Package: openalexPro (via r-universe)

June 3, 2026

Type Package

Title Providing a more advanced access to OpenAlex for the power user

Version 0.10.4

Author Rainer M Krug

Maintainer Rainer M Krug <Rainer@krugs.de>

Description More about what it does (maybe more than one line).

URL https://github.com/openalexPro/openalexPro,
https://openalexpro.github.io/openalexPro/,
https://doi.org/10.5281/zenodo.17453180

BugReports https://github.com/openalexPro/openalexPro/issues

License GPL (>= 2)

Depends R (>= 4.2)

Imports arrow, cli, DBI, dplyr, duckdb, future, future.apply, httr2, jqr, progressr, rlang

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

Suggests jsonlite, keyring, knitr, openalexR (>= 1.4.0), rmarkdown, quarto, testthat (>= 3.0.0), vcr (> 1.7.0), vdiffr, withr

Encoding UTF-8

VignetteBuilder quarto

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake libjq-dev libssl-dev xz-utils

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

Date/Publication 2026-06-03 13:58:37 UTC

RemoteUrl https://github.com/openalexPro/openalexPro

RemoteRef main

RemoteSha ad136db0e41e52cd58d5eed34c91ee742f9fa2f5

Contents

`build_corpus_index` *Build a Parquet ID-lookup index*

Description

Moved to the **openalexSnapshot package.**

This function has been removed from **openalexPro**. Please install the **openalexSnapshot** package and call ‘openalexSnapshot::build_corpus_index()‘ instead.

Usage

```
build_corpus_index(...)
```

Arguments

...	Ignored.

See Also

https://github.com/rkrug/openalexSnapshot

---

compatibility_report *Render and open the compatibility report*

Description

Renders the Quarto report at ‘system.file("compatibility.qmd", package = "openalexPro") and opens the resulting HTML in your default browser.

Usage

```
compatibility_report(
  output_dir = "Compatibility Report",
  open = TRUE,
  quiet = FALSE
)
```

Arguments

output_dir	Directory to write the rendered HTML and the data into. Defaults to the flder ‘./Compatibility Report‘.
open	Logical; if ‘TRUE‘ (default) opens the rendered HTML in the system browser.
quiet	Logical; suppress rendering output if ‘TRUE‘. Default: ‘FALSE‘.

Details

This report is designed to help you validate client–API compatibility in real time. During rendering, the report performs live requests against the OpenAlex API and compares the responses to the package’s expected behavior. No cached data are used: every section issues fresh API calls so that the output reflects the current state of the upstream service. The report summarizes differences in fields, types, pagination and response shapes to surface potential regressions from upstream changes or local client updates.

Note: Because it depends on live API calls, rendering may take longer and requires network access. Be mindful of API rate limits when running the report repeatedly.

Value

Invisibly returns the path to the rendered HTML file.

`extract_doi` *Extract DOIs or Components from Character Vectors*

Description

Extracts DOIs or specific DOI components (resolver, prefix, or suffix) from a character vector. Assumes that each element of ‘x‘ contains at most one DOI (with or without resolver).

Usage

```
extract_doi(
  x,
  non_doi_value = "",
  normalize = TRUE,
  what = c("doi", "resolver", "prefix", "suffix")
)
```

Arguments

`x`	A character vector potentially containing DOIs (e.g., raw DOIs, DOI URLs, or strings with embedded DOIs).
`non_doi_value`	Value to use for elements where no DOI or component is found. If ‘NULL‘, only matched elements are returned.
`normalize`	Logical. If ‘TRUE‘ (default), convert extracted DOIs and suffixes to lowercase and trim surrounding whitespace. Has no effect for ‘what = "prefix"‘ or ‘what = "resolver"‘.
`what`	What to extract from each element. One of: **"doi"** The full DOI name (prefix + "/" + suffix). Example: ‘"10.5281/zenodo.1234567"‘ (default) **"resolver"** The resolver URL (e.g., ‘"https://doi.org/"‘, ‘"http://dx.doi.org/"‘) if present **"prefix"** The DOI prefix only (e.g., ‘"10.5281"‘) **"suffix"** The DOI suffix only (e.g., ‘"zenodo.1234567"‘)

Value

A character vector: - If ‘non_doi_value‘ is not ‘NULL‘, a vector of the same length as ‘x‘, with unmatched entries replaced. - If ‘non_doi_value‘ is ‘NULL‘, a vector of only matched entries.

