Package: patentsview (via r-universe)

December 15, 2024

Type Package

Title An R Client to the 'PatentsView' API

Version 0.3.0

Encoding UTF-8

Description Provides functions to simplify the 'PatentsView' API (<https://patentsview.org/apis/purpose>) query language, send GET and POST requests to the API's twenty seven endpoints, and parse the data that comes back.

URL https://docs.ropensci.org/patentsview/index.html

BugReports https://github.com/ropensci/patentsview/issues

License MIT + file LICENSE

LazyData TRUE

Depends R (>= 3.1)

Imports httr, lifecycle, jsonlite, utils

Suggests knitr, rmarkdown, testthat, tidyr

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

Config/pak/sysreqs libssl-dev

Repository https://ropensci.r-universe.dev

RemoteUrl https://github.com/ropensci/patentsview

RemoteRef master

RemoteSha 1972fca3063386b9242db162adee21774ab79b53

Contents

`cast_pv_data` *Cast PatentsView data*

Description

This will cast the data fields returned by `search_pv` so that they have their most appropriate data types (e.g., date, numeric, etc.).

Usage

```
cast_pv_data(data)
```

Arguments

`data`	The data returned by `search_pv`. This is the first element of the three-element result object you got back from `search_pv`. It should be a list of length 1, with one data frame inside it. See examples.

Value

The same type of object that you passed into `cast_pv_data`.

Examples

```
## Not run:

fields <- c("patent_date", "patent_title", "patent_year")
res <- search_pv(query = "{\"patent_number\":\"5116621\"}", fields = fields)
cast_pv_data(data = res$data)

## End(Not run)
```

fieldsdf *Fields data frame*

Description

A data frame containing the names of retrievable fields for each of the endpoints. You can find this data on the API's online documentation for each endpoint as well (e.g., the patents endpoint field list table).

Usage

```
fieldsdf
```

Format

A data frame with the following columns:

endpoint The endpoint that this field record is for

field The complete name of the field, including the parent group if applicable

data_type The field's input data type

group The group the field belongs to

common_name The field name without the parent group structure

get_endpoints *Get endpoints*

Description

This function reminds the user what the possible PatentsView API endpoints are.

Usage

```
get_endpoints()
```

Value

A character vector with the names of each endpoint.

`get_fields` *Get list of retrievable fields*

Description

This function returns a vector of fields that you can retrieve from a given API endpoint (i.e., the fields you can pass to the `fields` argument in `search_pv`). You can limit these fields to only cover certain entity group(s) as well (which is recommended, given the large number of possible fields for each endpoint).

Usage

```
get_fields(endpoint, groups = NULL)
```

Arguments

`endpoint`	The API endpoint whose field list you want to get. See `get_endpoints` for a list of the 27 endpoints.
`groups`	A character vector giving the group(s) whose fields you want returned. A value of `NULL` indicates that you want all of the endpoint's fields (i.e., do not filter the field list based on group membership). See the field tables located online to see which groups you can specify for a given endpoint (e.g., the patent endpoint table), or use the `fieldsdf` table (e.g., `unique(fieldsdf[fieldsdf$endpoint == "patent", "group"])`).

Value

A character vector with field names.

Examples

```
# Get all assignee-level fields for the patent endpoint:
fields <- get_fields(endpoint = "patent", groups = "assignees")

# ...Then pass to search_pv:
## Not run:

search_pv(
  query = '{"_gte":{"patent_date":"2007-01-04"}}',
  fields = fields
)

## End(Not run)
# Get all patent and assignee-level fields for the patent endpoint:
fields <- get_fields(endpoint = "patent", groups = c("assignees", "patents"))

## Not run:
# ...Then pass to search_pv:
```

```
search_pv(
  query = '{"_gte":{"patent_date":"2007-01-04"}}',
  fields = fields
)

## End(Not run)
```

`get_ok_pk` *Get OK primary key*

Description

This function suggests a value that you could use for the pk argument in `unnest_pv_data`, based on the endpoint you searched. It will return a potential unique identifier for a given entity (i.e., a given endpoint). For example, it will return "patent_id" when `endpoint = "patent"`.

Usage

```
get_ok_pk(endpoint)
```

Arguments

`endpoint`	The endpoint which you would like to know a potential primary key for.

Value

The name of a primary key (pk) that you could pass to `unnest_pv_data`.

Examples

```
get_ok_pk(endpoint = "inventor")
get_ok_pk(endpoint = "cpc_subclass")
get_ok_pk("publication/rel_app_text")
```

`qry_funs` *List of query functions*

Description

A list of functions that make it easy to write PatentsView queries. See the details section below for a list of the 14 functions, as well as the writing queries vignette for further details.

