Package 'rplos'

October 14, 2022

Title Interface to the Search API for 'PLoS' Journals

Description A programmatic interface to the 'SOLR' based search API (<http://api.plos.org/>) provided by the Public Library of Science journals to search their articles. Functions are included for searching for articles, retrieving articles, making plots, doing 'faceted' searches, 'highlight' searches, and viewing results of 'highlighted' searches in a browser.

Version 1.0.0

License MIT + file LICENSE

URL https://docs.ropensci.org/rplos/ https://github.com/ropensci/rplos

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

LazyData true

VignetteBuilder knitr

Encoding UTF-8

Language en-US

Imports ggplot2, crul (>= 0.7.4), jsonlite, dplyr, plyr, lubridate, reshape2, whisker, solrium (>= 1.0.2)

Suggests xml2, knitr, rmarkdown, testthat, webmockr, vcr (>= 0.2.6)

RoxygenNote 7.1.1

X-schema.org-applicationCategory Literature

X-schema.org-keywords PLOS, library, science, JSON, XML, API, web, api-client, article, full text

X-schema.org-isPartOf https://ropensci.org

NeedsCompilation no

Author Scott Chamberlain [aut, cre] (<https://orcid.org/0000-0003-1444-9135>), Carl Boettiger [aut], Karthik Ram [aut], rOpenSci [fnd] (https://ropensci.org)

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2021-02-23 20:50:02 UTC

R topics documented:

`facetplos` *Do faceted searches on PLOS Journals full-text content*

Description

Do faceted searches on PLOS Journals full-text content

Usage

```
facetplos(
  q = "*:*",
  facet.query = NA,
  facet.field = NA,
  facet.prefix = NA,
  facet.sort = NA,
  facet.limit = NA,
  facet.offset = NA,
  facet.mincount = NA,
  facet.missing = NA,
  facet.method = NA,
```

```
  facet.enum.cache.minDf = NA,
  facet.threads = NA,
  facet.date = NA,
  facet.date.start = NA,
  facet.date.end = NA,
  facet.date.gap = NA,
  facet.date.hardend = NA,
  facet.date.other = NA,
  facet.date.include = NA,
  facet.range = NA,
  facet.range.start = NA,
  facet.range.end = NA,
  facet.range.gap = NA,
  facet.range.hardend = NA,
  facet.range.other = NA,
  facet.range.include = NA,
  start = NA,
  rows = NA,
  url = NA,
  sleep = 6,
  errors = "simple",
  proxy = NULL,
  callopts = list(),
  ...
)
```

Arguments

q	Query terms.
facet.query	This param allows you to specify an arbitrary query in the Lucene default syntax to generate a facet count. By default, faceting returns a count of the unique terms for a "field", while facet.query allows you to determine counts for arbitrary terms or expressions. This parameter can be specified multiple times to indicate that multiple queries should be used as separate facet constraints. It can be particularly useful for numeric range based facets, or prefix based facets – see example below (i.e. price:[* TO 500] and price:[501 TO *]).
facet.field	This param allows you to specify a field which should be treated as a facet. It will iterate over each Term in the field and generate a facet count using that Term as the constraint. This parameter can be specified multiple times to indicate multiple facet fields. None of the other params in this section will have any effect without specifying at least one field name using this param.
facet.prefix	Limits the terms on which to facet to those starting with the given string prefix. Note that unlike fq, this does not change the search results – it merely reduces the facet values returned to those beginning with the specified prefix. This parameter can be specified on a per field basis.
facet.sort	See solr_facet.
facet.limit	This param indicates the maximum number of constraint counts that should be

returned for the facet fields. A negative value means unlimited. Default: 100. Can be specified on a per field basis.

`facet.offset` This param indicates an offset into the list of constraints to allow paging. Default: 0. This parameter can be specified on a per field basis.

`facet.mincount` This param indicates the minimum counts for facet fields should be included in the response. Default: 0. This parameter can be specified on a per field basis.

`facet.missing` Set to "true" this param indicates that in addition to the Term based constraints of a facet field, a count of all matching results which have no value for the field should be computed. Default: FALSE. This parameter can be specified on a per field basis.

`facet.method` See `solr_facet`.

`facet.enum.cache.minDf`

This param indicates the minimum document frequency (number of documents matching a term) for which the filterCache should be used when determining the constraint count for that term. This is only used when facet.method=enum method of faceting. A value greater than zero will decrease memory usage of the filterCache, but increase the query time. When faceting on a field with a very large number of terms, and you wish to decrease memory usage, try a low value of 25 to 50 first. Default: 0, causing the filterCache to be used for all terms in the field. This parameter can be specified on a per field basis.

`facet.threads` This param will cause loading the underlying fields used in faceting to be executed in parallel with the number of threads specified. Specify as facet.threads=# where # is the maximum number of threads used. Omitting this parameter or specifying the thread count as 0 will not spawn any threads just as before. Specifying a negative number of threads will spin up to Integer.MAX_VALUE threads. Currently this is limited to the fields, range and query facets are not yet supported. In at least one case this has reduced warmup times from 20 seconds to under 5 seconds.

`facet.date` Specify names of fields (of type DateField) which should be treated as date facets. Can be specified multiple times to indicate multiple date facet fields.

`facet.date.start`

The lower bound for the first date range for all Date Faceting on this field. This should be a single date expression which may use the DateMathParser syntax. Can be specified on a per field basis.

`facet.date.end` The minimum upper bound for the last date range for all Date Faceting on this field (see facet.date.hardend for an explanation of what the actual end value may be greater). This should be a single date expression which may use the DateMathParser syntax. Can be specified on a per field basis.

`facet.date.gap` The size of each date range expressed as an interval to be added to the lower bound using the DateMathParser syntax. Eg: facet.date.gap=+1DAY. Can be specified on a per field basis.

`facet.date.hardend`

A Boolean parameter instructing Solr what to do in the event that facet.date.gap does not divide evenly between facet.date.start and facet.date.end. If this is true, the last date range constraint will have an upper bound of facet.date.end; if false, the last date range will have the smallest possible upper bound greater

then facet.date.end such that the range is exactly facet.date.gap wide. Default: FALSE. This parameter can be specified on a per field basis.

`facet.date.other`
: See `solr_facet`.

`facet.date.include`
: See `solr_facet`.

