect_railway_station_data_by_
    ↪initial(initial, update)

print(railway_station_data_a)
# {'A': <codes>,
#   'Last updated date': <date>}
```

**fetch\_railway\_station\_data**(*update=False*, *pickle\_it=False*, *data\_dir=None*, *verbose=False*)

Fetch railway station data from local backup.

#### Parameters

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **pickle\_it** (*bool*) – whether to replace the current package data with newly collected data, defaults to `False`
- **data\_dir** (*str*, *None*) – name of package data folder, defaults to `None`

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

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

**Return type** dict

**Example:**

```
from pyrcs.other_assets import Stations

stn = Stations()

update = False
pickle_it = False
data_dir = None

railway_station_data = stn.fetch_railway_station_data(update,
↪pickle_it, data_dir)

print(railway_station_data)
# {'Railway station data': <codes>,
#   'Latest update date': <date>}
```

### 3.2.5 Tunnels

A class for collecting railway tunnel lengths.

<code>Tunnels.parse_tunnel_length(x)</code>	Parse data in 'Length' column, i.e. convert miles/yards to metres.
<code>Tunnels.collect_railway_tunnel_lengths(page_no)</code>	Collect data (page no) tunnel lengths for a given page number from source web page.
<code>Tunnels.fetch_railway_tunnel_lengths()</code>	Fetch data of railway tunnel lengths from local backup.

**class** `pyrcs.other_assets.Tunnels` (*data\_dir=None, update=False*)

A class for collecting railway tunnel lengths.

#### Parameters

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

**Example:**

```
from pyrcs.other_assets import Tunnels

tunnels = Tunnels()

print(tunnels.Name)
# Railway tunnel lengths

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

**static** `parse_tunnel_length(x)`  
Parse data in 'Length' column, i.e. convert miles/yards to metres.

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

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

**Return type** tuple

**Examples:**

```
from pyrcs.other_assets import Tunnels

tunnels = Tunnels()

tunnels.parse_tunnel_length('')
# (nan, 'Unavailable')

tunnels.parse_tunnel_length('1m 182y')
# (1775.7648, None)

tunnels.parse_tunnel_length('formerly 0m236y')
# (215.7984, 'Formerly')

tunnels.parse_tunnel_length('0.325km (0m 356y)')
# (325.5264, '0.325km')

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

**collect\_railway\_tunnel\_lengths\_by\_page** (*page\_no*, *update=False*, *verbose=False*)

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

**Parameters**

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

**Returns** tunnel lengths data of the given *page\_no* and date of when the data was last updated

**Return type** dict

**Examples:**

```
from pyrcs.other_assets import Tunnels

tunnels = Tunnels()

update = True

page_no = 1
railway_tunnel_lengths_1 = tunnels.collect_railway_tunnel_lengths_
    ↪by_page(page_no, update)
print(railway_tunnel_lengths_1)
# {'Page 1 (A-F)': <codes>,
#   'Last updated date': <date>}

page_no = 4
railway_tunnel_lengths_4 = tunnels.collect_railway_tunnel_lengths_
    ↪by_page(page_no, update)
```

(continues on next page)



(continued from previous page)

```
print(railway_tunnel_lengths_4)
# {'Page 4 (others)': <codes>,
#   'Last updated date': <date>}
```

**fetch\_railway\_tunnel\_lengths** (*update=False, pickle\_it=False, data\_dir=None, verbose=False*)

Fetch data of railway tunnel lengths from local backup.

#### Parameters

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

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

**Return type** dict

**Example:**

```
from pyrcs.other_assets import Tunnels

tunnels = Tunnels()

update = False
pickle_it = False
data_dir = None

railway_tunnel_lengths = tunnels.fetch_railway_tunnel_
    lengths(update, pickle_it, data_dir)

print(railway_tunnel_lengths)
# {'Tunnels': <codes>,
#   'Latest update date': <date>}
```

## 3.2.6 Viaducts

A class for collecting railway viaducts.

---

*Viaducts.collect\_railway\_viaducts\_by\_page(*page\_num*)* Collect data of railway viaducts for a given page number from source web page.

---

*Viaducts.fetch\_railway\_viaducts(*[update, pickle\_it, data\_dir]*)* Fetch data of railway viaducts from local backup. ...]

---

**class** `pyrcs.other_assets.Viaducts` (*data\_dir=None, update=False*)

A class for collecting railway viaducts.

#### Parameters

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

Example:

