Package 'BMRSr'

October 12, 2022

Type Package

Title Wrapper Functions to the 'BMRS API'

Version 1.0.3

Description A set of wrapper functions to better interact with the 'Balancing Mechanism Reporting System API' (<https://bmreports.com/>).

License GPL (>= 2)

Encoding UTF-8

LazyData true

Depends R (>= 2.10)

Imports httr, xml2, stringr, tibble, readr, methods, purrr, dplyr, rlang

RoxygenNote 7.1.1

URL https://bmrsr.arawles.co.uk/

Suggests covr, knitr, rmarkdown, ggplot2, tidyr, testthat

VignetteBuilder knitr

NeedsCompilation no

Author Adam Rawles [aut, cre]

Maintainer Adam Rawles <adamrawles@hotmail.co.uk>

Repository CRAN

Date/Publication 2021-06-14 14:00:02 UTC

R topics documented:

`build_b_call` *Create an API call for B-data flows*

Description

Create an API call for B-data flows

Usage

```
build_b_call(
  data_item,
  api_key,
  settlement_date = NULL,
  period = NULL,
  year = NULL,
  month = NULL,
  week = NULL,
  process_type = NULL,
  start_time = NULL,
  end_time = NULL,
  start_date = NULL,
  end_date = NULL,
  service_type = c("csv", "xml"),
  api_version = "v1",
  ...
)
```

Arguments

data_item	character string; the id of the B flow
api_key	character string; api key retrieved from the Elexon portal
settlement_date	character string; settlement date (automatically cleaned by format_date)
period	character string; settlement period
year	character string; year
month	character string; month
week	character string; week
process_type	character string; process type
start_time	character string; start time
end_time	character string; end time
start_date	character string; start date
end_date	character string; end date
service_type	character string; file format (csv or xml)
api_version	character string; version of the api to use (currently on v1)
...	additional parameters that will be appended onto the query string

Value

list; list with entries `url` for the call, `service_type` and `data_item`

See Also

Other call-building functions: `build_call()`, `build_legacy_call()`, `build_remit_call()`

Examples

```
## Not run:
    build_b_call(data_item = "B1730",
    api_key = "12345", settlement_date = "14-12-2016")

    build_b_call(data_item = "B1510",
    api_key = "12345", start_date = "01 Jan 2019",
    start_time = "00:00:00", end_date = "02 Jan 2019",
    end_time = "24:00:00", service_type = "csv")

## End(Not run)
```

build_call *Build an API call (uses the appropriate function based on the data item)*

Description

Build an API call (uses the appropriate function based on the data item)

Usage

```
build_call(
  data_item,
  api_key,
  service_type = c("csv", "xml"),
  api_version = "v1",
  warn = TRUE,
  ...
)
```

Arguments

data_item	character string; data item to be retrieved
api_key	character string; user's API key
service_type	character string; one of "csv" or "xml" to define return format
api_version	character string; API version to use - currently only on version 1
warn	logical; should you be warned if any of the parameters you've supplied may not be appropriate for that data item? Default is TRUE.
...	values to be passed to appropriate build_x_call function

Value

list; list with entries `url` for the call, `service_type` and `data_item`

See Also

`build_b_call()`

`build_remit_call()`

`build_legacy_call()`

Other call-building functions: `build_b_call()`, `build_legacy_call()`, `build_remit_call()`

Examples

```
build_call(data_item = "TEMP", api_key = "12345", from_date = "12 Jun 2018",
to_date = "13 Jun 2018", service_type = "csv")
build_call(data_item = "QAS", api_key = "12345",
settlement_date = "01 Jun 2019", service_type = "xml")
```