Usage

```
qry_funs
```

Format

An object of class `list` of length 14.

Details

1. Comparison operator functions

There are 6 comparison operator functions that work with fields of type integer, float, date, or string:

- `eq` - Equal to
- `neq` - Not equal to
- `gt` - Greater than
- `gte` - Greater than or equal to
- `lt` - Less than
- `lte` - Less than or equal to

There are 2 comparison operator functions that only work with fields of type string:

- `begins` - The string begins with the value string
- `contains` - The string contains the value string

There are 3 comparison operator functions that only work with fields of type fulltext:

- `text_all` - The text contains all the words in the value string
- `text_any` - The text contains any of the words in the value string
- `text_phrase` - The text contains the exact phrase of the value string

2. Array functions

There are 2 array functions:

- `and` - Both members of the array must be true
- `or` - Only one member of the array must be true

3. Negation function

There is 1 negation function:

- `not` - The comparison is not true

Value

An object of class `pv_query`. This is basically just a simple list with a print method attached to it.

Examples

```
qry_funs$eq(patent_date = "2001-01-01")

qry_funs$not(qry_funs$eq(patent_date = "2001-01-01"))
```

`retrieve_linked_data` *Get Linked Data*

Description

Some of the endpoints now return HATEOAS style links to get more data. E.g., the inventors endpoint may return a link such as: "https://search.patentsview.org/api/v1/inventor/252373/"

Usage

```
retrieve_linked_data(url, api_key = Sys.getenv("PATENTSVIEW_API_KEY"), ...)
```

Arguments

`url`	The link that was returned by the API on a previous call.
`api_key`	API key. See Here for info on creating a key.
`...`	Arguments passed along to httr's `GET` or `POST` function.

Value

A list with the following three elements:

data A list with one element - a named data frame containing the data returned by the server. Each row in the data frame corresponds to a single value for the primary entity. For example, if you search the assignees endpoint, then the data frame will be on the assignee-level, where each row corresponds to a single assignee. Fields that are not on the assignee-level would be returned in list columns.

query_results Entity counts across all pages of output (not just the page returned to you).

request Details of the HTTP request that was sent to the server. When you set `all_pages = TRUE`, you will only get a sample request. In other words, you will not be given multiple requests for the multiple calls that were made to the server (one for each page of results).

Examples

```
## Not run:

retrieve_linked_data(
  "https://search.patentsview.org/api/v1/cpc_subgroup/G01S7:4811/"
 )

## End(Not run)
```

search_pv *Search PatentsView*

Description

This function makes an HTTP request to the PatentsView API for data matching the user's query.

Usage

```
search_pv(
  query,
  fields = NULL,
  endpoint = "patent",
  subent_cnts = FALSE,
  mtchd_subent_only = lifecycle::deprecated(),
  page = 1,
  per_page = 1000,
  all_pages = FALSE,
  sort = NULL,
  method = "GET",
  error_browser = NULL,
  api_key = Sys.getenv("PATENTSVIEW_API_KEY"),
  ...
)
```

Arguments

query	The query that the API will use to filter records. query can come in any one of the following forms: • A character string with valid JSON. E.g., '{"_gte":{"patent_date":"2007-01-04"}}' • A list which will be converted to JSON by search_pv. E.g., list("_gte" = list("patent_date" = "2007-01-04")) • An object of class pv_query, which you create by calling one of the functions found in the qry_funs list...See the writing queries vignette for details. E.g., qry_funs$gte(patent_date = "2007-01-04")
fields	A character vector of the fields that you want returned to you. A value of NULL indicates that the default fields should be returned. Acceptable fields for a given endpoint can be found at the API's online documentation (e.g., check out the field list for the patents endpoint) or by viewing the fieldsdf data frame (View(fieldsdf)). You can also use get_fields to list out the fields available for a given endpoint.
endpoint	The web service resource you wish to search. Use get_endpoints() to list the available endpoints.

`subent_cnts` **[Deprecated]** Non-matched subentities will always be returned under the new version of the API

`mtchd_subent_only` **[Deprecated]** This is always FALSE in the new version of the API.

`page` The page number of the results that should be returned.

`per_page` The number of records that should be returned per page. This value can be as high as 1,000 (e.g., `per_page = 1000`).

`all_pages` Do you want to download all possible pages of output? If `all_pages = TRUE`, the values of `page` and `per_page` are ignored.