```
from pyrcs.other_assets import Viaducts

viaducts = Viaducts()

print(viaducts.Name)
# Railway viaducts

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

**collect\_railway\_viaducts\_by\_page** (*page\_no*, *update=False*, *verbose=False*)

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

**Parameters**

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

**Returns** railway viaducts data of the given *page\_no* and date of when the data was last updated

**Return type** dict

Example:

```
from pyrcs.other_assets import Viaducts

viaducts = Viaducts()

update = True

page_no = 1
railway_viaducts_1 = viaducts.collect_railway_viaducts_by_
    ↪page(page_no, update)

print(railway_viaducts_1)
# {'Page 1 (A-C)': <codes>,
#  'Last updated date': <date>}
```

**fetch\_railway\_viaducts** (*update=False*, *pickle\_it=False*, *data\_dir=None*, *verbose=False*)

Fetch data of railway viaducts from local backup.

**Parameters**

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

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

**Return type** dict

Example:

```
from pyrcs.other_assets import Viaducts

viaducts = Viaducts()

update = False
pickle_it = False
data_dir = None

railway_viaducts = viaducts.fetch_railway_viaducts(update, pickle_
↪it, data_dir)

print(railway_tunnel_lengths)
# {'Viaducts': <codes>,
#   'Latest update date': <date>}
```

<i>depots</i>	Collecting depots codes.
<i>features</i>	Collecting codes of infrastructure features.
<i>signal_boxes</i>	Collecting signal box prefix codes.
<i>stations</i>	Collecting railway station data.
<i>tunnels</i>	Collecting codes of railway tunnel lengths.
<i>viaducts</i>	Collecting codes of railway viaducts.

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

**Parameters** `update` (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`

Example:

```
from pyrcs import OtherAssets

oa = OtherAssets()

# To get railway station data
stations = oa.Stations

# data of railway stations whose names start with 'A'
railway_station_data_a = stations.collect_railway_station_data_by_
↪initial('A')
```

### 3.3 updater

A module for updating package data.

<i>collect_site_map</i> ([ <i>confirmation_required</i> ])	Collect data of the site map.
<i>fetch_site_map</i> ([ <i>update</i> , ...])	Fetch the site map from the package data.
<i>update_backup_data</i> ([ <i>verbose</i> , <i>time_gap</i> ])	Update package data.

`pyrcs.updater.collect_site_map` (*confirmation\_required=True*)

Collect data of the site map.

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

**Returns** dictionary of site map data

**Return type** dict

`pyrcs.updater.fetch_site_map(update=False, confirmation_required=True, verbose=False)`

Fetch the site map from the package data.

**Parameters**

- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **confirmation\_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool*, *int*) – whether to print relevant information in console as the function runs, defaults to `False`

**Returns** dictionary of site map data

**Return type** dict

**Examples:**

```
from pyrcs.updater import fetch_site_map

update = False
site_map = fetch_site_map(update)

update = True
site_map = fetch_site_map(update)
```

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

Update package data.

**Parameters**

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

**Example:**

```
from pyrcs.updater import update_backup_data

verbose = True
time_gap = 5

update_backup_data(verbose, time_gap)
```

## 3.4 utils

A module of helper functions.

### 3.4.1 Source homepage

---

<code>homepage_url()</code>	Specify the homepage URL of the data source.
-----------------------------	--

---

`pyrcs.utils.homepage_url()`

Specify the homepage URL of the data source.

**Returns** URL of the data source homepage

**Return type** `str`

### 3.4.2 Directory

---

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

---

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

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

**Parameters**

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

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

**Return type** `str`

**Example:**

```
from pyrcs.utils import cd_dat

dat_dir = "dat"
mkdir = False

cd_dat("line-data", dat_dir=dat_dir, mkdir=mkdir)
# "\dat\line-data"
```

### 3.4.3 Converters

---

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

---

Continued on next page

Table 18 – continued from previous page

<code>nr_mileage_num_to_str(num_mileage)</code>	Convert numerical-type Network Rail mileage to string-type one.
<code>nr_mileage_to_yards(nr_mileage)</code>	Convert Network Rail mileages to yards.
<code>yards_to_nr_mileage(yards)</code>	Convert yards to Network Rail mileages.
<code>shift_num_nr_mileage(nr_mileage, shift_yards)</code>	Shift Network Rail mileage by given yards.
<code>year_to_financial_year(date)</code>	Convert calendar year of a given date to Network Rail financial year.