`build_legacy_call` *Create an API call for legacy data*

Description

Create an API call for legacy data

Usage

```
build_legacy_call(
  data_item,
  api_key,
  from_date = NULL,
  to_date = NULL,
  settlement_date = NULL,
  settlement_period = NULL,
  bm_unit_id = NULL,
  bm_unit_type = NULL,
  lead_party_name = NULL,
  ngc_bm_unit_name = NULL,
  from_cleared_date = NULL,
  to_cleared_date = NULL,
  is_two_day_window = NULL,
  from_datetime = NULL,
  to_datetime = NULL,
  from_settlement_date = NULL,
  to_settlement_date = NULL,
  period = NULL,
  fuel_type = NULL,
  balancing_service_volume = NULL,
  zone_identifier = NULL,
  start_time = NULL,
  end_time = NULL,
  trade_name = NULL,
  trade_type = NULL,
  api_version = "v1",
  service_type = "csv",
  ...
)
```

Arguments

`data_item`	character string; the id of the legacy data
`api_key`	character string; api key retrieved from the Elexon portal
`from_date`	character string; from date (automatically cleaned by format_date)
`to_date`	character string; to date (automatically cleaned by format_date)

`settlement_date`
: character string; settlement date (automatically cleaned by format_date)

`settlement_period`
: character string; settlement period

`bm_unit_id`
: character string; BM Unit ID

`bm_unit_type`
: character string; BM Unit type

`lead_party_name`
: character string; lead party name

`ngc_bm_unit_name`
: character string; NGC BM Unit name

`from_cleared_date`
: character string; from cleared date (automatically cleaned by format_date)

`to_cleared_date`
: character string; to cleared dat (automatically cleaned by format_date)

`is_two_day_window`
: character string; is two day window

`from_datetime`
: character string; from datetime

`to_datetime`
: character string; to datetime

`from_settlement_date`
: character string; from settlement date (automatically cleaned by format_date)

`to_settlement_date`
: character string; to settlement date (automatically cleaned by format_date)

`period`
: character string; period

`fuel_type`
: character string; fuel type

`balancing_service_volume`
: character string; balancing service volume

`zone_identifier`
: character string; zone identifier

`start_time`
: character string; start time

`end_time`
: character string; end time

`trade_name`
: character string; trade name

`trade_type`
: character string; trade type

`api_version`
: character string; version of the api to use (currently on v1)

`service_type`
: character string; file format (csv or xml)

`...`
: additional parameters that will be appended onto the query string

Value

list; list with entries `url` for the call, `service_type` and `data_item`

See Also

Other call-building functions: `build_b_call()`, `build_call()`, `build_remit_call()`

Examples

```
build_legacy_call(data_item = "FUELINST", api_key = "12345",
from_datetime = "14-12-201613:00:00", to_datetime = "14-12-201614:00:00")
build_legacy_call(data_item = "QAS", api_key = "12345",
settlement_date = "01 Jun 2019", service_type = "xml")
```

`build_remit_call` *Create an API call for REMIT flows*

Description

Create an API call for REMIT flows

Usage

```
build_remit_call(
  data_item,
  api_key,
  event_start = NULL,
  event_end = NULL,
  publication_from = NULL,
  publication_to = NULL,
  participant_id = NULL,
  asset_id = NULL,
  event_type = NULL,
  fuel_type = NULL,
  message_type = NULL,
  message_id = NULL,
  unavailability_type = NULL,
  active_flag = NULL,
  sequence_id = NULL,
  service_type = "xml",
  api_version = "v1",
  ...
)
```

Arguments

`data_item`	character string; the id of the REMIT flow
`api_key`	character string; api key retrieved from the Elexon portal
`event_start`	character string; event start (automatically cleaned by format_date)
`event_end`	character string; event end (automatically cleaned by format_date)
`publication_from`	character string; publication from (automatically cleaned by format_date)
`publication_to`	character string; publication to (automatically cleaned by format_date)

participant_id	character string; participant id
asset_id	character string; asset id
event_type	character string; event type
fuel_type	character string; fuel type
message_type	character string; message type
message_id	character string; message id
unavailability_type	character string; unavailability type
active_flag	character string; active flag
sequence_id	character string; sequence id
service_type	character string; file format (csv or xml)
api_version	character string; version of the api to use (currently on v1)
...	additional parameters that will be appended onto the query string