`facet.range`
: Indicates what field to create range facets for. Example: facet.range=price&facet.range=age

`facet.range.start`
: The lower bound of the ranges. Can be specified on a per field basis. Example: f.price.facet.range.start=0.0&f.age.facet.range.start=10

`facet.range.end`
: The upper bound of the ranges. Can be specified on a per field basis. Example: f.price.facet.range.end=1000.0&f.age.facet.range.start=99

`facet.range.gap`
: The size of each range expressed as a value to be added to the lower bound. For date fields, this should be expressed using the DateMathParser syntax. (ie: facet.range.gap=+1DAY). Can be specified on a per field basis. Example: f.price.facet.range.gap=100&f.a

`facet.range.hardend`
: A Boolean parameter instructing Solr what to do in the event that facet.range.gap does not divide evenly between facet.range.start and facet.range.end. If this is true, the last range constraint will have an upper bound of facet.range.end; if false, the last range will have the smallest possible upper bound greater then facet.range.end such that the range is exactly facet.range.gap wide. Default: FALSE. This parameter can be specified on a per field basis.

`facet.range.other`
: See `solr_facet`.

`facet.range.include`
: See `solr_facet`.

`start`
: Record to start at, default to beginning.

`rows`
: Number of records to return.

`url`
: URL endpoint

`sleep`
: Number of seconds to wait between requests. No need to use this for a single call. However, if you are doing many calls in a loop or lapply type call, sleep parameter is used to prevent your IP address from being blocked. You can only do 10 requests per minute, so one request every 6 seconds is about right.

`errors`
: (character) One of simple or complete. Simple gives http code and error message on an error, while complete gives both http code and error message, and stack trace, if available.

`proxy`
: List of arguments for a proxy connection, including one or more of: url, port, username, password, and auth. See `proxy` for help, which is used to construct the proxy connection.

`callopts`
: Further args passed on to `HttpClient`

`...`
: Further args to `solr_facet`

Value

A list

Examples

```
## Not run:
# Facet on a single field
facetplos(q='*:*', facet.field='journal')
facetplos(q='alcohol', facet.field='article_type')

# Facet on multiple fields
facetplos(q='alcohol', facet.field=c('journal','subject'))

# Using mincount
facetplos(q='alcohol', facet.field='journal', facet.mincount='500')

# Using facet.query to get counts
## A single facet.query term
facetplos(q='*:*', facet.field='journal', facet.query='cell')
## Many facet.query terms
facetplos(q='*:*', facet.field='journal', facet.query='cell,bird')

# Range faceting
facetplos(q='*:*', url=url, facet.range='counter_total_all',
   facet.range.start=5, facet.range.end=1000, facet.range.gap=10)
facetplos(q='alcohol', facet.range='alm_facebookCount', facet.range.start=1000,
   facet.range.end=5000, facet.range.gap = 100)

# Range faceting with > 1 field, same settings
facetplos(q='*:*', url=url, facet.range=c('counter_total_all','alm_twitterCount'),
 facet.range.start=5, facet.range.end=1000, facet.range.gap=10)

# Range faceting with > 1 field, different settings
facetplos(q='*:*', url=url, facet.range=c('counter_total_all','alm_twitterCount'),
 f.counter_total_all.facet.range.start=5, f.counter_total_all.facet.range.end=1000,
 f.counter_total_all.facet.range.gap=10, f.alm_twitterCount.facet.range.start=5,
 f.alm_twitterCount.facet.range.end=1000, f.alm_twitterCount.facet.range.gap=10)

## End(Not run)
```

`full_text_urls` *Create urls for full text articles in PLOS journals.*

Description

Create urls for full text articles in PLOS journals.

Usage

```
full_text_urls(doi)
```

Arguments

doi	One or more doi's

Details

We give **NA** for DOIs that are for annotations. Those can easily be removed like `Filter(Negate(is.na), res)`

Value

One or more urls, same length as input vector of dois

Examples

```
## Not run:
full_text_urls(doi='10.1371/journal.pone.0086169')
full_text_urls(doi='10.1371/journal.pbio.1001845')
full_text_urls(doi=c('10.1371/journal.pone.0086169',
  '10.1371/journal.pbio.1001845'))

# contains some annotation DOIs
dois <- searchplos(q = "*:*", fq='doc_type:full', limit=20)$data$id
full_text_urls(dois)

# contains no annotation DOIs
dois <- searchplos(q = "*:*",
  fq=list('doc_type:full', 'article_type:"Research Article"'),
limit=20)$data$id
full_text_urls(dois)

## End(Not run)
```

---

highbrow *Browse highlighted fragments in your default browser.*

Description

Browse highlighted fragments in your default browser.

Usage

```
highbrow(input = NULL, output = NULL, browse = TRUE)
```

Arguments

input	Input, usually output from a call to highplos
output	Path and file name for output file. If NULL, a temp file is used.
browse	Browse file in your default browse immediately after file creation. If FALSE, the file is written, but not opened.

Examples

```
## Not run:
out <- highplos(q='alcohol', hl.fl = 'abstract', rows=10)
highbrow(out)

out <- highplos(q='alcohol', hl.fl = 'abstract', rows=100)
highbrow(out)

## End(Not run)
```

---

highplos *Do highlighted searches on PLOS Journals full-text content*

Description

Do highlighted searches on PLOS Journals full-text content

Usage

```
highplos(
  q,
  fl = NULL,
  fq = NULL,
  hl.fl = NULL,
  hl.snippets = NULL,
  hl.fragsize = NULL,
  hl.q = NULL,
  hl.mergeContiguous = NULL,
  hl.requireFieldMatch = NULL,
  hl.maxAnalyzedChars = NULL,
  hl.alternateField = NULL,
  hl.maxAlternateFieldLength = NULL,
  hl.preserveMulti = NULL,
  hl.maxMultiValuedToExamine = NULL,
  hl.maxMultiValuedToMatch = NULL,
  hl.formatter = NULL,
  hl.simple.pre = NULL,
  hl.simple.post = NULL,
  hl.fragmenter = NULL,
  hl.fragListBuilder = NULL,
  hl.fragmentsBuilder = NULL,
  hl.boundaryScanner = NULL,
  hl.bs.maxScan = NULL,
  hl.bs.chars = NULL,
  hl.bs.type = NULL,
  hl.bs.language = NULL,
  hl.bs.country = NULL,
```

```
  hl.useFastVectorHighlighter = NULL,
  hl.usePhraseHighlighter = NULL,
  hl.highlightMultiTerm = NULL,
  hl.regex.slop = NULL,
  hl.regex.pattern = NULL,
  hl.regex.maxAnalyzedChars = NULL,
  start = 0,
  rows = NULL,
  errors = "simple",
  proxy = NULL,
  callopts = list(),
  sleep = 6,
  ...
)
```

