Package: scopusflow (via r-universe)

June 22, 2026

Title A Reproducible Workflow Layer for 'Scopus' Bibliographic Searches

Version 0.3.0

Description A coherent, quota-aware workflow layer over the Elsevier 'Scopus' Search 'API' <https://dev.elsevier.com/sc_apis.html>. It builds reproducible search plans, retrieves records with rate-limit handling, retry with back-off and optional resumable caching, normalises results to a stable tidy schema, extracts and tracks changes in Digital Object Identifiers (DOIs), compares publication trends across topics and exports to formats compatible with downstream bibliometric tools. Network and 'API' errors are surfaced as typed conditions so that callers can respond to them programmatically. 'Scopus' is a trademark of Elsevier. This package is an independent client and is not affiliated with or endorsed by Elsevier.

License MIT + file LICENSE

URL https://github.com/pablobernabeu/scopusflow, https://pablobernabeu.github.io/scopusflow/

BugReports https://github.com/pablobernabeu/scopusflow/issues

Depends R (>= 4.1.0)

Imports cli, httr2 (>= 1.0.0), jsonlite, rlang (>= 1.0.0), stats, tibble, tools, utils

Suggests bslib, callr, fansi, ggplot2, knitr, rmarkdown, shiny, spelling, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Language en-GB

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libssl-dev

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

Date/Publication 2026-06-22 19:58:01 UTC

RemoteUrl https://github.com/pablobernabeu/scopusflow

RemoteRef HEAD

RemoteSha 57c8da1a14fa9cd0cdc7d04d2362f62024a4b9ef

Contents

as_bibliometrix . . . 3
as_bibtex . . . 4
as_tibble.scopus_records . . . 5
example_records . . . 6
plot_scopus_comparison . . . 7
plot_scopus_top . . . 8
plot_scopus_trend . . . 9
run_app . . . 10
scopus_abstract . . . 11
scopus_cache_clear . . . 12
scopus_cache_dir . . . 12
scopus_combine . . . 13
scopus_compare_topics . . . 14
scopus_count . . . 15
scopus_diff_dois . . . 16
scopus_extract_dois . . . 17
scopus_fetch . . . 18
scopus_fetch_plan . . . 19
scopus_field_tags . . . 21
scopus_has_key . . . 21
scopus_plan . . . 22
scopus_query . . . 23
scopus_quota . . . 24
scopus_top . . . 25
scopus_trend . . . 26
summary.scopus_records . . . 27
write_scopus_records . . . 28

Index **29**

`as_bibliometrix` *Convert records to a bibliometrix-compatible data frame*

Description

Re-maps a scopus_records tibble to the tagged column layout used by the **bibliometrix** package (and the wider ISI/Web of Science convention), so results can flow into downstream science-mapping workflows.

Usage

```
as_bibliometrix(x)
```

Arguments

`x`	A scopus_records tibble.

Details

This produces the *shape* bibliometrix expects from the core descriptive fields. It reconstructs only what the 'Scopus' Search API returns, so richer fields that some bibliometrix analyses use, such as full author affiliations or cited references, are left out. To obtain those, export a full 'Scopus' CSV or BibTeX file from the web interface and read it with `bibliometrix::convert2df()`.

Value

A data frame (classed `bibliometrixDB`) with the standard tag columns `AU` (authors), `TI` (title), `SO` (source or publication), `DI` (DOI), `PY` (publication year), `TC` (times cited), `UT` (record id) and `DB` (`"SCOPUS"`). Character tag fields are upper-cased to match the bibliometrix convention.

Examples

```
recs <- scopus_records(list(entry = list(
  list(`dc:identifier` = "SCOPUS_ID:1", `prism:doi` = "10.1/a",
       `dc:title` = "A study", `dc:creator` = "Doe J.",
       `prism:publicationName` = "Journal", `prism:coverDate` = "2020-01-01",
       `citedby-count` = "3")
)))
as_bibliometrix(recs)
```

`as_bibtex` *Export records to BibTeX or RIS*

Description

Turns a scopus_records set into a BibTeX or RIS string, the interchange formats that reference managers (Zotero, EndNote, Mendeley) and LaTeX bibliographies import. Each record becomes one entry, with its authors split out and the 'Scopus' identifier kept as a note. Records are treated as journal articles, the dominant 'Scopus' content type. BibTeX citation keys are made unique within the export, and special characters are escaped.

Usage

```
as_bibtex(x, file = NULL)

as_ris(x, file = NULL)
```

Arguments

`x`	A scopus_records tibble.
`file`	Optional path to write to. With the default `NULL` the formatted string is returned; with a path it is written there and returned invisibly. Nothing is written unless a path is given.

Value

A length-one character string of the formatted records (returned invisibly when `file` is supplied).