Value

list; list with entries `url` for the call, `service_type` and `data_item`

See Also

Other call-building functions: `build_b_call()`, `build_call()`, `build_legacy_call()`

Examples

```
build_remit_call(data_item = "MessageListRetrieval", api_key = "12345",
event_start = "14-12-2016", event_end = "15-12-2016")
build_remit_call(data_item = "MessageDetailRetrieval", api_key = "12345",
participant_id = 21, service_type = "xml")
```

---

change_parameter_name *Convert a parameter name to a different format*

Description

The names of the parameters that are used in the R functions do not perfectly correspond with the parameter name expected by the API. This function converts an argument parameter name (e.g. `settlement_date`) to the URL argument name (e.g. `SettlementDate`) or the other way around

Usage

```
change_parameter_name(
  parameter,
  from = c("argument", "url"),
  to = c("url", "argument")
)
```

Arguments

parameter	character; name of the parameter provided to the relevant build() function
from	character; one of "argument" or "url" depending on whether parameter is in the argument or URL format
to	character; one of "argument" or "url"

Value

character; name of the parameter used in the URL request or build() function. If no match is found, character(0)

---

check_data_item *Check the data item to ensure that it is a valid request*

Description

Check the data item to ensure that it is a valid request

Usage

```
check_data_item(
  data_item,
  type = c("any", "B Flow", "Legacy", "REMIT"),
  silent = FALSE
)
```

Arguments

data_item	character; the data item to check
type	character; the type of data_item - one of "B Flow", "Legacy", or "REMIT" or "any" for any type
silent	boolean; whether to show a warning if not a valid data item

Value

boolean: returns true if data_item is valid, false if it is not

Examples

```
check_data_item("B1720", "B Flow") #valid
check_data_item("B1720", "Legacy") #invalid - incorrect type
check_data_item("B1111", "REMIT") #invalid - incorrect data item and type
```

check_data_item_version

Check the data item to ensure that it is valid for the version specified

Description

Currently, "B1610" is the only data item that no longer supports v1 and equally is the only data item that supports v2.

Usage

```
check_data_item_version(data_item, version = 1, silent = TRUE)
```

Arguments

data_item	character; the data item to check
version	character/numeric; the API version, either as a number (e.g. 1) or as a case-insensitive string (e.g. "v1" or "V2"). Default is 1.
silent	boolean; whether to show a warning if that version is not valid for the provided data item. Default is TRUE.

Value

boolean; returns TRUE if data_item is valid for the provided version, FALSE if it is not

Examples

```
check_data_item_version("B1610", 1)
check_data_item_version("B1710", 1)
```

check_period

Check the the provided Settlement Period value is valid

Description

Currently accepted values for Settlement Period is 1-50 and *

Usage

```
check_period(period)
```

Arguments

period	numeric/character; value to check. Must be numeric and between 1 and 50 or a character that's "*"

Value

character; period as character

---

clean_date_columns *Reformat date, time, and datetime columns*

Description

Reformat date, time, and datetime columns

Usage

```
clean_date_columns(x)
```

Arguments

x	tibble/df; dataset with the columns to be formatted

Value

tibble/df; dataset with reformatted columns (if any needed reformatting)

Examples

```
generation_dataset_unclean <- as.data.frame(
apply(generation_dataset_example, 2, as.character)
) #Create a version of the example generation dataset with character columns
clean_date_columns(generation_dataset_unclean)
```

---

fix_all_parameters *Fixes multiple parameters*

Description

Provided with a list of `build_..._call()` parameters, this function will fix each one and return a new list with the fixed parameters. This is implemented by applying the `fix_parameter` function