Arguments

q	Search terms (character). You can search on specific fields by doing 'field:your query'. For example, a real query on a specific field would be 'author:Smith'.
fl	Fields to return from search (character) [e.g., 'id,title'], any combination of search fields [type 'data(plosfields)', then 'plosfields'].
fq	List specific fields to filter the query on (if NA, all queried). The options for this parameter are the same as those for the fl parameter. Note that using this parameter doesn't influence the actual query, but is used to filter the resuls to a subset of those you want returned. For example, if you want full articles only, you can do 'doc_type:full'. In another example, if you want only results from the journal PLOS One, you can do 'journal_key:PLoSONE'. See journalnamekey() for journal abbreviations.
hl.fl	A comma-separated list of fields for which to generate highlighted snippets. If left blank, the fields highlighted for the LuceneQParser are the defaultSearch-Field (or the df param if used) and for the DisMax parser the qf fields are used. A '*' can be used to match field globs, e.g. 'text_*' or even '*' to highlight on all fields where highlighting is possible. When using '*', consider adding hl.requireFieldMatch=TRUE.
hl.snippets	Max no. of highlighted snippets to generate per field. Note: it is possible for any number of snippets from zero to this value to be generated. This parameter accepts per-field overrides. Default: 1.
hl.fragsize	The size, in characters, of the snippets (aka fragments) created by the highlighter. In the original Highlighter, "0" indicates that the whole field value should be used with no fragmenting. See https://cwiki.apache.org/confluence/display/solr/HighlightingPa for more info
hl.q	Set a query request to be highlighted. It overrides q parameter for highlighting. Solr query syntax is acceptable for this parameter.
hl.mergeContiguous	Collapse contiguous fragments into a single fragment. "true" indicates contiguous fragments will be collapsed into single fragment. This parameter accepts

per-field overrides. This parameter makes sense for the original Highlighter only. Default: FALSE.

`hl.requireFieldMatch`
: If TRUE, then a field will only be highlighted if the query matched in this particular field (normally, terms are highlighted in all requested fields regardless of which field matched the query). This only takes effect if "hl.usePhraseHighlighter" is TRUE. Default: FALSE.

`hl.maxAnalyzedChars`
: How many characters into a document to look for suitable snippets. This parameter makes sense for the original Highlighter only. Default: 51200. You can assign a large value to this parameter and use hl.fragsize=0 to return highlighting in large fields that have size greater than 51200 characters.

`hl.alternateField`
: If a snippet cannot be generated (due to no terms matching), you can specify a field to use as the fallback. This parameter accepts per-field overrides.

`hl.maxAlternateFieldLength`
: If hl.alternateField is specified, this parameter specifies the maximum number of characters of the field to return. Any value less than or equal to 0 means unlimited. Default: unlimited.

`hl.preserveMulti`
: Preserve order of values in a multiValued list. Default: FALSE.

`hl.maxMultiValuedToExamine`
: When highlighting a multiValued field, stop examining the individual entries after looking at this many of them. Will potentially return 0 snippets if this limit is reached before any snippets are found. If maxMultiValuedToMatch is also specified, whichever limit is hit first will terminate looking for more. Default: Integer.MAX_VALUE

`hl.maxMultiValuedToMatch`
: When highlighting a multiValued field, stop examining the individual entries after looking at this many matches are found. If maxMultiValuedToExamine is also specified, whichever limit is hit first will terminate looking for more. Default: Integer.MAX_VALUE

`hl.formatter` Specify a formatter for the highlight output. Currently the only legal value is "simple", which surrounds a highlighted term with a customizable pre- and post text snippet. This parameter accepts per-field overrides. This parameter makes sense for the original Highlighter only.

`hl.simple.pre` The text which appears before and after a highlighted term when using the simple formatter. This parameter accepts per-field overrides. The default values are "" and "" This parameter makes sense for the original Highlighter only. Use hl.tag.pre and hl.tag.post for FastVectorHighlighter (see example under hl.fragmentsBuilder)

`hl.simple.post` The text which appears before and after a highlighted term when using the simple formatter. This parameter accepts per-field overrides. The default values are "" and "" This parameter makes sense for the original Highlighter only. Use hl.tag.pre and hl.tag.post for FastVectorHighlighter (see example under hl.fragmentsBuilder)

`hl.fragmenter` Specify a text snippet generator for highlighted text. The standard fragmenter is gap (which is so called because it creates fixed-sized fragments with gaps for multi-valued fields). Another option is regex, which tries to create fragments that "look like" a certain regular expression. This parameter accepts per-field overrides. Default: "gap"

`hl.fragListBuilder`
Specify the name of SolrFragListBuilder. This parameter makes sense for FastVectorHighlighter only. To create a fragSize=0 with the FastVectorHighlighter, use the SingleFragListBuilder. This field supports per-field overrides.

`hl.fragmentsBuilder`
Specify the name of SolrFragmentsBuilder. This parameter makes sense for FastVectorHighlighter only.

`hl.boundaryScanner`
Configures how the boundaries of fragments are determined. By default, boundaries will split at the character level, creating a fragment such as "uick brown fox jumps over the la". Valid entries are breakIterator or simple, with breakIterator being the most commonly used. This parameter makes sense for FastVectorHighlighter only.

`hl.bs.maxScan` Specify the length of characters to be scanned by SimpleBoundaryScanner. Default: 10. This parameter makes sense for FastVectorHighlighter only.

`hl.bs.chars` Specify the boundary characters, used by SimpleBoundaryScanner. This parameter makes sense for FastVectorHighlighter only.

`hl.bs.type` Specify one of CHARACTER, WORD, SENTENCE and LINE, used by BreakIteratorBoundaryScanner. Default: WORD. This parameter makes sense for FastVectorHighlighter only.

`hl.bs.language` Specify the language for Locale that is used by BreakIteratorBoundaryScanner. This parameter makes sense for FastVectorHighlighter only. Valid entries take the form of ISO 639-1 strings.

`hl.bs.country` Specify the country for Locale that is used by BreakIteratorBoundaryScanner. This parameter makes sense for FastVectorHighlighter only. Valid entries take the form of ISO 3166-1 alpha-2 strings.

`hl.useFastVectorHighlighter`
Use FastVectorHighlighter. FastVectorHighlighter requires the field is termVectors=on, termPositions=on and termOffsets=on. This parameter accepts per-field overrides. Default: FALSE

`hl.usePhraseHighlighter`
Use SpanScorer to highlight phrase terms only when they appear within the query phrase in the document. Default: TRUE.

`hl.highlightMultiTerm`
If the SpanScorer is also being used, enables highlighting for range/wildcard/fuzzy/prefix queries. Default: FALSE. This parameter makes sense for the original Highlighter only.

`hl.regex.slop` Factor by which the regex fragmenter can stray from the ideal fragment size (given by hl.fragsize) to accomodate the regular expression. For instance, a slop of 0.2 with fragsize of 100 should yield fragments between 80 and 120 characters in length. It is usually good to provide a slightly smaller fragsize

when using the regex fragmenter. Default: .6. This parameter makes sense for the original Highlighter only.