`pyrcs.utils.mile_chain_to_nr_mileage(miles_chains)`

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

**Parameters** `miles_chains` (*str*, *numpy.nan*, *None*) – mileage data presented in the form ‘<miles>.<chains>’

**Returns** Network Rail mileage in the form ‘<miles>.<yards>’

**Return type** `str`

**Examples:**

```
from pyrcs.utils import mile_chain_to_nr_mileage

miles_chains = '0.18' # AAM 0.18 Tewkesbury Junction with ANZ (84.62)
mile_chain_to_nr_mileage(miles_chains) # '0.0396'

miles_chains = None # or np.nan, or ''
mile_chain_to_nr_mileage(miles_chains) # ''
```

`pyrcs.utils.nr_mileage_to_mile_chain(str_mileage)`

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

**Parameters** `str_mileage` (*str*, *numpy.nan*, *None*) – Network Rail mileage data presented in the form ‘<miles>.<yards>’

**Returns** ‘<miles>.<chains>’

**Return type** `str`

**Examples:**

```
from pyrcs.utils import nr_mileage_to_mile_chain

str_mileage = '0.0396'
nr_mileage_to_mile_chain(str_mileage) # '0.18'

str_mileage = None # or np.nan, or ''
nr_mileage_to_mile_chain(str_mileage) # ''
```

`pyrcs.utils.nr_mileage_str_to_num(str_mileage)`

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

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

**Returns** numerical-type Network Rail mileage

**Return type** `float`

**Examples:**

```

from pyrcs.utils import nr_mileage_str_to_num

str_mileage = '0.0396'
nr_mileage_str_to_num(str_mileage)    # 0.0396

str_mileage = ''
nr_mileage_str_to_num(str_mileage)    # nan

```

`pyrcs.utils.nr_mileage_num_to_str(num_mileage)`

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

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

**Returns** string-type Network Rail mileage in the form ‘<miles>.<yards>’

**Return type** `str`

**Examples:**

```

import numpy as np
from pyrcs.utils import nr_mileage_num_to_str

num_mileage = 0.0396
nr_mileage_num_to_str(num_mileage)    # '0.0396'

num_mileage = np.nan
nr_mileage_num_to_str(num_mileage)    # ''

```

`pyrcs.utils.nr_mileage_to_yards(nr_mileage)`

Convert Network Rail mileages to yards.

**Parameters** `nr_mileage` (*float, str*) – Network Rail mileage

**Returns** `yards`

**Return type** `int`

**Examples:**

```

from pyrcs.utils import nr_mileage_to_yards

nr_mileage = '0.0396'
nr_mileage_to_yards(nr_mileage)    # 396

nr_mileage = 0.0396
nr_mileage_to_yards(nr_mileage)    # 396

```

`pyrcs.utils.yards_to_nr_mileage(yards)`

Convert yards to Network Rail mileages.

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

**Returns** Network Rail mileage in the form ‘<miles>.<yards>’

**Return type** `str`

**Examples:**

```

from pyrcs.utils import yards_to_nr_mileage

yards = 396
yards_to_nr_mileage(yards)    # '0.0396'

```

(continues on next page)

(continued from previous page)

```
yards = 396.0
yards_to_nr_mileage(yards) # '0.0396'

yards = None
yards_to_nr_mileage(yards) # ''
```

`pyrcs.utils.shift_num_nr_mileage(nr_mileage, shift_yards)`  
Shift Network Rail mileage by given yards.

**Parameters**

- **nr\_mileage** (*float, int, str*) – Network Rail mileage
- **shift\_yards** (*int, float*) – yards by which the given nr\_mileage is shifted

**Returns** shifted numerical Network Rail mileage

**Return type** float

**Examples:**

```
from pyrcs.utils import shift_num_nr_mileage

nr_mileage = '0.0396' # or 0.0396
shift_yards = 220
shift_num_nr_mileage(nr_mileage, shift_yards) # 0.0616

nr_mileage = '0.0396'
shift_yards = 220.99
shift_num_nr_mileage(nr_mileage, shift_yards) # 0.0617

nr_mileage = 10
shift_yards = 220
shift_num_nr_mileage(nr_mileage, shift_yards) # 10.022
```

`pyrcs.utils.year_to_financial_year(date)`  
Convert calendar year of a given date to Network Rail financial year.