Usage

```
fix_all_parameters(params = list())
```

Arguments

params	list; list of the params. Should have a name and a value

Value

list; list of the fixed parameters

See Also

fix_parameter

`fix_parameter` *Fixes parameters provided in the* `build_x_call()` *functions*

Description

Fixes parameters provided in the `build_x_call()` functions

Usage

```
fix_parameter(param, before = NULL, ...)
```

Arguments

`param`	list; named list with the parameter name and value (e.g. `list(settlement_date = "01/01/2020")`)
`before`	function; function to fix the parameter. `param` will be passed as the first argument to this function. Default NULL does nothing
`...`	additional arguments passed to the `before` function

Value

modified `param` object (if `before` isn't NULL)

See Also

fix_all_parameters

full_request *Create an API call, send the request and retrieve the results, and parse them*

Description

Create an API call, send the request and retrieve the results, and parse them

Usage

```
full_request(
  ...,
  get_params = list(),
  parse = TRUE,
  clean_dates = TRUE,
  rename = TRUE
)
```

Arguments

...	values to be passed to appropriate build_x_call function
get_params	list; parameters to be passed to the send_request function (which will pass those parameters to httr::get)
parse	boolean; whether the results should be parsed or returned as a response() object
clean_dates	boolean; whether the csv response columns should be cleaned (reformatted to be correct date/time/datetime)
rename	boolean; whether blank columns should be renamed (not always accurate)

Value

If parse == TRUE, a tibble if service_type = "csv", otherwise a list. If parse == FALSE, a response() object is returned

Examples

```
full_request(data_item = "B1730", api_key = "12345",
settlement_date = "14-12-2016", parse = TRUE, service_type = "xml")
```

`generation_dataset_example` *An example dataset from BMRS showing generation by fuel type.*

Description

A dataset containing UK generation by fuel type between 1 July 2019 and 3 July 2019 at half-hourly intervals.

Usage

```
generation_dataset_example
```

Format

A data frame with 8655 rows and 6 variables:

record_type data item

settlement_date Settlement Date of the observation

settlement_period Settlement Period of the observation

spot_time Spot Time of the observation; this is essentially an amalgamation of settlement_date and settlement_period

ccgt Generation from Combined Cycle Gas Turbines (MW)

oil Generation from oil (MW)

coal Generation from coal(MW)

nuclear Generation from nuclear (MW)

wind Generation from wind (MW)

ps Generation from pumped storage (MW)

npshyd Generation from hydro (non-pump storage; MW)

ocgt Generation from Open Cycle Gas Turbines (MW)

other Generation from other, not-listed sources (MW)

intfr Generation from the French interconnector (MW)

intirl Generation from the Northern Irish interconnector (MW)

intned Generation from the Dutch interconnector (MW)

intew Generation from the Irish interconnector (MW)

biomass Generation from biomass (MW)

intnem Generation from Belgian interconnector (MW)

Source

https://www.bmreports.com/bmrs/?q=help/about-us

get_cleaning_function *Get the cleaning function required for a parameter*

Description

Before a parameter can be added to a request, it often needs to be cleaned. This function returns the appropriate function for a parameter. Parameters can be supplied with their name used in the `build()` functions ("argument") or in the URL

Usage

```
get_cleaning_function(parameter, format = c("argument", "url"))
```

Arguments

`parameter`	character; name of the parameter. Either the parameter as it's passed to the `build()` functions or the name of the parameter in the URL depending on the value of `format`
`format`	character; what format is `parameter` in? One of "argument" (default) or "url"

Value

character; name of the cleaning function. If there is no associated cleaning function, then `NULL`

get_column_names *Get the column names for a returned CSV Legacy dataset*

Description

Get the column names for a returned CSV Legacy dataset

Usage

```
get_column_names(data_item)
```

Arguments

`data_item`	character string; data item for the dataset

Value

vector; a vector of character strings with the column headings

Examples

```
get_column_names("TEMP")
```

get_data_items *Get a vector containing all of the permissible data items*

Description

Get a vector containing all of the permissible data items

Usage

```
get_data_items(type = "any")
```

Arguments

type	character; parameter to return only data items of a specific type ("Legacy", "B Flow", "REMIT", or "any")