`hl.regex.pattern`
: The regular expression for fragmenting. This could be used to extract sentences (see example solrconfig.xml) This parameter makes sense for the original Highlighter only.

`hl.regex.maxAnalyzedChars`
: Only analyze this many characters from a field when using the regex fragmenter (after which, the fragmenter produces fixed-sized fragments). Applying a complicated regex to a huge field is expensive. Default: 10000. This parameter makes sense for the original Highlighter only.

`start`
: Record to start at (used in combination with limit when you need to cycle through more results than the max allowed=1000)

`rows`
: Number of results to return (integer)

`errors`
: (character) One of simple or complete. Simple gives http code and error message on an error, while complete gives both http code and error message, and stack trace, if available.

`proxy`
: List of arguments for a proxy connection, including one or more of: url, port, username, password, and auth. See `proxy` for help, which is used to construct the proxy connection.

`callopts`
: Optional additional curl options passed to `HttpClient`

`sleep`
: Number of seconds to wait between requests. No need to use this for a single call to searchplos. However, if you are using searchplos in a loop or lapply type call, do sleep parameter is used to prevent your IP address from being blocked. You can only do 10 requests per minute, so one request every 6 seconds is about right.

`...`
: Further arguments passed on to solr_highlight

Value

A list.

Examples

```
## Not run:
highplos(q='alcohol', hl.fl = 'abstract', rows=10)
highplos(q='alcohol', hl.fl = c('abstract','title'), rows=10)
highplos(q='everything:"sports alcohol"~7', hl.fl='everything')
highplos(q='alcohol', hl.fl='abstract', hl.fragsize=20, rows=5)
highplos(q='alcohol', hl.fl='abstract', hl.snippets=5, rows=5)
highplos(q='alcohol', hl.fl='abstract', hl.snippets=5,
  hl.mergeContiguous='true', rows=5)
highplos(q='alcohol', hl.fl='abstract', hl.useFastVectorHighlighter='true',
  rows=5)
highplos(q='everything:"experiment"', fq='doc_type:full', rows=100,
  hl.fl = 'title')

## End(Not run)
```

isocodes *Country names and FIPS codes*

Description

Country names and FIPS codes

journalnamekey *Get short keys for journals to use in searching specific journals.*

Description

Get short keys for journals to use in searching specific journals.

Usage

```
journalnamekey(...)
```

Arguments

...	optional curl options passed to HttpClient

Value

(character) journal name keys

Examples

```
## Not run:
journalnamekey()

## End(Not run)
```

`plosabstract` *Search PLoS Journals abstracts.*

Description

Search PLoS Journals abstracts.

Usage

```
plosabstract(
  q = NULL,
  fl = "id",
  fq = NULL,
  sort = NULL,
  start = 0,
  limit = 10,
  sleep = 6,
  errors = "simple",
  proxy = NULL,
  callopts = NULL,
  progress = NULL,
  ...
)
```

Arguments

`q`	Search terms (character). You can search on specific fields by doing 'field:your query'. For example, a real query on a specific field would be 'author:Smith'.
`fl`	Fields to return from search (character) [e.g., 'id,title'], any combination of search fields (see the dataset `plosfields`)
`fq`	List specific fields to filter the query on (if NA, all queried). The options for this parameter are the same as those for the fl parameter. Note that using this parameter doesn't influence the actual query, but is used to filter the results to a subset of those you want returned. For example, if you want full articles only, you can do `'doc_type:full'`. In another example, if you want only results from the journal PLOS One, you can do `'journal_key:PLoSONE'`. See `journalnamekey` for journal abbreviations.
`sort`	Sort results according to a particular field, and specify ascending (asc) or descending (desc) after a space; see examples. For example, to sort the counter_total_all field in descending fashion, do sort='counter_total_all desc'
`start`	Record to start at (used in combination with limit when you need to cycle through more results than the max allowed=1000). See Pagination below
`limit`	Number of results to return (integer). Setting `limit=0` returns only metadata. See Pagination below

sleep	Number of seconds to wait between requests. No need to use this for a single call to searchplos. However, if you are using searchplos in a loop or lapply type call, do sleep parameter is used to prevent your IP address from being blocked. You can only do 10 requests per minute, so one request every 6 seconds is about right.
errors	(character) One of simple or complete. Simple gives http code and error message on an error, while complete gives both http code and error message, and stack trace, if available.
proxy	List of arguments for a proxy connection, including one or more of: url, port, username, password, and auth. See `proxy` for help, which is used to construct the proxy connection.
callopts	(list) optional curl options passed to `HttpClient`
progress	a function with logic for printing a progress bar for an HTTP request, ultimately passed down to **curl**. only supports `httr::progress()`
...	Additional Solr arguments

Details

Details:

Value

Abstract content, in addition to any other fields requested in a list.

Faceting

Read more about faceting here: urlhttp://wiki.apache.org/solr/SimpleFacetParameters

Website vs. API behavior

Don't be surprised if queries you perform in a scripting language, like using `rplos` in R, give different results than when searching for articles on the PLOS website. I am not sure what exact defaults they use on their website. There are a few things to consider. You can tweak which types of articles are returned: Try using the `article_type` filter in the `fq` parameter. For which journal to search, e.g., do `'journal_key:PLoSONE'`. See `journalnamekey()` for journal abbreviations.

Phrase searching

To search phrases, e.g., **synthetic biology** as a single item, rather than separate occurrences of **synthetic** and **biology**, simply put double quotes around the phrase. For example, to search for cases of **synthetic biology**, do `searchplos(q = '"synthetic biology"')`.

You can modify phrase searches as well. For example, `searchplos(q = '"synthetic biology" ~ 10')` asks for cases of **synthetic biology** within 10 words of each other. See examples.

Pagination

The searchplos function and the many functions that are wrappers around searchplos all do paginatino internally for you. That is, if you request for example, 2000 results, the max you can get in any one request is 1000, so we'll do two requests for you. And so on for larger requests.

You can always do your own paginatino by doing a lapply type call or a for loop to cycle through pages of results.

Examples

```
## Not run:
plosabstract(q = 'drosophila', fl='abstract', limit=10)
plosabstract(q = 'drosophila', fl=c('id','author'), limit = 5)
plosabstract(q = 'drosophila', fl='author', limit = 5)
plosabstract(q = 'drosophila', fl=c('id','author','title'), limit = 5)

## End(Not run)
```

---

plosauthor *Search PLoS Journals authors.*

Description

Search PLoS Journals authors.