Examples

```
x <- c(
  "https://doi.org/10.5281/zenodo.1234567",
  " 10.1000/XYZ456  ",
  "no doi here",
  NA
```

```
)

extract_doi(x)  # Full DOIs (default)
extract_doi(x, what = "resolver")
extract_doi(x, what = "prefix")
extract_doi(x, what = "suffix")
extract_doi(x, non_doi_value = NA_character_)
extract_doi(x, non_doi_value = NULL)
```

`id_block` *Compute ID block from OpenAlex IDs*

Description

This function computes the ID block partition key from OpenAlex IDs. The ID block is calculated as the trailing numeric portion of the ID divided by 10,000.

Usage

```
id_block(ids)
```

Arguments

`ids`	Character vector of OpenAlex IDs in any format: • Short form: `"W2741809807"` • Long form: `"https://openalex.org/W2741809807"` • Path-based: `"https://openalex.org/domains/2"`

Details

OpenAlex IDs come in several formats:

- Standard: `https://openalex.org/{type}{number}` (e.g., `W1234567890`)
- Path-based: `https://openalex.org/{entity_type}/{number}` (e.g., `domains/2`, `subfields/2208`)

The ID block is computed as `floor(number / 10000)`, where `number` is the trailing numeric portion of the ID. This groups approximately 10,000 IDs into each block, useful for partitioning large datasets.

Value

Integer vector of ID blocks.

Examples

```
# Short form IDs
id_block(c("W2741809807", "W2741809808", "W1234567890"))
# Returns: c(274180, 274180, 123456)

# Long form IDs
id_block("https://openalex.org/W2741809807")
# Returns: 274180

# Path-based IDs
id_block("https://openalex.org/domains/2")
# Returns: 0

# Works with any entity type
id_block(c("A123456789", "I987654321"))
# Returns: c(12345, 98765)
```

`infer_json_schema` *Infer unified JSON schema using DuckDB*

Description

Infers the schema of each JSON/NDJSON file individually via DuckDB's 'read_json_auto()' and merges the results using type-widening rules. Processing files one at a time avoids the out-of-memory errors that occur when opening all files in a single DuckDB query.

Usage

```
infer_json_schema(
  con,
  files,
  sample_size = 20,
  extra_options = "",
  verbose = TRUE,
  schema_cache_dir = NULL
)
```

Arguments

`con`	An active DuckDB connection ('DBI::dbConnect(duckdb::duckdb())') with the JSON extension loaded ('LOAD json').
`files`	Character vector of paths to JSON or NDJSON ('.gz') files.
`sample_size`	Number of files to sample for schema inference. Higher values give more accurate schemas but take longer. Use '0' or 'NULL' to use all files. Default is '20'.

`extra_options` Additional options appended to the ‘read_json_auto‘ SQL call, e.g. ‘", maximum_object_size=1000000000"‘ for large JSON objects. Default is ‘"""‘.

`verbose` If ‘TRUE‘, print progress messages and a progress bar. Default is ‘TRUE‘.

`schema_cache_dir` Path to a directory for caching per-file and unified schemas. The directory is created if it does not exist. ‘NULL‘ (default) disables caching.

Details

The returned columns clause can be passed directly to ‘read_json(..., columns = <result>)‘ or ‘read_json_auto(..., columns = <result>)‘ in subsequent DuckDB queries to enforce a consistent schema across all files.

Value

A DuckDB columns clause string (e.g. `"{'col1': 'VARCHAR', 'col2': 'BIGINT', ...}"`) suitable for use as the ‘columns‘ argument to ‘read_json()‘. Returns ‘NULL‘ if schema inference fails for all files.

Caching

When ‘schema_cache_dir‘ is provided, two levels of caching apply: - **Unified schema** (‘unified_schema.csv‘): if present, loaded and returned immediately — no DuckDB queries needed. Delete this file to force re-inference. - **Per-file schemas** (‘<update_date>_<part_name>.csv‘): each file's schema is saved as it is inferred. On restart, already-cached files are skipped, enabling mid-run resume for large file sets.