See Also

`as_bibliometrix()`, `write_scopus_records()`, `scopus_extract_dois()`

Examples

```
cat(substr(as_bibtex(example_records), 1, 200))
cat(substr(as_ris(example_records), 1, 200))
```

`as_tibble.scopus_records` *Normalise raw 'Scopus' entries to a stable tidy schema*

Description

Converts the nested list returned by the 'Scopus' Search API into a flat, predictable tibble with one row per record. This shape is the common currency of the package. Both `scopus_fetch()` and `scopus_fetch_plan()` return it, and the DOI, comparison and export helpers all consume it.

Usage

```
## S3 method for class 'scopus_records'
as_tibble(x, ...)

## S3 method for class 'scopus_records'
as.data.frame(x, ...)

## S3 method for class 'scopus_records'
autoplot(object, ...)

scopus_records(x, query = NA_character_)

is_scopus_records(x)
```

Arguments

`x`	An object to test.
`...`	Ignored, for S3 compatibility.
`object`	A scopus_records object (for the `autoplot()` method).
`query`	Optional character scalar recording the query that produced the entries, kept in the `query` column for provenance.

Details

The 'Scopus' API signals an empty result set with a single sentinel entry that carries an `error` field and no identifier. This is detected and turned into a zero-row result rather than a spurious record, while a genuine record that also carries a per-entry `error` annotation is kept.

Value

The coercion methods return a plain tibble or data frame with the same columns and the `scopus_records` class removed.

The `autoplot()` method returns a ggplot2::ggplot of the records per year.

A tibble of class `scopus_records` with the columns `entry_number` (integer), `scopus_id` (character), `doi` (character), `title` (character), `authors` (character, the creator names joined with `"; `

" when several are listed), year (integer, the leading four digits of the cover date), date (character, the ISO cover date), publication (character, the source title), citations (integer) and query (character). A missing field becomes NA, and an empty result set yields a zero-row tibble with the same columns.

is_scopus_records() returns a length-one logical.

Examples

```
# A minimal entry as the API would return it.
raw <- list(entry = list(
  list(
    `dc:identifier` = "SCOPUS_ID:1",
    `prism:doi` = "10.1000/abc",
    `dc:title` = "An example",
    `dc:creator` = "Doe J.",
    `prism:publicationName` = "Journal of Examples",
    `prism:coverDate` = "2020-05-01",
    `citedby-count` = "7"
  )
))
scopus_records(raw, query = "TITLE(example)")
```

---

example_records *Example set of normalised 'Scopus' records*

Description

A small set of six illustrative records spanning several disciplines (genome editing, machine learning, climate science, materials, oncology and physics), in the shape that scopus_fetch() returns. It is provided so that the package can be explored and its examples run without a 'Scopus' API key. The records were normalised from the static page fixture bundled in inst/extdata.

Usage

```
example_records
```

Format

A scopus_records tibble with six rows and the standard schema:

entry_number Position within the retrieval.

scopus_id The 'Scopus' record identifier.

doi Digital Object Identifier.

title Document title.

authors First or corresponding author.

year Publication year.

date Cover date in ISO form.

publication Source title.

citations Citation count.

query The query that produced the record.

Source

Synthetic example data, illustrative only.

Examples

```
example_records
summary(example_records)
```

`plot_scopus_comparison` *Plot a topic comparison*

Description

Draws a line chart of each comparison topic's share of the reference literature over time, from the output of `scopus_compare_topics()`. The chart uses integer year breaks, a colour-blind-safe palette and, for a handful of topics, labels the lines directly so the reader need not consult a legend. Shaded bands convey how stable each yearly share is.

Usage

```
plot_scopus_comparison(
  x,
  pub_count_in_legend = TRUE,
  highlight = NULL,
  interval = TRUE,
  ...
)

## S3 method for class 'scopus_comparison'
autoplot(object, ...)
```

Arguments

`x`	A `scopus_comparison` object from `scopus_compare_topics()`.
`pub_count_in_legend`	Logical. When `TRUE` (the default), each topic's label carries its total record count, for example `effect size (n = 1,204)`.
`highlight`	Optional character scalar naming one comparison topic to draw the eye to. The named topic is drawn in an accent colour, and the others in grey, which is useful when one topic is the focus of a figure.

`interval`	Logical. When `TRUE` (the default), a shaded band around each line shows a Wilson interval on the yearly share. See *Details* for how to read it.
`...`	Currently unused, present for S3 consistency.
`object`	A `scopus_comparison` object (for the `autoplot()` method).

Details

This needs the suggested package **ggplot2** and raises an informative error when it is absent. The chart shows the comparison topics alone, since the reference is the 100% denominator against which they are measured. A year for which the reference has no records carries no defined share and is omitted, which is noted in the caption.

The shaded band is a Wilson score interval computed from the comparison count and the reference count for each year. 'Scopus' returns exact counts rather than a sample, so the band is not a confidence interval in the inferential sense. It is best read as an illustrative stability range: it is wide where the reference set for a year is small, and so the share would move easily, and narrow where the reference set is large. It says nothing about query wording, indexing lag or coverage, which are the larger real uncertainties.

Value

A ggplot2::ggplot object. Printing it draws the plot.