Usage

```
plosauthor(
  q = NULL,
  fl = "id",
  fq = NULL,
  sort = NULL,
  start = 0,
  limit = 10,
  sleep = 6,
  errors = "simple",
  proxy = NULL,
  callopts = NULL,
  progress = NULL,
  ...
)
```

Arguments

q	Search terms (character). You can search on specific fields by doing 'field:your query'. For example, a real query on a specific field would be 'author:Smith'.
fl	Fields to return from search (character) [e.g., 'id,title'], any combination of search fields (see the dataset plosfields)

fq	List specific fields to filter the query on (if NA, all queried). The options for this parameter are the same as those for the fl parameter. Note that using this parameter doesn't influence the actual query, but is used to filter the results to a subset of those you want returned. For example, if you want full articles only, you can do 'doc_type:full'. In another example, if you want only results from the journal PLOS One, you can do 'journal_key:PLoSONE'. See journalnamekey for journal abbreviations.
sort	Sort results according to a particular field, and specify ascending (asc) or descending (desc) after a space; see examples. For example, to sort the counter_total_all field in descending fashion, do sort='counter_total_all desc'
start	Record to start at (used in combination with limit when you need to cycle through more results than the max allowed=1000). See Pagination below
limit	Number of results to return (integer). Setting limit=0 returns only metadata. See Pagination below
sleep	Number of seconds to wait between requests. No need to use this for a single call to searchplos. However, if you are using searchplos in a loop or lapply type call, do sleep parameter is used to prevent your IP address from being blocked. You can only do 10 requests per minute, so one request every 6 seconds is about right.
errors	(character) One of simple or complete. Simple gives http code and error message on an error, while complete gives both http code and error message, and stack trace, if available.
proxy	List of arguments for a proxy connection, including one or more of: url, port, username, password, and auth. See proxy for help, which is used to construct the proxy connection.
callopts	(list) optional curl options passed to HttpClient
progress	a function with logic for printing a progress bar for an HTTP request, ultimately passed down to **curl**. only supports httr::progress()
...	Additional Solr arguments

Details

Details:

Value

Author names, in addition to any other fields requested in a data.frame.

Faceting

Read more about faceting here: urlhttp://wiki.apache.org/solr/SimpleFacetParameters

Website vs. API behavior

Don't be surprised if queries you perform in a scripting language, like using rplos in R, give different results than when searching for articles on the PLOS website. I am not sure what exact defaults they use on their website. There are a few things to consider. You can tweak which types

of articles are returned: Try using the `article_type` filter in the `fq` parameter. For which journal to search, e.g., do `'journal_key:PLoSONE'`. See `journalnamekey()` for journal abbreviations.

Phrase searching

To search phrases, e.g., **synthetic biology** as a single item, rather than separate occurrences of **synthetic** and **biology**, simply put double quotes around the phrase. For example, to search for cases of **synthetic biology**, do `searchplos(q = '"synthetic biology"')`.

You can modify phrase searches as well. For example, `searchplos(q = '"synthetic biology" ~ 10')` asks for cases of **synthetic biology** within 10 words of each other. See examples.

Pagination

The `searchplos` function and the many functions that are wrappers around `searchplos` all do paginatino internally for you. That is, if you request for example, 2000 results, the max you can get in any one request is 1000, so we'll do two requests for you. And so on for larger requests.

You can always do your own paginatino by doing a lapply type call or a for loop to cycle through pages of results.

Examples

```
## Not run:
plosauthor('Smith', 'id', limit=50)
plosauthor(q='Smith', fl=c('id','author'), limit=10)

## End(Not run)
```

---

plosfields *PLoS API fields to use for searching/retreiving data.*

Description

PLoS API fields to use for searching/retreiving data.

---

plosfigtabcaps *Search PLoS Journals figure and table captions.*

Description

Search PLoS Journals figure and table captions.

Usage

```
plosfigtabcaps(
  q = NULL,
  fl = "id",
  fq = NULL,
  sort = NULL,
  start = 0,
  limit = 10,
  sleep = 6,
  errors = "simple",
  proxy = NULL,
  callopts = NULL,
  progress = NULL,
  ...
)
```

Arguments

q	Search terms (character). You can search on specific fields by doing 'field:your query'. For example, a real query on a specific field would be 'author:Smith'.
fl	Fields to return from search (character) [e.g., 'id,title'], any combination of search fields (see the dataset `plosfields`)
fq	List specific fields to filter the query on (if NA, all queried). The options for this parameter are the same as those for the fl parameter. Note that using this parameter doesn't influence the actual query, but is used to filter the results to a subset of those you want returned. For example, if you want full articles only, you can do `'doc_type:full'`. In another example, if you want only results from the journal PLOS One, you can do `'journal_key:PLoSONE'`. See `journalnamekey` for journal abbreviations.
sort	Sort results according to a particular field, and specify ascending (asc) or descending (desc) after a space; see examples. For example, to sort the counter_total_all field in descending fashion, do sort='counter_total_all desc'
start	Record to start at (used in combination with limit when you need to cycle through more results than the max allowed=1000). See Pagination below
limit	Number of results to return (integer). Setting `limit=0` returns only metadata. See Pagination below
sleep	Number of seconds to wait between requests. No need to use this for a single call to searchplos. However, if you are using searchplos in a loop or lapply type call, do sleep parameter is used to prevent your IP address from being blocked. You can only do 10 requests per minute, so one request every 6 seconds is about right.
errors	(character) One of simple or complete. Simple gives http code and error message on an error, while complete gives both http code and error message, and stack trace, if available.
proxy	List of arguments for a proxy connection, including one or more of: url, port, username, password, and auth. See `proxy` for help, which is used to construct the proxy connection.

callopts	(list) optional curl options passed to HttpClient
progress	a function with logic for printing a progress bar for an HTTP request, ultimately passed down to **curl**. only supports httr::progress()
...	Additional Solr arguments

Details

Details:

Value

fields that you specify to return in a data.frame, along with the DOI's found.

Faceting

Read more about faceting here: urlhttp://wiki.apache.org/solr/SimpleFacetParameters

Website vs. API behavior

Don't be surprised if queries you perform in a scripting language, like using rplos in R, give different results than when searching for articles on the PLOS website. I am not sure what exact defaults they use on their website. There are a few things to consider. You can tweak which types of articles are returned: Try using the article_type filter in the fq parameter. For which journal to search, e.g., do 'journal_key:PLoSONE'. See journalnamekey() for journal abbreviations.

Phrase searching

To search phrases, e.g., **synthetic biology** as a single item, rather than separate occurrences of **synthetic** and **biology**, simply put double quotes around the phrase. For example, to search for cases of **synthetic biology**, do searchplos(q = '"synthetic biology"').

You can modify phrase searches as well. For example, searchplos(q = '"synthetic biology" ~ 10') asks for cases of **synthetic biology** within 10 words of each other. See examples.

Pagination

The searchplos function and the many functions that are wrappers around searchplos all do paginatino internally for you. That is, if you request for example, 2000 results, the max you can get in any one request is 1000, so we'll do two requests for you. And so on for larger requests.