Value

vector; data items as character string

Examples

```
get_data_items()
```

get_data_item_type *Get the data item type of a data item*

Description

Get the data item type of a data item

Usage

```
get_data_item_type(data_item)
```

Arguments

data_item	character string; data item to be retrieved

Examples

```
get_data_item_type("TEMP")
```

get_function — *Get the correct function to create the API call depending on the data item*

Description

Get the correct function to create the API call depending on the data item

Usage

```
get_function(data_item)
```

Arguments

data_item	character string; data item to be retrieved

Value

function

Examples

```
get_function("TEMP")
```

get_parameters — *Get the required parameters for a data item*

Description

Get the required parameters for a data item

Usage

```
get_parameters(data_item)
```

Arguments

data_item	character; the data item to get the parameters for

Value

A list containing the named parameters required for that call

Examples

```
get_parameters("TEMP")
```

parse_clean_csv *Parse a 'clean' .csv response*

Description

Some .csv files are returned without the EOF tag and with only 1 line before the data. This function is used to parse these files, whereas the `parse_eof_csv()` function is used to parse those files with the EOF tag and junk lines.

Usage

```
parse_clean_csv(content)
```

Arguments

`content`	character; the original response object parsed as a single text string.

Value

tibble; a tibble containing the data in the .csv file

See Also

Other parsers: `parse_eof_csv()`

parse_eof_csv *Parse a .csv response with a EOF tag left in*

Description

Some .csv files returned from the API still have an EOF tag left at the bottom and contain 4 lines of nonsense. This function is used to parse these files, whereas the `parse_clean_csv()` function is used to parse .csv files without this tag and the junk lines.

Usage

```
parse_eof_csv(content)
```

Arguments

`content`	character; the original response object parsed as a single text string.

Value

tibble; a tibble containing the data in the .csv file

See Also

Other parsers: parse_clean_csv()

---

parse_response *Parse the results of a call*

Description

Parse the results of a call

Usage

```
parse_response(
  response,
  format = NULL,
  clean_dates = TRUE,
  rename = TRUE,
  warn_on_initial_parse = FALSE
)
```

Arguments

response	A response object returned from the API request
format	character string; NULL to use response service type or "csv" or "xml" to force that format
clean_dates	boolean; whether to clean date/time columns
rename	boolean; whether to rename column headings (they are usually blank from the API)
warn_on_initial_parse	logical; should warning messages be shown during the orignal attempt at parsing the response? The default is FALSE as many of the data items need further cleaning and so the warning messages from the original attempt to parse the file are uninformative.

Value

A tibble if format == "csv", otherwise a list

Examples

```
list_example <- parse_response(
send_request(
build_call("TEMP", api_key = "12345", from_date = "01 Jun 2019",
to_date = "10 Jun 2019", service_type = "xml")
), "xml")
```

send_request — *Send an API request (basically a wrapper to httr:GET that adds a marker for the data item)*

Description

Send an API request (basically a wrapper to httr:GET that adds a marker for the data item)

Usage

```
send_request(request, config_options = list())
```

Arguments

request	list; a named list with at least a url to be sent and the data item contained within (most easily generated from build_call())
config_options	list; a named list of config options to be passed to httr::GET

Value

A response() object with an added data_item attribute

Examples

```
send_request(
build_call(data_item = "TEMP", from_date = "01 Jun 2019", to_date = "10 Jun 2019", api_key = "test")
)
```

try_parse — *Wrapper to the tryCatch version to be used for the parsing function*

Description

This simple wrapper returns an empty tibble on error and returns a custom warning message.

Usage

```
try_parse(expr, error_message, ...)
```

Arguments

expr	expression; expression to be evaluated for errors
error_message	character; character string to be displayed as a warning on error
...	extra parameters to be passed to the tryCatch() function.

Value

evaluated expression on success or empty tibble on error

Index