See Also

`scopus_compare_topics()`

Examples

```
cmp <- tibble::tibble(
  query = "q", query_type = "comparison",
  abridged_query = rep(c("computer vision", "drug discovery"), each = 4),
  year = rep(2017:2020, 2), n = c(220, 280, 360, 430, 30, 55, 90, 150),
  reference_n = rep(1500, 8),
  comparison_percentage = c(14.7, 18.7, 24, 28.7, 2, 3.7, 6, 10),
  average_comparison_percentage = rep(c(21.5, 5.4), each = 4)
)
class(cmp) <- c("scopus_comparison", class(cmp))
plot_scopus_comparison(cmp)
plot_scopus_comparison(cmp, highlight = "drug discovery")
```

---

plot_scopus_top *Plot the most frequent values in a record set*

Description

Draws a horizontal bar chart from the output of `scopus_top()`.

Usage

```
plot_scopus_top(x, ...)

## S3 method for class 'scopus_top'
autoplot(object, ...)
```

Arguments

x	A scopus_top object from scopus_top().
...	Currently unused, present for S3 consistency.
object	A scopus_top object (for the autoplot() method).

Value

A ggplot2::ggplot object. Needs the suggested package **ggplot2**.

See Also

scopus_top()

Examples

```
plot_scopus_top(scopus_top(example_records, by = "source"))
```

---

plot_scopus_trend *Plot a publication trend*

Description

Draws annual record counts over time from the output of scopus_trend().

Usage

```
plot_scopus_trend(x, ...)

## S3 method for class 'scopus_trend'
autoplot(object, ...)
```

Arguments

x	A scopus_trend object from scopus_trend().
...	Currently unused, present for S3 consistency.
object	A scopus_trend object (for the autoplot() method).

Value

A ggplot2::ggplot object. Needs the suggested package **ggplot2**.

See Also

`scopus_trend()`

Examples

```
tr <- tibble::tibble(query = "q", year = 2015:2020, n = c(120, 180, 240, 310, 400, 520))
class(tr) <- c("scopus_trend", class(tr))
plot_scopus_trend(tr)
```

---

run_app *Launch the scopusflow app*

Description

Starts a local, code-free Shiny app for building a search, retrieving records, comparing topic trends and exporting the results, with a live terminal that streams the retrieval's progress and a panel that mirrors every choice as runnable R code. A *Demo mode* (on by default) synthesises records and a topic comparison so the whole workflow can be explored with no key and no network; switch it off and supply a key to query 'Scopus' for real. The app runs on your own machine: your API key never leaves it, and requests originate from your own network, which is what the 'Scopus' API expects. It needs the suggested packages **shiny**, **bslib** and **callr** (and **ggplot2** for the plots, **fansi** for coloured terminal output).

Usage

```
run_app(host = "127.0.0.1", port = NULL, launch.browser = TRUE)
```

Arguments

`host`	The address to listen on. Defaults to `"127.0.0.1"`, so the app is reachable only from your own machine and the key is never exposed on the network.
`port`	The port to listen on, or `NULL` (default) to choose one.
`launch.browser`	Logical, whether to open a browser. Passed to `shiny::runApp()`.

Value

Called for its side effect of running the app; does not return until the app is closed.

See Also

`scopus_plan()`, `scopus_fetch_plan()`

Examples

```
## Not run:
run_app()

## End(Not run)
```

`scopus_abstract` *Retrieve abstracts and richer metadata*

Description

Fetches the abstract text and core metadata for one or more records from the Elsevier 'Scopus' Abstract Retrieval API. This complements the Search API used elsewhere in the package: a search returns many records with a few fields each, whereas this returns the fuller record, including the abstract, for a known identifier.

Usage

```
scopus_abstract(
  ids,
  by = c("doi", "scopus_id"),
  api_key = NULL,
  inst_token = NULL,
  verbose = FALSE
)
```

Arguments

`ids`	Character vector of identifiers to look up, either Digital Object Identifiers or 'Scopus' record identifiers (with or without the `"SCOPUS_ID:"` prefix), according to `by`.
`by`	Either `"doi"` or `"scopus_id"`, the kind of identifier in `ids`.
`api_key`, `inst_token`	Optional credentials (see `scopus_has_key()`).
`verbose`	Logical. When `TRUE`, progress is reported.

Value

A tibble of class `scopus_abstracts`, one row per identifier, with columns `id` (the input identifier), `scopus_id`, `doi`, `title`, `abstract`, `publication`, `year` and `citations`. A field the API does not return is `NA`. An identifier that cannot be retrieved (for example one not in 'Scopus') yields a row of `NA`s with a warning, so a batch is not lost to a single failure.

API access

This performs one request per identifier and requires a valid API key and internet access; full-text abstract access can also depend on your entitlement. See the *API access* section of `scopus_count()` for the conditions that may be raised.

See Also

scopus_fetch(), scopus_extract_dois()

Examples

```
scopus_abstract("10.1038/s41586-019-0001-1")
```

---

scopus_cache_clear *Clear the scopusflow managed cache*

Description

Deletes the cache files written under scopus_cache_dir(). A cache you created in a directory of your own is left untouched.