You can always do your own paginatino by doing a lapply type call or a for loop to cycle through pages of results.

Examples

```
## Not run:
plosfigtabcaps('ecology', 'id', limit=100)
plosfigtabcaps(q='ecology', fl='figure_table_caption', limit=10)

## End(Not run)
```

plossubject *Search PLoS Journals subjects.*

Description

Search PLoS Journals subjects.

Usage

```
plossubject(
  q = NULL,
  fl = "id",
  fq = NULL,
  sort = NULL,
  start = 0,
  limit = 10,
  sleep = 6,
  errors = "simple",
  proxy = NULL,
  callopts = NULL,
  progress = NULL,
  ...
)
```

Arguments

q	Search terms (character). You can search on specific fields by doing 'field:your query'. For example, a real query on a specific field would be 'author:Smith'.
fl	Fields to return from search (character) [e.g., 'id,title'], any combination of search fields (see the dataset plosfields)
fq	List specific fields to filter the query on (if NA, all queried). The options for this parameter are the same as those for the fl parameter. Note that using this parameter doesn't influence the actual query, but is used to filter the results to a subset of those you want returned. For example, if you want full articles only, you can do 'doc_type:full'. In another example, if you want only results from the journal PLOS One, you can do 'journal_key:PLoSONE'. See journalnamekey for journal abbreviations.
sort	Sort results according to a particular field, and specify ascending (asc) or descending (desc) after a space; see examples. For example, to sort the counter_total_all field in descending fashion, do sort='counter_total_all desc'
start	Record to start at (used in combination with limit when you need to cycle through more results than the max allowed=1000). See Pagination below
limit	Number of results to return (integer). Setting limit=0 returns only metadata. See Pagination below

sleep	Number of seconds to wait between requests. No need to use this for a single call to searchplos. However, if you are using searchplos in a loop or lapply type call, do sleep parameter is used to prevent your IP address from being blocked. You can only do 10 requests per minute, so one request every 6 seconds is about right.
errors	(character) One of simple or complete. Simple gives http code and error message on an error, while complete gives both http code and error message, and stack trace, if available.
proxy	List of arguments for a proxy connection, including one or more of: url, port, username, password, and auth. See proxy for help, which is used to construct the proxy connection.
callopts	(list) optional curl options passed to HttpClient
progress	a function with logic for printing a progress bar for an HTTP request, ultimately passed down to **curl**. only supports httr::progress()
...	Additional Solr arguments

Details

Details:

See https://journals.plos.org/plosone/browse for subject areas.

Value

Subject content, in addition to any other fields requested in a data.frame.

Faceting

Read more about faceting here: urlhttp://wiki.apache.org/solr/SimpleFacetParameters

Website vs. API behavior

Don't be surprised if queries you perform in a scripting language, like using rplos in R, give different results than when searching for articles on the PLOS website. I am not sure what exact defaults they use on their website. There are a few things to consider. You can tweak which types of articles are returned: Try using the article_type filter in the fq parameter. For which journal to search, e.g., do 'journal_key:PLoSONE'. See journalnamekey() for journal abbreviations.

Phrase searching

To search phrases, e.g., **synthetic biology** as a single item, rather than separate occurrences of **synthetic** and **biology**, simply put double quotes around the phrase. For example, to search for cases of **synthetic biology**, do searchplos(q = '"synthetic biology"').

You can modify phrase searches as well. For example, searchplos(q = '"synthetic biology" ~ 10') asks for cases of **synthetic biology** within 10 words of each other. See examples.

Pagination

The searchplos function and the many functions that are wrappers around searchplos all do paginatino internally for you. That is, if you request for example, 2000 results, the max you can get in any one request is 1000, so we'll do two requests for you. And so on for larger requests.

You can always do your own paginatino by doing a lapply type call or a for loop to cycle through pages of results.

Examples

```
## Not run:
plossubject('marine ecology', limit = 5)
plossubject(q='marine ecology',  fl = c('id','journal','title'), limit = 20)
plossubject(q='marine ecology', fl = c('id','journal'),
   fq='doc_type:full', limit = 9)
plossubject(q='marine ecology', fl = c('id','journal'),
   fq=list('doc_type:full','!article_type_facet:"Issue%20Image"'),
   limit = 9)

## End(Not run)
```

---

plostitle *Search PLoS Journals titles.*

Description

Search PLoS Journals titles.

Usage

```
plostitle(
  q = NULL,
  fl = "id",
  fq = NULL,
  sort = NULL,
  start = 0,
  limit = 10,
  sleep = 6,
  errors = "simple",
  proxy = NULL,
  callopts = NULL,
  progress = NULL,
  ...
)
```

Arguments

q	Search terms (character). You can search on specific fields by doing 'field:your query'. For example, a real query on a specific field would be 'author:Smith'.
fl	Fields to return from search (character) [e.g., 'id,title'], any combination of search fields (see the dataset plosfields)
fq	List specific fields to filter the query on (if NA, all queried). The options for this parameter are the same as those for the fl parameter. Note that using this parameter doesn't influence the actual query, but is used to filter the results to a subset of those you want returned. For example, if you want full articles only, you can do 'doc_type:full'. In another example, if you want only results from the journal PLOS One, you can do 'journal_key:PLoSONE'. See journalnamekey for journal abbreviations.
sort	Sort results according to a particular field, and specify ascending (asc) or descending (desc) after a space; see examples. For example, to sort the counter_total_all field in descending fashion, do sort='counter_total_all desc'
start	Record to start at (used in combination with limit when you need to cycle through more results than the max allowed=1000). See Pagination below
limit	Number of results to return (integer). Setting limit=0 returns only metadata. See Pagination below
sleep	Number of seconds to wait between requests. No need to use this for a single call to searchplos. However, if you are using searchplos in a loop or lapply type call, do sleep parameter is used to prevent your IP address from being blocked. You can only do 10 requests per minute, so one request every 6 seconds is about right.
errors	(character) One of simple or complete. Simple gives http code and error message on an error, while complete gives both http code and error message, and stack trace, if available.
proxy	List of arguments for a proxy connection, including one or more of: url, port, username, password, and auth. See proxy for help, which is used to construct the proxy connection.
callopts	(list) optional curl options passed to HttpClient
progress	a function with logic for printing a progress bar for an HTTP request, ultimately passed down to **curl**. only supports httr::progress()
...	Additional Solr arguments

Details

Details:

Value

Titles, in addition to any other fields requested in a data.frame.