Type-widening rules

When a column has different types across files, the unified type is chosen by these rules (in order): 1. All identical → keep as-is. 2. Any ‘STRUCT‘/‘LIST‘/‘MAP‘ vs simpler type → complex type wins. 3. Multiple ‘STRUCT‘ types → pick the one with the most fields. 4. Numeric conflicts → widest type wins (‘TINYINT < SMALLINT < INTEGER < BIGINT < HUGEINT < FLOAT < DOUBLE‘). 5. Fallback → ‘VARCHAR‘.

See Also

[snapshot_to_parquet()] which uses this function internally.

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb())
DBI::dbExecute(con, "LOAD json")
files <- list.files("path/to/snapshot/works", pattern = "\\.gz$",
                    recursive = TRUE, full.names = TRUE)
schema <- infer_json_schema(con, files, sample_size = 50,
                            schema_cache_dir = "path/to/cache")
DBI::dbDisconnect(con, shutdown = TRUE)
# schema is now a string like: {'id': 'VARCHAR', 'title': 'VARCHAR', ...}
```

```
## End(Not run)
```

`jq_execute` *Execute a jq transformation from an OpenAlex-style JSON to JSONL*

Description

This function runs a jq filter to extract records from the "results" array (or from the root if type = "single"), reconstruct the abstract text, generate a citation string, and optionally add a page field. It writes the result as newline-delimited JSON (.jsonl), suitable for Arrow or DuckDB. For details on the jq filter logic, see the `vignette`("jq", package = "openalexPro").

Usage

```
jq_execute(
  input_json,
  output_jsonl,
  add_columns = list(),
  jq_filter = NULL,
  page = NULL,
  type = c("results", "single", "group_by")
)
```

Arguments

`input_json`	Path to the input JSON file
`output_jsonl`	Path to the output .jsonl file
`add_columns`	List of additional fields to be added to the output. They nave to be provided as a named list, e./g. ‘list(column_1 = "value_1", column_2 = 2)‘. Only Scalar values are supported.
`jq_filter`	Optional custom jq filter string. If NULL, the default filter is used.
`page`	Optional integer to be added as a "page" field in each output record
`type`	Either "results" (default, expects a .results[] array) or "single" (treat input as array of records directly)

Value

Invisibly returns the output path

lookup_by_id *Look up records by OpenAlex ID*

Description

Moved to the **openalexSnapshot package.**

This function has been removed from **openalexPro**. Please install the **openalexSnapshot** package and call ‘openalexSnapshot::lookup_by_id()‘ instead.

Usage

```
lookup_by_id(...)
```

Arguments

`...`	Ignored.

See Also

https://github.com/rkrug/openalexSnapshot

oa_cache_schema *Populate the local baseline-schema cache from a snapshot metadata directory*

Description

Copies `unified_schema.csv` files from an OpenAlex snapshot metadata directory (e.g. `/Volumes/openalex/openalex-sna` into the user-level cache used by `pro_request_parquet(schema = "auto")`.

Usage

```
oa_cache_schema(source, entities = "all", overwrite = FALSE, verbose = TRUE)
```

Arguments

`source`	Path to the snapshot metadata directory, e.g. `"/Volumes/openalex/openalex-snapshot_metadata"`.
`entities`	Character vector of entity names to cache, or `"all"` (default) to cache every entity directory found under `source`.
`overwrite`	Logical. Overwrite an existing cached file? Default `FALSE`.
`verbose`	Logical. Print progress messages? Default `TRUE`.

Details

Once cached, the schemas are used even when the source volume is not mounted. Update the cache periodically to pick up new fields added by OpenAlex (run with `overwrite = TRUE`).

Value

The path to the schemata cache directory (invisibly).

See Also

pro_request_parquet for the schema parameter.

`oa_normalize_duckdb_type` *Canonicalise a DuckDB type string.*

Description

Uppercases SQL type keywords (`BIGINT`, `VARCHAR`, `STRUCT`, ...) while preserving the case of struct field identifiers.

Usage

```
oa_normalize_duckdb_type(t)
```

Arguments

`t`	A character scalar: a raw DuckDB type string, e.g. `"struct(author struct(display_name varchar))"`.

Value

A character scalar with normalised type keywords.

`oa_works_abstract_sql` *Return the DuckDB SQL expression that reconstructs a plain-text abstract from the* `abstract_inverted_index` *column in OpenAlex works data.*

Description

The expression walks the map, collects (position, word) pairs, sorts by position ascending, and joins words with single spaces. Returns NULL when `abstract_inverted_index` is NULL.

Usage