Usage

```
scopus_cache_clear()
```

Value

Invisibly, TRUE once the managed cache directory is removed or found to be absent.

Examples

```
# Safe to call even when nothing is cached.
scopus_cache_clear()
```

---

scopus_cache_dir *Managed cache directory for scopusflow*

Description

Returns (and creates on request) a per-user cache directory under tools::R_user_dir(), suitable for passing to cache_dir in scopus_fetch_plan(). The cache is entirely optional and can be cleared with scopus_cache_clear().

Usage

```
scopus_cache_dir(create = FALSE)
```

Arguments

create	Logical. When TRUE, the directory is created if it is absent.

Value

The cache directory path, invisibly when `create = TRUE`.

Examples

```
scopus_cache_dir(create = FALSE)
```

---

scopus_combine *Combine record sets into one*

Description

Binds several scopus_records objects into a single one, renumbering `entry_number` across the result and, optionally, dropping duplicates. This is the safe way to merge separate fetches: plain `rbind()` would leave duplicate entry numbers, and `c()` would return a list.

Usage

```
scopus_combine(..., dedupe = FALSE)

## S3 method for class 'scopus_records'
c(x, ...)
```

Arguments

`...`	Two or more scopus_records objects, or a single list of them.
dedupe	Logical. When `TRUE`, records sharing a 'Scopus' identifier, or failing that a DOI (compared case-insensitively), are kept once.
x	A scopus_records object (for the `c()` method).

Value

A scopus_records tibble. Per-retrieval attributes such as `total_results` are not carried over, since they describe a single fetch.

See Also

`scopus_fetch_plan()`, which combines plan cells the same way.

Examples

```
# Merging a set with itself and de-duplicating recovers the distinct records.
scopus_combine(example_records, example_records, dedupe = TRUE)
```

`scopus_compare_topics` *Compare publication trends across topics*

Description

Compares how often a set of *comparison* topics co-occur with a *reference* topic over time. For each year and each comparison term, the number of records matching the reference combined with that term is expressed as a percentage of the records matching the reference alone. This reveals which sub-topics are growing or shrinking within a literature.

Usage

```
scopus_compare_topics(
  reference_query,
  comparison_terms,
  years,
  field = NULL,
  view = c("STANDARD", "COMPLETE"),
  api_key = NULL,
  inst_token = NULL,
  verbose = FALSE
)
```

Arguments

`reference_query`	Character scalar. The reference topic that anchors the comparison (for example `"language learning"`).
`comparison_terms`	Character vector of topics to compare against the reference (for example `c("effect size", "Bayesian")`). Each is combined with the reference using a logical AND.
`years`	Integer vector of publication years to span (for example `2015:2020`).
`field`	Optional 'Scopus' field tag applied to every component of every query (see `scopus_plan()`).
`view`	Either `"STANDARD"` or `"COMPLETE"`.
`api_key, inst_token`	Optional credentials (see `scopus_has_key()`).
`verbose`	Logical. When `TRUE`, progress is reported.

Value

A tibble of class `scopus_comparison` with the columns `query` (the full query used), `query_type` (`"reference"` or `"comparison"`), `abridged_query` (the topic label for plotting), `year`, `n` (records that year), `reference_n` (reference records that year), `comparison_percentage` (`100 * n / reference_n`, or `NA` when `reference_n` is 0) and `average_comparison_percentage` (the same ratio computed on period totals). Comparison rows are sorted by descending average percentage.

API access

This performs one count request per term per year, so it requires a valid API key and internet access. The *API access* section of `scopus_count()` gives the details. A modest number of terms and years keeps the call within quota.

See Also

`plot_scopus_comparison()` to visualise the result.

Examples

```
cmp <- scopus_compare_topics(
  reference_query = "deep learning",
  comparison_terms = c("computer vision", "drug discovery"),
  years = 2018:2022,
  field = "TITLE-ABS-KEY"
)
cmp
```

---

scopus_count *Count 'Scopus' results for a query*

Description

Retrieves only the total number of records matching a query, without downloading them. This is the inexpensive way to size a retrieval before committing quota. The count can guide how to partition a `scopus_plan()`, or simply report how large a topic is.

Usage

```
scopus_count(
  query,
  years = NULL,
  field = NULL,
  view = c("STANDARD", "COMPLETE"),
  api_key = NULL,
  inst_token = NULL
)
```

Arguments

`query`	Character scalar. The base search expression.
`years`	Optional integer vector of publication years to restrict to.
`field`	Optional 'Scopus' field tag to wrap the query in (see `scopus_plan()`).
`view`	Either `"STANDARD"` or `"COMPLETE"`.
`api_key, inst_token`	Optional credentials, resolved by default from options or environment variables (see `scopus_has_key()`).

Value

A single number giving the total number of matching records, or NA when the API reports no total. It is returned as a double so that very large totals are represented exactly rather than overflowing, with the parsed quota (see scopus_quota()) attached as the quota attribute so a workflow can pace itself off a count.