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

**Returns** Network Rail financial year of the given date

**Return type** int

**Example:**

```
from pyrcs.utils import year_to_financial_year

date = datetime.datetime.now()

year_to_financial_year(date) # 2020
```

## 3.4.4 Parsers

---

<code>parse_tr(header, trs)</code>	Parse a list of parsed HTML <tr> elements.
<code>parse_table(source[, parser])</code>	Parse HTML <tr> elements for creating a data frame.

---

Continued on next page



Table 19 – continued from previous page

<code>parse_location_name(location_name)</code>	Parse location name (and its associated note).
<code>parse_date(str_date[, as_date_type])</code>	Parse a date.

`pyrcs.utils.parse_tr(header, trs)`  
 Parse a list of parsed HTML <tr> elements.

See also [PT-1].

#### Parameters

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

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

**Return type** list

**Example:**

```
import bs4
import fake_useragent
from pyrcs.utils import fake_requests_headers, parse_tr

source = requests.get(
    'http://www.railwaycodes.org.uk/elrs/elra.shtm',
    headers=fake_requests_headers())
parsed_text = bs4.BeautifulSoup(source.text, 'lxml')
header = [x.text for x in parsed_text.find_all('th')] # Column names
trs = parsed_text.find_all('tr')

parse_tr(header, trs) # returns a list of lists
```

`pyrcs.utils.parse_table(source, parser='lxml')`  
 Parse HTML <tr> elements for creating a data frame.

#### Parameters

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

#### Returns

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

**Return type** tuple

**Examples:**

```
import bs4
import fake_useragent
from pyrcs.utils import fake_requests_headers, parse_table

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

parse_table(source, parser)
```

`pyrcs.utils.parse_location_name(location_name)`

Parse location name (and its associated note).

**Parameters** `location_name` (*str*, *None*) – location name (in raw data)

**Returns** location name and, if any, note

**Return type** tuple

**Examples:**

```
from pyrcs.utils import parse_location_name

location_dat = 'Abbey Wood'
parse_location_name(location_dat)
# ('Abbey Wood', '')

location_dat = None
parse_location_name(location_dat)
# ('', '')

location_dat = 'Abercynon (formerly Abercynon South)'
parse_location_name(location_dat)
# ('Abercynon', 'formerly Abercynon South')

location_dat = 'Allerton (reopened as Liverpool South Parkway)'
parse_location_name(location_dat)
# ('Allerton', 'reopened as Liverpool South Parkway')

location_dat = 'Ashford International [domestic portion]'
parse_location_name(location_dat)
# ('Ashford International', 'domestic portion')
```

`pyrcs.utils.parse_date(str_date, as_date_type=False)`

Parse a date.

**Parameters**

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

**Returns** parsed date as a string or `datetime.date`

**Return type** `str`, `datetime.date`

**Examples:**

```
from pyrcs.utils import parse_date

str_date = '2020-01-01'

as_date_type = True
parse_date(str_date, as_date_type) # datetime.date(2020, 1, 1)
```

### 3.4.5 Get useful information

<code>fake_requests_headers([randomized])</code>	Make a fake HTTP headers for <code>requests.get</code> .
<code>get_last_updated_date(url[, parsed, ...])</code>	Get last update date.
<code>get_catalogue(page_url[, update, ...])</code>	Get the catalogue for a class.
<code>get_category_menu(menu_url[, update, ...])</code>	Get a menu of the available classes.
<code>get_station_data_catalogue(source_url, ...)</code>	Get catalogue of railway station data.
<code>get_track_diagrams_items(source_url, source_key)</code>	Get catalogue of track diagrams.

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

Make a fake HTTP headers for `requests.get`.

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

**Returns** fake HTTP headers

**Return type** dict

**Examples:**

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

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

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

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

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

Get last update date.