Faceting

Read more about faceting here: urlhttp://wiki.apache.org/solr/SimpleFacetParameters

Website vs. API behavior

Don't be surprised if queries you perform in a scripting language, like using `rplos` in R, give different results than when searching for articles on the PLOS website. I am not sure what exact defaults they use on their website. There are a few things to consider. You can tweak which types of articles are returned: Try using the `article_type` filter in the `fq` parameter. For which journal to search, e.g., do `'journal_key:PLoSONE'`. See `journalnamekey()` for journal abbreviations.

Phrase searching

To search phrases, e.g., **synthetic biology** as a single item, rather than separate occurrences of **synthetic** and **biology**, simply put double quotes around the phrase. For example, to search for cases of **synthetic biology**, do `searchplos(q = '"synthetic biology"')`.

You can modify phrase searches as well. For example, `searchplos(q = '"synthetic biology" ~ 10')` asks for cases of **synthetic biology** within 10 words of each other. See examples.

Pagination

The `searchplos` function and the many functions that are wrappers around `searchplos` all do paginatino internally for you. That is, if you request for example, 2000 results, the max you can get in any one request is 1000, so we'll do two requests for you. And so on for larger requests.

You can always do your own paginatino by doing a lapply type call or a for loop to cycle through pages of results.

Examples

```
## Not run:
plostitle(q='drosophila', fl='title', limit=99)
plostitle(q='drosophila', fl=c('title','journal'), limit=10)
plostitle(q='drosophila',  limit = 5)

## End(Not run)
```

---

plosviews *Search PLoS Journals by article views.*

Description

Search PLoS Journals by article views.

Usage

```
plosviews(search, byfield = NULL, views = "alltime", limit = NULL, ...)
```

Arguments

search	search terms (character)
byfield	field to search by, e.g., subject, author, etc. (character)
views	views all time (alltime) or views last 30 days (last30) (character)
limit	number of results to return (integer)
...	Optional additional curl options passed to HttpClient

Examples

```
## Not run:
plosviews('10.1371/journal.pone.0002154', 'id', 'alltime')
plosviews('10.1371/journal.pone.0002154', 'id', 'last30')
plosviews('10.1371/journal.pone.0002154', 'id', 'alltime,last30')
plosviews(search='marine ecology', byfield='subject', limit=50)
plosviews(search='evolution', views = 'alltime', limit = 99)
plosviews('bird', views = 'alltime', limit = 99)

## End(Not run)
```

---

plosword *Search results on a keyword over all fields in PLoS Journals.*

Description

Search results on a keyword over all fields in PLoS Journals.

Usage

```
plosword(terms, vis = FALSE, ...)
```

Arguments

terms	search terms (character)
vis	visualize results in bar plot or not (TRUE or FALSE)
...	Optional additional curl options passed to HttpClient

Value

Number of search results (vis = FALSE), or number of search in a table and a histogram of results (vis = TRUE).

Examples

```
## Not run:
plosword('Helianthus')
plosword(list('monkey','replication','design','sunflower','whale'),
   vis = TRUE)

## End(Not run)
```

`plos_fulltext` *Get full text xml of PLOS papers given a DOI*

Description

Get full text xml of PLOS papers given a DOI

Usage

```
plos_fulltext(doi, ...)

## S3 method for class 'plosft'
print(x, ...)
```

Arguments

`doi`	One or more DOIs
`...`	Curl options passed on to `HttpClient`
`x`	Input to print method

Value

Character string of XML.

Examples

```
## Not run:
plos_fulltext(doi='10.1371/journal.pone.0086169')
plos_fulltext(c('10.1371/journal.pone.0086169',
  '10.1371/journal.pbio.1001845'))
dois <- searchplos(q = "*:*",
  fq = list('doc_type:full', 'article_type:"Research Article"'),
  limit = 3)$data$id
out <- plos_fulltext(dois)
out[dois[1]]
out[1:2]

# Extract text from the XML strings - xml2 package required
if (requireNamespace("xml2")) {
  library("xml2")
```

```
  lapply(out, function(x){
    tmp <- xml2::read_xml(x)
    xml2::xml_find_all(tmp, "//ref-list//ref")
  })
}


## End(Not run)
```

---

plot_throughtime *Plot results through time for serach results from PLoS Journals.*

Description

Plot results through time for serach results from PLoS Journals.

Usage

```
plot_throughtime(terms, limit = NA, ...)
```

Arguments

terms	search terms (character)
limit	number of results to return (integer)
...	optional curl options passed to HttpClient

Value

Number of search results (vis = FALSE), or number of search in a table and a histogram of results (vis = TRUE).

Examples

```
## Not run:
plot_throughtime(terms='phylogeny', limit=300)
plot_throughtime(list('drosophila','monkey'), 100)
plot_throughtime(list('drosophila','flower','dolphin','cell','cloud'), 100)

## End(Not run)
```

rplos *Connect with PLoS API data*

Description

rplos provides an R interface to the PLoS Search API. More information about each function can be found in its help documentation.

rplos functions

rplos functions make HTTP requests using the **crul** package, and parse json using the **jsonlite** package.

PLoS API key

You used to need an API key to use this package - no longer needed

Tutorials

See the rOpenSci website for a tutorial: https://ropensci.org/tutorials/rplos_tutorial.html

Throttling

Beware, PLOS recently has started throttling requests. That is, they will give error messages like "(503) Service Unavailable - The server cannot process the request due to a high load", which probably means you've done too many requests in a certain time period.

Here's what they say (http://api.plos.org/solr/faq/#solr_api_recommended_usage) on the matter:

"Please limit your API requests to 7200 requests a day, 300 per hour, 10 per minute and allow 5 seconds for your search to return results. If you exceed this threshold, we will lock out your IP address. If you're a high-volume user of the PLOS Search API and need more API requests a day, please contact us at api@plos.org to discuss your options. We currently limit API users to no more than five concurrent connections from a single IP address.""

Author(s)

Scott Chamberlain

Carl Boettiger <cboettig@gmail.com>

Karthik Ram <karthik.ram@gmail.com>

Examples

```
## Not run:
searchplos(q='ecology', fl=c('id','publication_date'), limit = 2)

# Get only full article DOIs
out <- searchplos(q="*:*", fl='id', fq='doc_type:full', start=0, limit=250)
head(out$data)
```

```
# Get DOIs for only PLoS One articles
out <- searchplos(q="*:*", fl='id', fq='journal_key:PLoSONE',
  start=0, limit=15)
head(out$data)

## End(Not run)
```

`rplos-defunct` *Defunct functions in rplos*

Description

- `crossref`: service no longer provided - see the package `rcrossref`
- `citations`: service no longer available

`searchplos` *Base function to search PLoS Journals*

Description

Base function to search PLoS Journals

Usage

```
searchplos(
  q = NULL,
  fl = "id",
  fq = NULL,
  sort = NULL,
  start = 0,
  limit = 10,
  sleep = 6,
  errors = "simple",
  proxy = NULL,
  callopts = list(),
  progress = NULL,
  ...
)
```