```
oa_works_abstract_sql()
```

Details

abstract_inverted_index is normalised to MAP(VARCHAR, BIGINT[]) via a double JSON cast (::JSON::MAP(VARCHAR, BIGINT[])) before map_entries() is called. This makes the expression safe regardless of whether DuckDB inferred the column as MAP, STRUCT, or VARCHAR (raw JSON text): all three round-trip through the JSON representation identically.

Value

A character scalar containing the SQL expression.

---

oa_works_citation_sql *Return the DuckDB SQL expression that builds a short citation string from the* authorships *and* publication_year *columns in OpenAlex works data.*

Description

Format: "Author (year)" / "A & B (year)" / "A et al. (year)". Null year renders as "(n.d.)". Null or empty authorships yields NULL.

Usage

```
oa_works_citation_sql()
```

Value

A character scalar containing the SQL expression.

---

opt_api_key *Get API key for OpenAlex API*

Description

Get API key for OpenAlex API

Usage

```
opt_api_key(api_key)
```

Arguments

api_key	character vector or NULL. If specified, value to assign to the api key option. Default is 'NULL'.

Value

The API key, if 'api_key' is not specified the current one, otherwise the old one.

opt_filter_names *Get available filter names from OpenAlex API*

Description

Get available filter names from OpenAlex API

Usage

```
opt_filter_names(update = FALSE)
```

Arguments

update	logical. If 'TRUE' update the existing value. Default is 'FALSE'.

Value

A character vector of available filter names

opt_select_fields *Get available select fields from OpenAlex API*

Description

Get available select fields from OpenAlex API

Usage

```
opt_select_fields(update = FALSE)
```

Arguments

update	logical. If 'TRUE' update the existing value. Default is 'FALSE'.

Value

A character vector of available select fields

`prepare_snapshot` *Prepare a directory for OpenAlex snapshot management*

Description

Copies the Makefile for snapshot management to the specified directory and provides instructions for creating and managing OpenAlex snapshots.

Usage

```
prepare_snapshot(path = ".", overwrite = FALSE)
```

Arguments

`path`	Character. The directory where the Makefile and documentation should be copied. Defaults to the current working directory.
`overwrite`	Logical. Whether to overwrite existing files. Defaults to FALSE.

Details

This function sets up a directory for managing OpenAlex snapshots by:

1. Copying a Makefile with targets for downloading and converting snapshots
2. Copying documentation about the snapshot process

The Makefile provides the following targets:

help Show available make targets

snapshot Download/sync OpenAlex snapshot from S3

parquet Convert snapshot to parquet format

parquet_index Build ID indexes for fast lookups

clean Remove generated directories

Value

Invisibly returns the path to the created Makefile.

Examples

```
## Not run:
# Prepare current directory
prepare_snapshot()

# Prepare a specific directory
prepare_snapshot("/path/to/openalex-data")

## End(Not run)
```

pro_api_key *Retrieve the OpenAlex Pro API key*

Description

Retrieves the OpenAlex Pro API key from one of several locations, checked in the following order:

Usage

```
pro_api_key()
```

Details

1. The R option `openalexPro$api_key`
2. The environment variable `openalexPro.api_key`
3. The system keyring via the **keyring** package (only if the package **keyring** is installed)

If no API key is found, `NULL` or an empty string may be returned, depending on the environment variable state.

Value

A character string containing the API key, or `NULL` if no key could be found.

See Also

`options`, `Sys.getenv`

Examples

```
## Not run:
pro_api_key()

options(openalexPro = list(api_key = "my-key"))
pro_api_key()

## End(Not run)
```

pro_download_content *Download full-text PDFs or TEI XML for OpenAlex works*

Description

Downloads full-text content from the OpenAlex content endpoint (content.openalex.org) for a vector of work IDs. One file is written per ID. Downloads can be parallelised via the workers argument.

Usage