**Parameters**

- `url (str)` – URL link of a requested web page
- `parsed (bool)` – whether to reformat the date, defaults to `True`
- `as_date_type (bool)` – whether to return the date as `datetime.date`, defaults to `False`

**Returns** date of when the specified web page was last updated

**Return type** str, `datetime.date`, `None`

**Examples:**

```
from pyrcs.utils import get_last_updated_date

parsed = True

url = 'http://www.railwaycodes.org.uk/crs/CRSa.shtm'

date_type = False
get_last_updated_date(url, parsed, date_type)
# '<year>-<month>-<day>'
```

(continues on next page)

(continued from previous page)

```
date_type = True
get_last_updated_date(url, parsed, date_type)
# datetime.date(<year>, <month>, <day>)

url = 'http://www.railwaycodes.org.uk/linedatamenu.shtm'
get_last_updated_date(url, parsed, date_type)
# None
```

```
pyrcs.utils.get_catalogue(page_url, update=False, confirmation_required=True,
                           json_it=True, verbose=False)
```

Get the catalogue for a class.

#### Parameters

- **page\_url** (*str*) – URL of the main page of a code category
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **confirmation\_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **json\_it** (*bool*) – whether to save the catalogue as a .json file, defaults to `True`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`

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

**Return type** dict

#### Examples:

```
from pyrcs.utils import get_catalogue

update = False
verbose = True

page_url = 'http://www.railwaycodes.org.uk/elrs/elr0.shtm'
confirmation_required = True
catalogue = get_catalogue(page_url, update, confirmation_required,
                           verbose)

page_url = 'http://www.railwaycodes.org.uk/linedatamenu.shtm'
confirmation_required = False
catalogue = get_catalogue(page_url, update, confirmation_required,
                           verbose)
```

```
pyrcs.utils.get_category_menu(menu_url, update=False, confirmation_required=True,
                              json_it=True, verbose=False)
```

Get a menu of the available classes.

#### Parameters

- **menu\_url** (*str*) – URL of the menu page
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to `False`
- **confirmation\_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **json\_it** (*bool*) – whether to save the catalogue as a .json file, defaults to `True`

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

**Returns**

**Return type** dict

**Example:**

```
from pyrcs.utils import get_category_menu

update = False
confirmation_required = True
verbose = True

menu_url = 'http://www.railwaycodes.org.uk/linedatamenu.shtm'
cls_menu = get_category_menu(menu_url)

print(cls_menu)
# {'<category name>': {'<title>': '<URL>'}}
```

`pyrcs.utils.get_station_data_catalogue` (*source\_url*, *source\_key*, *update=False*)  
Get catalogue of railway station data.

**Parameters**

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

**Returns** catalogue of railway station data

**Return type** dict

`pyrcs.utils.get_track_diagrams_items` (*source\_url*, *source\_key*, *update=False*)  
Get catalogue of track diagrams.

**Parameters**

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

**Returns** catalogue of railway station data

**Return type** dict

### 3.4.6 Rectification of location names

---

<code>fetch_location_names_repl_dict</code> ([ <i>k</i> , <i>regex</i> , ...])	Create a dictionary for rectifying location names.
---	--

---

<code>update_location_name_repl_dict</code> ( <i>new_items</i> , <i>regex</i> )	Update the location-name replacement dictionary in the package data.
--	--

---

`pyrcs.utils.fetch_location_names_repl_dict` (*k=None*, *regex=False*,  
*as\_dataframe=False*)

Create a dictionary for rectifying location names.

#### Parameters

- **k** (*str*, *int*, *float*, *bool*, *None*) – key of the created dictionary, defaults to *None*
- **regex** (*bool*) – whether to create a dictionary for replacement based on regular expressions, defaults to *False*
- **as\_dataframe** (*bool*) – whether to return the created dictionary as a `pandas.DataFrame`, defaults to *False*

**Returns** dictionary for rectifying location names

**Return type** `dict`, `pandas.DataFrame`

#### Examples:

```
from pyrcs.utils import fetch_location_names_repl_dict

k = None
regex = False
as_dataframe = True
fetch_location_names_repl_dict(k, regex, as_dataframe)

regex = True
as_dataframe = False
fetch_location_names_repl_dict(k, regex, as_dataframe)
```

`pyrcs.utils.update_location_name_repl_dict` (*new\_items*, *regex*, *verbose=False*)

Update the location-name replacement dictionary in the package data.

#### Parameters

- **new\_items** (*dict*) – new items to replace
- **regex** (*bool*) – whether this update is for regular-expression dictionary
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to *False*

#### Example:

```
from pyrcs.utils import update_location_name_repl_dict

verbose = True

new_items = {} regex = False update_location_name_repl_dict(new_items, regex, verbose)
```

## 3.4.7 Fixers

---

<code>fix_num_stanox</code> ( <i>stanox_code</i> )	Fix ‘STANOX’ if it is loaded as numbers.
--	--

---

`pyrcs.utils.fix_num_stanox` (*stanox\_code*)

Fix ‘STANOX’ if it is loaded as numbers.