`sort` A named character vector where the name indicates the field to sort by and the value indicates the direction of sorting (direction should be either "asc" or "desc"). For example, `sort = c("patent_number" = "asc")` or `sort = c("patent_number" = "asc", "patent_date" = "desc")`. `sort = NULL` (the default) means do not sort the results. You must include any fields that you wish to sort by in `fields`.

`method` The HTTP method that you want to use to send the request. Possible values include "GET" or "POST". Use the POST method when your query is very long (say, over 2,000 characters in length).

`error_browser` **[Deprecated]**

`api_key` API key. See Here for info on creating a key.

`...` Arguments passed along to httr's `GET` or `POST` function.

Value

A list with the following three elements:

data A list with one element - a named data frame containing the data returned by the server. Each row in the data frame corresponds to a single value for the primary entity. For example, if you search the assignees endpoint, then the data frame will be on the assignee-level, where each row corresponds to a single assignee. Fields that are not on the assignee-level would be returned in list columns.

query_results Entity counts across all pages of output (not just the page returned to you).

request Details of the HTTP request that was sent to the server. When you set `all_pages = TRUE`, you will only get a sample request. In other words, you will not be given multiple requests for the multiple calls that were made to the server (one for each page of results).

Examples

```
## Not run:

search_pv(query = '{"_gt":{"patent_year":2010}}')

search_pv(
  query = qry_funs$gt(patent_year = 2010),
  fields = get_fields("patent", c("patents", "assignees"))
)
```

```
search_pv(
  query = qry_funs$gt(patent_year = 2010),
  method = "POST",
  fields = "patent_number",
  sort = c("patent_number" = "asc")
)

search_pv(
  query = qry_funs$eq(inventor_name_last = "Crew"),
  endpoint = "inventor",
  all_pages = TRUE
)

search_pv(
  query = qry_funs$contains(assignee_individual_name_last = "Smith"),
  endpoint = "assignee"
)

search_pv(
  query = qry_funs$contains(inventors_at_grant.name_last = "Smith"),
  endpoint = "patent",
  config = httr::timeout(40)
)

## End(Not run)
```

`unnest_pv_data` *Unnest PatentsView data*

Description

This function converts a single data frame that has subentity-level list columns in it into multiple data frames, one for each entity/subentity. The multiple data frames can be merged together using the primary key variable specified by the user (see the relational data chapter in "R for Data Science" for an in-depth introduction to joining tabular data).

Usage

```
unnest_pv_data(data, pk = get_ok_pk(names(data)))
```

Arguments

`data`	The data returned by `search_pv`. This is the first element of the three-element result object you got back from `search_pv`. It should be a list of length 1, with one data frame inside it. See examples.
`pk`	The column/field name that will link the data frames together. This should be the unique identifier for the primary entity. For example, if you used the patents endpoint in your call to `search_pv`, you could specify `pk = "patent_number"`.

This identifier has to have been included in your `fields` vector when you called `search_pv`. You can use `get_ok_pk` to suggest a potential primary key for your data.

Value

A list with multiple data frames, one for each entity/subentity. Each data frame will have the `pk` column in it, so you can link the tables together as needed.

Examples

```
## Not run:

fields <- c("patent_id", "patent_title", "inventors.inventor_city", "inventors.inventor_country")
res <- search_pv(query = '{"_gte":{"patent_year":2015}}', fields = fields)
unnest_pv_data(data = res$data, pk = "patent_id")

## End(Not run)
```

`with_qfuns` *With qry_funs*

Description

This function evaluates whatever code you pass to it in the environment of the `qry_funs` list. This allows you to cut down on typing when writing your queries. If you want to cut down on typing even more, you can try assigning the `qry_funs` list into your global environment with: `list2env(qry_funs, envir = globalenv())`.

Usage

```
with_qfuns(code, envir = parent.frame())
```

Arguments

`code`	Code to evaluate. See example.
`envir`	Where should R look for objects present in `code` that aren't present in `qry_funs`.

Value

The result of `code` - i.e., your query.

Examples

```
# Without with_qfuns, we have to do:
qry_funs$and(
  qry_funs$gte(patent_date = "2007-01-01"),
  qry_funs$text_phrase(patent_abstract = c("computer program")),
  qry_funs$or(
    qry_funs$eq(inventor_last_name = "ihaka"),
    qry_funs$eq(inventor_first_name = "chris")
  )
)

#...With it, this becomes:
with_qfuns(
 and(
   gte(patent_date = "2007-01-01"),
   text_phrase(patent_abstract = c("computer program")),
   or(
     eq(inventor_last_name = "ihaka"),
     eq(inventor_first_name = "chris")
   )
 )
)
```

Index