Arguments

q	Search terms (character). You can search on specific fields by doing 'field:your query'. For example, a real query on a specific field would be 'author:Smith'.
fl	Fields to return from search (character) [e.g., 'id,title'], any combination of search fields (see the dataset plosfields)
fq	List specific fields to filter the query on (if NA, all queried). The options for this parameter are the same as those for the fl parameter. Note that using this parameter doesn't influence the actual query, but is used to filter the results to a subset of those you want returned. For example, if you want full articles only, you can do 'doc_type:full'. In another example, if you want only results from the journal PLOS One, you can do 'journal_key:PLoSONE'. See journalnamekey for journal abbreviations.
sort	Sort results according to a particular field, and specify ascending (asc) or descending (desc) after a space; see examples. For example, to sort the counter_total_all field in descending fashion, do sort='counter_total_all desc'
start	Record to start at (used in combination with limit when you need to cycle through more results than the max allowed=1000). See Pagination below
limit	Number of results to return (integer). Setting limit=0 returns only metadata. See Pagination below
sleep	Number of seconds to wait between requests. No need to use this for a single call to searchplos. However, if you are using searchplos in a loop or lapply type call, do sleep parameter is used to prevent your IP address from being blocked. You can only do 10 requests per minute, so one request every 6 seconds is about right.
errors	(character) One of simple or complete. Simple gives http code and error message on an error, while complete gives both http code and error message, and stack trace, if available.
proxy	List of arguments for a proxy connection, including one or more of: url, port, username, password, and auth. See proxy for help, which is used to construct the proxy connection.
callopts	(list) optional curl options passed to HttpClient
progress	a function with logic for printing a progress bar for an HTTP request, ultimately passed down to **curl**. only supports httr::progress()
...	Additional Solr arguments

Details

Details:

Value

An object of class "plos", with a list of length two, each element being a list itself.

Faceting

Read more about faceting here: urlhttp://wiki.apache.org/solr/SimpleFacetParameters

Website vs. API behavior

Don't be surprised if queries you perform in a scripting language, like using `rplos` in R, give different results than when searching for articles on the PLOS website. I am not sure what exact defaults they use on their website. There are a few things to consider. You can tweak which types of articles are returned: Try using the `article_type` filter in the `fq` parameter. For which journal to search, e.g., do `'journal_key:PLoSONE'`. See `journalnamekey()` for journal abbreviations.

Phrase searching

To search phrases, e.g., **synthetic biology** as a single item, rather than separate occurrences of **synthetic** and **biology**, simply put double quotes around the phrase. For example, to search for cases of **synthetic biology**, do `searchplos(q = '"synthetic biology"')`.

You can modify phrase searches as well. For example, `searchplos(q = '"synthetic biology" ~ 10')` asks for cases of **synthetic biology** within 10 words of each other. See examples.

Pagination

The `searchplos` function and the many functions that are wrappers around `searchplos` all do paginatino internally for you. That is, if you request for example, 2000 results, the max you can get in any one request is 1000, so we'll do two requests for you. And so on for larger requests.

You can always do your own paginatino by doing a lapply type call or a for loop to cycle through pages of results.

Examples

```
## Not run:
searchplos(q='ecology', fl=c('id','publication_date'), limit = 2)
searchplos('ecology', fl=c('id','publication_date'), limit = 2)
searchplos('ecology', c('id','title'), limit = 2)

# Get only full article DOIs
out <- searchplos(q="*:*", fl='id', fq='doc_type:full', start=0, limit=250)
head(out$data)

# Get DOIs for only PLoS One articles
out <- searchplos(q="*:*", fl='id', fq='journal_key:PLoSONE', start=0, limit=15)
out$data

# Get DOIs for full article in PLoS One
out <- searchplos(q="*:*", fl='id', fq=list('journal_key:PLoSONE',
   'doc_type:full'), limit=50)
out$data

# Serch for many q
q <- c('ecology','evolution','science')
lapply(q, function(x) searchplos(x, limit=2))

# Query to get some PLOS article-level metrics, notice difference between two outputs
out <- searchplos(q="*:*", fl=c('id','counter_total_all','alm_twitterCount'),fq='doc_type:full')
out_sorted <- searchplos(q="*:*", fl=c('id','counter_total_all','alm_twitterCount'),
```

```
   fq='doc_type:full', sort='counter_total_all desc')
out$data
out_sorted$data

# Show me all articles that have these two words less then about 15 words apart.
searchplos(q='everything:"sports alcohol"~15', fl='title', fq='doc_type:full')

# Now let's try to narrow our results to 7 words apart. Here I'm changing the ~15 to ~7
searchplos(q='everything:"sports alcohol"~7', fl='title', fq='doc_type:full')

# A list of articles about social networks that are popular on a social network
searchplos(q="*:*",fl=c('id','alm_twitterCount'),
  fq=list('doc_type:full','subject:"Social networks"','alm_twitterCount:[100 TO 10000]'),
   sort='counter_total_month desc')

# Now, lets also only look at articles that have seen some activity on twitter.
# Add "fq=alm_twitterCount:[1 TO *]" as a parameter within the fq argument.
searchplos(q='everything:"sports alcohol"~7', fl=c('alm_twitterCount','title'),
   fq=list('doc_type:full','alm_twitterCount:[1 TO *]'))
searchplos(q='everything:"sports alcohol"~7', fl=c('alm_twitterCount','title'),
   fq=list('doc_type:full','alm_twitterCount:[1 TO *]'),
   sort='counter_total_month desc')

# Return partial doc parts
## Return Abstracts only
out <- searchplos(q='*:*', fl=c('doc_partial_body','doc_partial_parent_id'),
   fq=list('doc_type:partial', 'doc_partial_type:Abstract'), limit=3)
## Return Title's only
out <- searchplos(q='*:*', fl=c('doc_partial_body','doc_partial_parent_id'),
   fq=list('doc_type:partial', 'doc_partial_type:Title'), limit=3)

# Remove DOIs for annotations (i.e., corrections)
searchplos(q='*:*', fl=c('id','article_type'),
   fq='-article_type:correction', limit=100)

# Remove DOIs for annotations (i.e., corrections) and Viewpoints articles
searchplos(q='*:*', fl=c('id','article_type'),
   fq=list('-article_type:correction','-article_type:viewpoints'), limit=100)

# Get eissn codes
searchplos(q='*:*', fl=c('id','journal','eissn','cross_published_journal_eissn'),
   fq="doc_type:full", limit = 60)

searchplos(q='*:*', fl=c('id','journal','eissn','cross_published_journal_eissn'),
   limit = 2000)

## End(Not run)
```

Index