**Parameters** **stanox\_code** (*str*, *int*) – STANOX code

**Returns** standard STANOX code

**Return type** str

**Examples:**

```
stanox_code = 65630
fix_num_stanox(stanox_code)  # '65630'

stanox_code = 2071
fix_num_stanox(stanox_code)  # '02071'
```

### 3.4.8 Misc

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

`pyrcs.utils.is_str_float(str_val)`

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

**Parameters** `str_val` (*str*) – a string-type variable

**Returns** whether `str_val` can express a float value

**Return type** bool

**Examples:**

```
str_val = ''
is_str_float(str_val)  # False

str_val = 'a'
is_str_float(str_val)  # False

str_val = '1'
is_str_float(str_val)  # True

str_val = '1.1'
is_str_float(str_val)  # True
```

<code>_line_data</code>	Collecting line data
<code>_other_assets</code>	Collecting data of other assets
<code>updater</code>	Update package data
<code>utils</code>	Utilities - Helper functions





## CHAPTER 4

---

### License

---

PyRCS is licensed under [GNU General Public License v3.0 \(GPLv3\)](#).



## CHAPTER 5

---

### Use of data

---

For the use of the data collected from this package, please refer to this link: <http://www.railwaycodes.org.uk/misc/contributing.shtm>



## CHAPTER 6

---

### Acknowledgements

---

Acknowledgements are given to those who contribute to the data resources: <http://www.railwaycodes.org.uk/misc/acknowledgements.shtm>



### c

`crs_nlc_tiploc_stanox`, 9

### d

`depots`, 35

### e

`electrification`, 15

`elrs_mileages`, 21

### f

`features`, 41

### l

`line_data.*`, 34

`line_names`, 27

`lor_codes`, 29

### o

`other_assets.*`, 55

### p

`pyrcs._line_data`, 35

`pyrcs._other_assets`, 55

`pyrcs.updater`, 55

`pyrcs.utils`, 56

### s

`signal_boxes`, 46

`stations`, 49

### t

`track_diagrams`, 33

`tunnels`, 51

### v

`viaducts`, 53





## A

amendment\_to\_location\_names\_dict() (pyrcs.line\_data.LocationIdentifiers static method), 10

## C

cd\_dat() (in module pyrcs.utils), 57  
 collect\_1950\_system\_codes() (pyrcs.other\_assets.Depots method), 38  
 collect\_buzzer\_codes() (pyrcs.other\_assets.Features method), 45  
 collect\_codes\_for\_energy\_tariff\_zones() (pyrcs.line\_data.Electrification method), 19  
 collect\_codes\_for\_independent\_lines() (pyrcs.line\_data.Electrification method), 17  
 collect\_codes\_for\_national\_network() (pyrcs.line\_data.Electrification method), 16  
 collect\_codes\_for\_ohns() (pyrcs.line\_data.Electrification method), 18  
 collect\_elr\_by\_initial() (pyrcs.line\_data.ELRMileages method), 22  
 collect\_elr\_lor\_converter() (pyrcs.line\_data.LOR method), 32  
 collect\_four\_digit\_pre\_tops\_codes() (pyrcs.other\_assets.Depots method), 37  
 collect\_gwr\_codes() (pyrcs.other\_assets.Depots method), 39  
 collect\_habds\_and\_wilds() (pyrcs.other\_assets.Features method), 41  
 collect\_line\_names() (pyrcs.line\_data.LineNames method), 28  
 collect\_location\_codes\_by\_initial() (pyrcs.line\_data.LocationIdentifiers method), 13  
 collect\_lor\_codes\_by\_prefix() (pyrcs.line\_data.LOR method), 30  
 collect\_mileage\_file\_by\_elr() (pyrcs.line\_data.ELRMileages method), 23  
 collect\_multiple\_station\_codes\_explanatory\_note() (pyrcs.line\_data.LocationIdentifiers method), 10  
 collect\_non\_national\_rail\_codes() (pyrcs.other\_assets.SignalBoxes method), 48  
 collect\_other\_systems\_codes() (pyrcs.line\_data.LocationIdentifiers method), 11  
 collect\_railway\_station\_data\_by\_initial() (pyrcs.other\_assets.Stations method), 50  
 collect\_railway\_tunnel\_lengths\_by\_page() (pyrcs.other\_assets.Tunnels method), 52  
 collect\_railway\_viaducts\_by\_page() (pyrcs.other\_assets.Viaducts method), 54  
 collect\_sample\_track\_diagrams\_catalogue() (pyrcs.line\_data.TrackDiagrams method), 33  
 collect\_signal\_box\_prefix\_codes() (pyrcs.other\_assets.SignalBoxes method), 47  
 collect\_site\_map() (in module pyrcs.updater), 55  
 collect\_telegraph\_codes() (pyrcs.other\_assets.Features method), 44