API access

This function performs a network request and therefore requires a valid API key and internet access. When no key is configured it raises a scopus_error_no_key condition, and other failures raise typed scopus_error subclasses such as scopus_error_rate_limit. A tryCatch() around the call lets a workflow handle these gracefully.

Examples

```
scopus_count("CRISPR", years = 2015:2020, field = "TITLE-ABS-KEY")
```

---

scopus_diff_dois *Compare two DOI retrievals*

Description

Identifies which DOIs were added, removed or unchanged between an earlier and a later retrieval. This supports change tracking: re-running a search later and seeing exactly what is new.

Usage

```
scopus_diff_dois(old, new)
```

Arguments

old, new	scopus_records objects or character vectors of DOIs, representing the earlier (old) and later (new) retrievals.

Value

A tibble of class scopus_doi_diff with columns doi and status, where status is an ordered factor with levels "added" (in new only), "removed" (in old only) and "unchanged" (in both). Rows are sorted by status then DOI, and printing shows the counts in each category.

See Also

scopus_extract_dois()

Examples

```
old <- c("10.1/a", "10.1/b")
new <- c("10.1/b", "10.1/c")
scopus_diff_dois(old, new)
```

---

scopus_extract_dois *Extract, clean and optionally export DOIs*

Description

Pulls Digital Object Identifiers from a scopus_records object (or a bare character vector), normalises them and removes missing values. The resulting list can be imported into a reference manager such as Zotero to assemble a bibliography.

Usage

```
scopus_extract_dois(x, dedupe = TRUE, file = NULL)
```

Arguments

x	A scopus_records tibble, or a character vector of DOIs.
dedupe	Logical, dropping duplicate DOIs by default.
file	Optional path at which to write the DOIs as a single-column CSV. A file is written only when this argument is supplied, and only to the exact path given, so the package always leaves the working directory untouched unless asked. Parent directories are assumed to exist already.

Details

Normalisation trims surrounding whitespace and strips common resolver prefixes (`https://doi.org/`, `http://dx.doi.org/`, `doi:`) so that the same article is counted once even when its DOI is formatted differently in two records. Because DOIs are case-insensitive, comparison and deduplication ignore case, while the output keeps the original casing.

Value

A character vector of cleaned DOIs, returned invisibly when `file` is written.

See Also

`scopus_diff_dois()` to compare two retrievals.

Examples

```
recs <- scopus_records(list(entry = list(
  list(`prism:doi` = "10.1/AAA"),
  list(`prism:doi` = "https://doi.org/10.1/aaa"),
  list(`prism:doi` = NULL)
)))
scopus_extract_dois(recs)

# Write to a temporary file (never the working directory).
path <- tempfile(fileext = ".csv")
scopus_extract_dois(recs, file = path)
```

`scopus_fetch` *Fetch 'Scopus' records for a query*

Description

Retrieves records page by page, accumulating them and returning a single normalised scopus_records tibble. Pagination, the API's hard `start` < `5000` ceiling, rate-limit handling and retry with back-off are all managed for you.

Usage

```
scopus_fetch(
  query,
  max_results = Inf,
  view = c("STANDARD", "COMPLETE"),
  page_size = NULL,
  field = NULL,
  years = NULL,
  cursor = FALSE,
  api_key = NULL,
  inst_token = NULL,
  verbose = FALSE
)
```

Arguments

`query`	Character scalar. The base search expression.
`max_results`	Maximum number of records to retrieve. Defaults to `Inf`, meaning all available records up to the API ceiling. With the default offset-based paging the 'Scopus' Search API refuses offsets of 5000 or more, so a single query yields at most 5000 records; set `cursor = TRUE`, or partition the search by year with `scopus_plan()`, to go beyond that.
`view`	Either `"STANDARD"` or `"COMPLETE"`.

page_size — Integer records per page, or NULL (default) to use the most quota-efficient page the view allows (200 for STANDARD, 25 for COMPLETE). See scopus_plan() for why larger pages cost less quota.

field — Optional 'Scopus' field tag to wrap the query in (see scopus_plan()).

years — Optional integer vector of publication years to restrict to.

cursor — Logical. When TRUE, retrieve the result set with cursor-based pagination, which has no 5000-record ceiling, so an entire large query can be harvested in one call. The records then arrive in the API's deep-paging order rather than sorted by relevance. As a safeguard against a non-conforming server that never signals the end, cursor paging stops after getOption("scopusflow.max_cursor_pages", 1e5) pages with a warning; set that option to Inf to remove the ceiling.

api_key, inst_token — Optional credentials, resolved by default from options or environment variables (see scopus_has_key()).

verbose — Logical. When TRUE, progress is reported as the retrieval proceeds.

Value

A scopus_records tibble. The reported total and the most recent parsed quota are attached as the total_results and quota attributes.

API access

Requires a valid API key and internet access. The *API access* section of scopus_count() lists the conditions that may be raised.

See Also

scopus_fetch_plan() for cached, resumable, partitioned retrieval.

