Package: patentsview (via r-universe)

October 29, 2026

Type Package

Title An R Client to the 'PatentsView' API

Version 1.0.0

Encoding UTF-8

Description Provides functions to simplify the 'PatentsView' API (<https://search.patentsview.org/docs/docs/Search%20API/SearchAPIReference/#api-query-language>) 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 (>= 4.1.0)

Imports httr2, lifecycle, jsonlite, utils, stats

Suggests dplyr, knitr, rmarkdown, testthat (>= 3.0.0), tidyr, vcr (>= 1.6.0)

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/Needs/website visNetwork, igraph, highcharter, dplyr, DT, magrittr, stringr

Config/pak/sysreqs libssl-dev

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

Date/Publication 2026-02-27 00:02:32 UTC

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

RemoteRef HEAD

RemoteSha c4e3b48943a715f2feb04c2e7bdeb81805879b39

Contents

fieldsdf 2
get_endpoints 2
get_fields 3
get_ok_pk 4
qry_funs 4
retrieve_linked_data 6
search_pv 6
unnest_pv_data 10
with_qfuns 11

Index **12**

`fieldsdf` *Fields data frame*

Description

A data frame containing the names of retrievable fields for each of the endpoints.

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 data type

group The group the field belongs to

`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

Get 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). Use the `fieldsdf` table (e.g., `unique(fieldsdf[fieldsdf$endpoint == "patent", "group"])`) to see which groups you can specify for a given endpoint.

Value

A character vector with field names.

Examples

```
# Get all top level (non-nested) fields for the patent endpoint:
fields <- get_fields(endpoint = "patent", groups = "patents")

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

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

## End(Not run)
# Get 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 column(s) that you could use for the pk argument in `unnest_pv_data`, based on the endpoint you searched. It will return a potential primary key - either a single column or a composite set of columns - for the endpoint.

Usage

```
get_ok_pk(endpoint)
```

Arguments

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

Value

The column names that represent a single row for the given endpoint.

Examples

```
get_ok_pk(endpoint = "inventor")
get_ok_pk(endpoint = "patent/foreign_citation")
```

`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 *Retrieve Linked Data*

Description

Some of the endpoints now return HATEOAS style links to get more data. E.g., the patent endpoint may return a link such as: "https://search.patentsview.org/api/v1/inventor/fl:th_ln:jefferson-1/". Use this function to fetch details from those links.

Usage

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

Arguments

url	A link that was returned by the API on a previous call.
api_key	API key, it defaults to Sys.getenv("PATENTSVIEW_API_KEY"). Request a key here.
...	Curl options passed along to httr2's req_options when we do GETs or POSTs.

Examples

```
## Not run:

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

## End(Not run)
```

search_pv *Search PatentsView*

Description

This 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 = lifecycle::deprecated(),
  mtchd_subent_only = lifecycle::deprecated(),
```

```
  page = lifecycle::deprecated(),
  per_page = lifecycle::deprecated(),
  size = 1000,
  after = NULL,
  all_pages = FALSE,
  sort = NULL,
  method = "GET",
  error_browser = lifecycle::deprecated(),
  exclude_withdrawn = 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 to the API that it should return the default fields for that endpoint. Acceptable fields for a given endpoint can be found in the fieldsdf data frame (View(fieldsdf)) or by using get_fields. Nested fields can be fully qualified, e.g., "application.filing_date" or you can use the group name that the field belongs to if you want all of the nested fields for that group. Note: The primary key columns for a given endpoint will be appended to your list of fields within search_pv. You can see the get_ok_pk to determine what those columns will be for a given endpoint. Note: If you specify all fields in a given group using their full qualified names, the group name will be substituted in the HTTTP request. This helps make HTTP requests shorter. This substitution will not happen when you specify all of the primary-entity fields (e.g., passing get_fields("patent", "patents") into search_pv would not substitute use the group name "patents" in place of all of the fields).
endpoint	The web service resource you wish to search. Use get_endpoints() to list the available endpoints.
subent_cnts	**[Deprecated]** This is always FALSE in the new version of the API as the total counts of unique subentities is no longer available.
mtchd_subent_only	**[Deprecated]** This is always FALSE in the new version of the API as non-matched subentities will always be returned.

page **[Deprecated]** The new version of the API does not use page as a parameter for paging, it uses after.

per_page **[Deprecated]** The API now uses size

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

after A list of sort key values that defaults to NULL. This exposes the API's paging parameter for users who want to implement their own paging. It cannot be set when all_pages = TRUE as the R package manipulates it for users automatically. See result set paging

all_pages Do you want to download all possible pages of output? If all_pages = TRUE, the value of size is 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_id" = "asc") or sort = c("patent_id" = "asc", "patent_date" = "desc"). sort = NULL (the default) means the API will use the default sort column for your given endpoint. 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 'r lifecycle::badge("deprecated"

exclude_withdrawn
only used by the patent endpoint, if FALSE withdrawn patents in the database can be returned by a query. The API defaults this to TRUE, not returning withdrawn patents in the database that met the query parameter.

api_key API key, it defaults to Sys.getenv("PATENTSVIEW_API_KEY"). Request a key here.

... Curl options passed along to httr2's req_options when we do GETs or POSTs.

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, as defined by the endpoint's primary key. For example, if you search the assignee endpoint, then the data frame will be on the assignee-level, where each row corresponds to a single assignee (primary key would be assignee_id). 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_id",
  sort = c("patent_id" = "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.inventor_name_last = "Smith"),
  endpoint = "patent",
  timeout = 40
)

search_pv(
  query = qry_funs$eq(patent_id = "11530080"),
  fields = "application"
)

search_pv(
  query = qry_funs$eq(patent_id = "9494444"),  # a withdrawn patent in the pview dbs
  fields = c("patent_id", "patent_date", "withdrawn"),
  exclude_withdrawn = FALSE
)

search_pv(
  query = qry_funs$eq(withdrawn = TRUE),
  fields = c("patent_id", "patent_date", "withdrawn"),
  exclude_withdrawn = FALSE
)

## 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 = lifecycle::deprecated())
```

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	**[Deprecated]**. should be the unique identifier for the primary entity. For example, if you used the patent endpoint in your call to search_pv, you could specify pk = "patent_id". **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)

## 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$eq(inventors.inventor_name_last = "Ihaka"),
    qry_funs$eq(inventors.inventor_name_last = "Chris")
  )
)

# ...With it, this becomes:
with_qfuns(
  and(
    gte(patent_date = "2007-01-01"),
    text_phrase(patent_abstract = c("computer program")),
    or(
      eq(inventors.inventor_name_last = "Ihaka"),
      eq(inventors.inventor_name_last = "Chris")
    )
  )
)
```

Index

∗ **datasets**
    fieldsdf, 2
    qry_funs, 4

fieldsdf, 2

get_endpoints, 2, *3*
get_fields, 3, *7*
get_ok_pk, 4, *7*, *10*

qry_funs, 4, *7*, *11*

req_options, *6*, *8*
retrieve_linked_data, 6

search_pv, *3*, 6, *10*

unnest_pv_data, *4*, 10

with_qfuns, 11