collect\_two\_char\_tops\_codes() (pyrcs.other\_assets.Depots method), 36  
 collect\_water\_troughs() (pyrcs.other\_assets.Features method), 43  
 crs\_nlc\_tiploc\_stanox (module), 9

## D

decode\_vulgar\_fraction() (pyrcs.other\_assets.Features static method), 41  
 Depots (class in pyrcs.other\_assets), 35  
 depots (module), 35

## E

Electrification (class in pyrcs.line\_data), 15  
 electrification (module), 15  
 ELMileages (class in pyrcs.line\_data), 21  
 elrs\_mileages (module), 21

## F

fake\_requests\_headers() (in module pyrcs.utils), 63  
 Features (class in pyrcs.other\_assets), 41  
 features (module), 41  
 fetch\_1950\_system\_codes() (pyrcs.other\_assets.Depots method), 38  
 fetch\_buzzer\_codes() (pyrcs.other\_assets.Features method), 45  
 fetch\_codes\_for\_energy\_tariff\_zones() (pyrcs.line\_data.Electrification method), 20  
 fetch\_codes\_for\_independent\_lines() (pyrcs.line\_data.Electrification method), 18  
 fetch\_codes\_for\_national\_network() (pyrcs.line\_data.Electrification method), 16  
 fetch\_codes\_for\_ohns() (pyrcs.line\_data.Electrification method), 19  
 fetch\_depot\_codes() (pyrcs.other\_assets.Depots method), 40  
 fetch\_electrification\_codes() (pyrcs.line\_data.Electrification method), 20  
 fetch\_elr() (pyrcs.line\_data.ELRMileages method), 22  
 fetch\_elr\_lor\_converter() (pyrcs.line\_data.LOR method), 32  
 fetch\_features\_codes() (pyrcs.other\_assets.Features method), 46  
 fetch\_four\_digit\_pre\_tops\_codes() (pyrcs.other\_assets.Depots method), 37  
 fetch\_gwr\_codes() (pyrcs.other\_assets.Depots method), 39  
 fetch\_habds\_and\_wilds() (pyrcs.other\_assets.Features method), 42  
 fetch\_line\_names() (pyrcs.line\_data.LineNames method), 28  
 fetch\_location\_codes() (pyrcs.line\_data.LocationIdentifiers method), 13  
 fetch\_location\_names\_repl\_dict() (in module pyrcs.utils), 66  
 fetch\_lor\_codes() (pyrcs.line\_data.LOR method), 31

`fetch_mileage_file()` (*pyrcs.line\_data.ELRMileages method*), 25  
`fetch_multiple_station_codes_explanatory_note()` (*pyrcs.line\_data.LocationIdentifiers method*), 11  
`fetch_non_national_rail_codes()` (*pyrcs.other\_assets.SignalBoxes method*), 49  
`fetch_other_systems_codes()` (*pyrcs.line\_data.LocationIdentifiers method*), 12  
`fetch_railway_station_data()` (*pyrcs.other\_assets.Stations method*), 50  
`fetch_railway_tunnel_lengths()` (*pyrcs.other\_assets.Tunnels method*), 53  
`fetch_railway_viaducts()` (*pyrcs.other\_assets.Viaducts method*), 54  
`fetch_sample_track_diagrams_catalogue()` (*pyrcs.line\_data.TrackDiagrams method*), 34  
`fetch_signal_box_prefix_codes()` (*pyrcs.other\_assets.SignalBoxes method*), 47  
`fetch_site_map()` (*in module pyrcs.updater*), 56  
`fetch_telegraph_codes()` (*pyrcs.other\_assets.Features method*), 44  
`fetch_two_char_tops_codes()` (*pyrcs.other\_assets.Depots method*), 36  
`fetch_water_troughs()` (*pyrcs.other\_assets.Features method*), 43  
`fix_num_stanox()` (*in module pyrcs.utils*), 66