Examples

```
recs <- scopus_fetch("graphene", field = "TITLE-ABS-KEY", max_results = 50)
recs
```

---

scopus_fetch_plan *Execute a 'Scopus' search plan, with optional caching and resume*

Description

Runs every cell of a scopus_plan() in turn, optionally caching each cell's result so that an interrupted or quota-limited retrieval can resume without re-spending quota on the cells already fetched. Results are accumulated and bound once into a single scopus_records tibble.

Usage

```
scopus_fetch_plan(
  plan,
  max_results = Inf,
  cache_dir = NULL,
  resume = TRUE,
  api_key = NULL,
  inst_token = NULL,
  verbose = FALSE
)
```

Arguments

plan	A scopus_plan object from scopus_plan().
max_results	Maximum records to retrieve per cell (default Inf).
cache_dir	Optional directory for per-cell cache files. The default of NULL performs no caching. Pass an explicit path you control, or scopus_cache_dir() to use a managed, clearable cache under tools::R_user_dir(). Caching happens only when you opt in through this argument.
resume	Logical. When TRUE and cache_dir is set, a cell whose cache file already exists is loaded from disk rather than fetched again.
api_key, inst_token	Optional credentials (see scopus_has_key()).
verbose	Logical. When TRUE, per-cell progress is reported.

Value

A scopus_records tibble combining all cells, with the originating plan attached as the plan attribute.

API access

Any cell not served from cache requires a valid API key and internet access. The *API access* section of scopus_count() gives the details.

See Also

scopus_cache_dir(), scopus_cache_clear()

Examples

```
plan <- scopus_plan("renewable energy", years = 2015:2022, partition = "year")
dir <- file.path(tempdir(), "energy-cache")
# `max_results` caps each yearly cell, so the example stays small and
# quota-light; drop it to harvest every record in the plan.
recs <- scopus_fetch_plan(plan, max_results = 25, cache_dir = dir, resume = TRUE)
```

scopus_field_tags *Recognised 'Scopus' field tags*

Description

Lists the field tags that `scopus_plan()`, `scopus_fetch()` and `scopus_compare_topics()` understand, together with a short note on what each one searches. Passing one of these tags as `field` restricts a query to the corresponding part of a record, so `TITLE-ABS-KEY` looks in the title, abstract and keywords while `AUTH` looks only at author names. Other valid 'Scopus' tags are accepted too. This is a guide to the common ones.

Usage

```
scopus_field_tags()
```

Value

A tibble with a `tag` column and a `searches` column describing the scope of each tag.

See Also

`scopus_plan()`

Examples

```
scopus_field_tags()
```

scopus_has_key *Locate the 'Scopus' API key and institutional token*

Description

`scopus_has_key()` reports whether an API key can be found, without revealing it. The key itself is resolved internally and is never printed by the package.

Usage

```
scopus_has_key()
```

Details

The key is looked up first from the `api_key` argument of whichever function is being called, then from the `scopusflow.api_key` option, and finally from the `SCOPUS_API_KEY` environment variable. An optional institutional token, used for off-campus access to subscriber content, is resolved the same way from the `inst_token` argument, the `scopusflow.inst_token` option, or the `SCOPUS_INST_TOKEN` environment variable.

A key is a secret. The safest home for it is `~/.Renviron`, as in `SCOPUS_API_KEY=xxxx`, rather than a script, and it should stay out of version control.

Value

A length-one logical that is safe to print, TRUE when a non-empty key is available and FALSE otherwise.

See Also

scopus_count(), scopus_fetch()

Examples

```
# Does the current session have a key configured?
scopus_has_key()
```

---

scopus_plan *Build a reproducible 'Scopus' search plan*

Description

A *plan* is a fully specified, inspectable description of one or more 'Scopus' queries to run. Splitting the act of *describing* a search from *executing* it makes workflows reproducible (the plan can be saved, reviewed and version controlled) and lets large retrievals be partitioned, for example one cell per year, so they can be cached and resumed.

Usage

```
scopus_plan(
  query,
  years = NULL,
  field = NULL,
  view = c("STANDARD", "COMPLETE"),
  page_size = NULL,
  partition = c("none", "year")
)

is_scopus_plan(x)
```

Arguments

query	Character scalar. The base search expression, without field tags or year filters (these are added through field and years).
years	Optional integer vector of publication years to restrict to, for example 2015:2020. When partition = "year", one plan cell is created for each distinct year. Otherwise the minimum and maximum define a single date range.
field	Optional character scalar naming a 'Scopus' field tag to wrap the query in, for example "TITLE-ABS-KEY", "TITLE", "AUTH" or "AFFIL". When NULL, the query is used verbatim. See scopus_field_tags() for the common tags.

view	Either "STANDARD" or "COMPLETE". COMPLETE returns more fields but requires a subscriber entitlement and is limited to a smaller page size.
page_size	Integer number of records to request per page, or NULL (the default) to use the largest page the view allows. The 'Scopus' Search API permits up to 200 records per page for the STANDARD view but only 25 for COMPLETE. Because the weekly quota is charged per request, requesting the maximum page size keeps the number of requests, and so the quota, as low as possible for a given result set. Lower it only where you have a reason to.
partition	Either "none" (a single query cell) or "year" (one cell per year in years). Partitioning by year is the recommended way to stay under the API's hard limit of start < 5000.
x	An object to test or print.

Value

A tibble of class scopus_plan, one row per cell, with columns cell, query (field-wrapped), date (year range string or NA), year (integer or NA), view and page_size. Plan-level settings are stored as attributes.

is_scopus_plan() returns a length-one logical.

See Also

scopus_fetch_plan() to execute a plan, scopus_count() to size it.

Examples

```
scopus_plan("quantum computing", years = 2015:2022, field = "TITLE-ABS-KEY")
scopus_plan("immunotherapy", years = 2010:2020, partition = "year")
```

---

scopus_query *Build a field-tagged 'Scopus' query*

Description

Combines several terms into one 'Scopus' query string, optionally wrapping each in a field tag and joining them with a boolean operator. It is a tidier alternative to pasting query fragments together by hand, which is where field-tag and bracket mistakes tend to creep in.

Usage

```
scopus_query(..., .op = c("AND", "OR", "AND NOT"), .field = NULL)
```

Arguments

...	Character terms to combine, for example "language learning" and "effect size".
.op	The boolean operator joining the terms, one of "AND", "OR" or "AND NOT".
.field	Optional field tag applied to every term (see scopus_field_tags()).

Value

A length-one character string suitable for scopus_count(), scopus_fetch() or the query of scopus_plan().

See Also

scopus_field_tags(), scopus_plan()

Examples

```
scopus_query("climate change", "adaptation", .field = "TITLE-ABS-KEY")
scopus_query("graphene", "supercapacitor", .op = "AND")
scopus_query("CRISPR", "Cas9", "Cas12", .op = "OR")
```

---

scopus_quota *Parse 'Scopus' quota and rate-limit headers*

Description

Elsevier returns the caller's weekly quota and short-term rate-limit status in response headers. scopus_quota() extracts them into a tidy list so a workflow can pause, schedule or report on the remaining allowance.

Usage

```
scopus_quota(resp)
```

Arguments

resp	An httr2::response object, typically captured during a request.

Details

The relevant headers are X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset (epoch seconds), X-ELS-Status and Retry-After. When the API raises a quota or rate-limit error, the parsed quota is also attached to the resulting condition, where it is available as cnd$quota.

Value

A list with elements limit, remaining, reset (a POSIXct time at which the rate-limit window resets, or NA), status and retry_after (seconds, or NA). A missing header yields NA.

Examples

```
# Build a fake response to show the shape of the output (no network used).
resp <- httr2::response(
  status_code = 200,
  headers = list(
    `X-RateLimit-Limit` = "20000",
    `X-RateLimit-Remaining` = "19987",
    `X-RateLimit-Reset` = "1700000000"
  )
)
scopus_quota(resp)
```

`scopus_top` *Most frequent values in a record set*

Description

Tallies the most common sources or authors across a scopus_records object. It works on records already in memory, so it makes no network request.

Usage

```
scopus_top(x, by = c("source", "author"), n = 10L)
```

Arguments

`x`	A scopus_records tibble.
`by`	What to tally: `"source"` (the publication titles) or `"author"`. Author strings holding several names separated by `"; "` are split, so each contributor is counted once per record.
`n`	The number of rows to return (the top `n`).

Value

A tibble of class `scopus_top` with columns `value` and `n`, sorted by descending count, with ties broken by `value` in byte order so the result is reproducible across platforms and locales. Exactly `n` rows are returned (fewer if there are fewer distinct values), so values tied at the cut-off rank may be dropped. The `by` choice is stored in the `by` attribute.

See Also

`plot_scopus_top()`, `summary.scopus_records()`

Examples

```
scopus_top(example_records, by = "source")
scopus_top(example_records, by = "author", n = 5)
```

scopus_trend *Annual publication counts for a query*

Description

Counts how many records match a query in each year, giving the size of a literature over time. It is the single-query companion to `scopus_compare_topics()`: where the comparison shows topics as a share of a reference, this shows the absolute count.

Usage

```
scopus_trend(
  query,
  years,
  field = NULL,
  view = c("STANDARD", "COMPLETE"),
  api_key = NULL,
  inst_token = NULL,
  verbose = FALSE
)
```

Arguments

`query`	Character scalar. The base search expression.
`years`	Integer vector of publication years to count over, for example `2010:2020`.
`field`	Optional 'Scopus' field tag to wrap the query in (see `scopus_plan()`).
`view`	Either `"STANDARD"` or `"COMPLETE"`.
`api_key, inst_token`	Optional credentials, resolved by default from options or environment variables (see `scopus_has_key()`).
`verbose`	Logical. When `TRUE`, progress is reported.

Value

A tibble of class `scopus_trend` with columns `query` (the field-wrapped query), `year` (integer) and `n` (the count that year, as a double so very large counts are exact). A year whose response omits a total is recorded as `NA` (with a warning) and contributes nothing to the total shown by `print()`.