## G

`get_catalogue()` (*in module pyrcs.utils*), 64  
`get_category_menu()` (*in module pyrcs.utils*), 64  
`get_conn_mileages()` (*pyrcs.line\_data.ELRMileages method*), 26  
`get_keys_to_prefixes()` (*pyrcs.line\_data.LOR method*), 29  
`get_last_updated_date()` (*in module pyrcs.utils*), 63  
`get_lor_page_urls()` (*pyrcs.line\_data.LOR method*), 30  
`get_names_of_independent_lines()` (*pyrcs.line\_data.Electrification method*), 17  
`get_station_data_catalogue()` (*in module pyrcs.utils*), 65  
`get_track_diagrams_items()` (*in module pyrcs.utils*), 65

## H

`homepage_url()` (*in module pyrcs.utils*), 57

## I

`identify_multiple_measures()` (*pyrcs.line\_data.ELRMileages static method*), 22  
`is_str_float()` (*in module pyrcs.utils*), 67

## L

`line_data.*` (*module*), 34  
`line_names` (*module*), 27  
`LineData` (*class in pyrcs.line\_data*), 35  
`LineNames` (*class in pyrcs.line\_data*), 27  
`LocationIdentifiers` (*class in pyrcs.line\_data*), 9  
`LOR` (*class in pyrcs.line\_data*), 29  
`lor_codes` (*module*), 29

## M

`make_location_codes_dictionary()` (*pyrcs.line\_data.LocationIdentifiers method*), 14  
`mile_chain_to_nr_mileage()` (*in module pyrcs.utils*), 58

## N

`nr_mileage_num_to_str()` (*in module pyrcs.utils*), 59  
`nr_mileage_str_to_num()` (*in module pyrcs.utils*), 58  
`nr_mileage_to_mile_chain()` (*in module pyrcs.utils*), 58  
`nr_mileage_to_yards()` (*in module pyrcs.utils*), 59

## O

`other_assets.*` (*module*), 55  
`OtherAssets` (*class in pyrcs.other\_assets*), 55

## P

`parse_additional_note_page()` (*pyrcs.line\_data.LocationIdentifiers static method*), 10  
`parse_current_operator()` (*pyrcs.other\_assets.Stations static method*), 50  
`parse_date()` (*in module pyrcs.utils*), 62  
`parse_location_name()` (*in module pyrcs.utils*), 61  
`parse_mileage_data()` (*pyrcs.line\_data.ELRMileages method*), 22  
`parse_table()` (*in module pyrcs.utils*), 61  
`parse_tr()` (*in module pyrcs.utils*), 61  
`parse_tunnel_length()` (*pyrcs.other\_assets.Tunnels static method*), 51  
`parse_vulgar_fraction_in_length()` (*pyrcs.other\_assets.Features method*), 41  
`pyrcs._line_data` (*module*), 35  
`pyrcs._other_assets` (*module*), 55  
`pyrcs.updater` (*module*), 55  
`pyrcs.utils` (*module*), 56

## S

`search_conn()` (*pyrcs.line\_data.ELRMileages static method*), 26  
`shift_num_nr_mileage()` (*in module pyrcs.utils*), 60  
`signal_boxes` (*module*), 46  
`SignalBoxes` (*class in pyrcs.other\_assets*), 47  
`Stations` (*class in pyrcs.other\_assets*), 49  
`stations` (*module*), 49

## T

`track_diagrams` (*module*), 33  
`TrackDiagrams` (*class in pyrcs.line\_data*), 33  
`Tunnels` (*class in pyrcs.other\_assets*), 51  
`tunnels` (*module*), 51

## U

`update_backup_data()` (*in module pyrcs.updater*), 56  
`update_catalogue()` (*pyrcs.line\_data.LOR method*), 30  
`update_location_name_repl_dict()` (*in module pyrcs.utils*), 66

## V

`Viaducts` (*class in pyrcs.other\_assets*), 53  
`viaducts` (*module*), 53

## Y

`yards_to_nr_mileage()` (*in module pyrcs.utils*), 59  
`year_to_financial_year()` (*in module pyrcs.utils*), 60