API access

This performs one count request per year, so it requires a valid API key and internet access; see the *API access* section of `scopus_count()`.

See Also

`plot_scopus_trend()`, `scopus_compare_topics()`

Examples

```
tr <- scopus_trend("graphene", years = 2010:2020, field = "TITLE-ABS-KEY")
tr
```

`summary.scopus_records`

Summarise a set of 'Scopus' records

Description

Gives a compact overview of a scopus_records object, reporting how many records it holds, the span of publication years they cover, how many distinct sources and Digital Object Identifiers appear among them and how widely they have been cited. It is a convenient way to take stock of a retrieval before any closer analysis.

Usage

```
## S3 method for class 'scopus_records'
summary(object, ...)
```

Arguments

`object`	A scopus_records tibble.
`...`	Ignored, present for compatibility with the `summary()` generic.

Value

A list of class `scopus_records_summary`, with elements `n_records`, `years` (the earliest and latest year present, each `NA` when no year is known), `n_sources`, `n_with_doi`, `total_citations`, `median_citations`, `top_cited` (the title of the most-cited record) and `top_source` (the most frequent source title). Printing it produces a short readable report.

Examples

```
summary(example_records)
```

`write_scopus_records` *Read and write 'Scopus' record sets*

Description

Save a scopus_records tibble to disk and read it back, with a stable round-trip. The file extension selects the format. An `.rds` file preserves the types and class exactly, while a `.csv` file is portable plain text.

Usage

```
write_scopus_records(x, path)

read_scopus_records(path)
```

Arguments

`x`	A scopus_records tibble to write.
`path`	Explicit file path. The functions read from, or write to, exactly this path and leave the working directory alone. Parent directories are assumed to exist already.

Value

`write_scopus_records()` returns `x` invisibly. `read_scopus_records()` returns a scopus_records tibble.

Examples

```
recs <- scopus_records(list(entry = list(
  list(`dc:identifier` = "SCOPUS_ID:1", `prism:doi` = "10.1/a",
       `dc:title` = "A study", `prism:coverDate` = "2020-01-01")
)))
path <- tempfile(fileext = ".csv")
write_scopus_records(recs, path)
read_scopus_records(path)
```

Index

∗ **datasets**
example_records, 6

as.data.frame.scopus_records
(as_tibble.scopus_records), 5
as_bibliometrix, 3
as_bibliometrix(), *4*
as_bibtex, 4
as_ris *(*as_bibtex*)*, 4
as_tibble.scopus_records, 5
autoplot.scopus_comparison
(plot_scopus_comparison), 7
autoplot.scopus_records
(as_tibble.scopus_records), 5
autoplot.scopus_top *(*plot_scopus_top*)*, 8
autoplot.scopus_trend
(plot_scopus_trend), 9

c.scopus_records *(*scopus_combine*)*, 13

example_records, 6

ggplot2::ggplot, *5*, *8–10*

httr2::response, *24*

is_scopus_plan *(*scopus_plan*)*, 22
is_scopus_records
(as_tibble.scopus_records), 5

plot_scopus_comparison, 7
plot_scopus_comparison(), *15*
plot_scopus_top, 8
plot_scopus_top(), *25*
plot_scopus_trend, 9
plot_scopus_trend(), *26*

read_scopus_records
(write_scopus_records), 28
run_app, 10

scopus_abstract, 11
scopus_cache_clear, 12
scopus_cache_clear(), *12*, *20*
scopus_cache_dir, 12
scopus_cache_dir(), *12*, *20*
scopus_combine, 13
scopus_compare_topics, 14
scopus_compare_topics(), *7, 8*, *21*, *26*
scopus_count, 15
scopus_count(), *11*, *15*, *19, 20*, *22–24*, *26*
scopus_diff_dois, 16
scopus_diff_dois(), *17*
scopus_extract_dois, 17
scopus_extract_dois(), *4*, *12*, *16*
scopus_fetch, 18
scopus_fetch(), *5, 6*, *12*, *21, 22*, *24*
scopus_fetch_plan, 19
scopus_fetch_plan(), *5*, *10*, *12, 13*, *19*, *23*
scopus_field_tags, 21
scopus_field_tags(), *22–24*
scopus_has_key, 21
scopus_has_key(), *11*, *14, 15*, *19, 20*, *26*
scopus_plan, 22
scopus_plan(), *10*, *14, 15*, *18–21*, *24*, *26*
scopus_query, 23
scopus_quota, 24
scopus_quota(), *16*
scopus_records, *3–6*, *13*, *16–20*, *25*, *27, 28*
scopus_records
(as_tibble.scopus_records), 5
scopus_top, 25
scopus_top(), *8, 9*
scopus_trend, 26
scopus_trend(), *9, 10*
shiny::runApp(), *10*
summary(), *27*
summary.scopus_records, 27
summary.scopus_records(), *25*

tibble, *5*, *21*
tools::R_user_dir(), *12*, *20*

tryCatch(), *16*

write_scopus_records, 28
write_scopus_records(), *4*