```
pro_download_content(
  ids,
  format = c("pdf", "grobid-xml"),
  output = ".",
  workers = 1L,
  api_key = pro_api_key(),
  endpoint = "https://content.openalex.org"
)
```

Arguments

ids	Character vector of OpenAlex work IDs (e.g. "W2741809807") or full OpenAlex URLs ("https://openalex.org/W2741809807"). Full URLs are normalised automatically.
format	File format to download. One of "pdf" (default) or "grobid-xml" (TEI XML).
output	Directory to save downloaded files into. Defaults to the current working directory. Created if it does not exist.
workers	Number of parallel download workers. Defaults to 1 (sequential). Set higher for faster batch downloads, subject to the content endpoint's rate limits.
api_key	OpenAlex API key (character string) or 'NULL'. Defaults to the openalexPro.apikey environment variable. If 'NULL' or '""', requests are sent without an API key.
endpoint	Base URL of the content endpoint. Defaults to "https://content.openalex.org".

Value

A data frame with one row per ID and columns:

id The (normalised) work ID.

file Full path to the saved file, or NA if not downloaded.

status One of "ok", "not_found" (HTTP 404), or "error".

message Error message, or NA on success.

Costs

Content downloads cost **$0.01 per file** — 10x the cost of a metadata search query. Use `has_content.pdf:true` or `has_content.grobid-xml:true` as filter arguments to `pro_query()` to discover which works have downloadable content before downloading.

Formats

`"pdf"` Full-text PDF (~60 million files available).

`"grobid-xml"` Machine-readable TEI XML parsed by Grobid (~43 million files). Suitable for structured text extraction.

Licensing

PDFs and XMLs retain their original copyright. OpenAlex does not grant additional rights. Check the `best_oa_location.license` field of each work for the applicable licence.

Examples

```
## Not run:
# Download a single PDF
result <- pro_download_content(
  ids    = "W2741809807",
  format = "pdf",
  output = tempdir()
)

# Find works with PDFs available, then download them
urls <- pro_query(
  entity          = "works",
  has_content.pdf = TRUE,
  from_publication_date = "2023-01-01",
  options = list(per_page = 10)
)
works <- pro_request(urls, output = tempdir())
# ... extract IDs from works data, then:
result <- pro_download_content(ids = work_ids, format = "pdf", workers = 4)

# XPAC works: discover via pro_query() with include_xpac = TRUE, then download
# (pro_download_content() works with any valid OpenAlex ID, including XPAC IDs)
urls_xpac <- pro_query(
  entity          = "works",
  has_content.pdf = TRUE,
  from_publication_date = "2023-01-01",
  options = list(include_xpac = TRUE, per_page = 10)
)
works_xpac <- pro_request(urls_xpac, output = tempdir())
# ... extract IDs from works_xpac data, then:
result_xpac <- pro_download_content(ids = xpac_ids, format = "pdf", workers = 4)

## End(Not run)
```

pro_fetch *Fetch and convert OpenAlex data to Parquet*

Description

Convenience wrapper that downloads records from OpenAlex via `pro_request()` and converts them directly to an Apache Parquet dataset via `pro_request_parquet()`. No intermediate JSONL files are written.

Usage

```
pro_fetch(
  query_url,
  pages = 10000,
  project_folder = NULL,
  overwrite = FALSE,
  api_key = pro_api_key(),
  delete_input = TRUE,
  workers = 1,
  verbose = FALSE,
  progress = TRUE,
  enrich = TRUE,
  count_only,
  error_log = NULL
)
```

Arguments

`query_url`	The URL of the API query or a list of URLs returned from `pro_query()`.
`pages`	The number of pages to be downloaded. The default is set to 10000, which would be 2,000,000 works. It is recommended to not increase it beyond 100000 due to server load and to use the snapshot instead. If `NULL`, all pages will be downloaded. Default: 100000.
`project_folder`	Directory where intermediate (`json`) and final (`parquet`) results are stored. If it does not exist, it is created. If `NULL`, a temporary directory is created.
`overwrite`	Logical. If `TRUE`, the `json` and `parquet` subdirectories are deleted from `project_folder` before the pipeline starts. If `FALSE` (the default) and any of those subdirectories already exist, the function stops with an error.
`api_key`	Character string API key or `NULL`. Defaults to `pro_api_key()`. If `NULL` or `""`, requests are sent without an API key (subject to OpenAlex's unauthenticated limits).
`delete_input`	Logical. If `TRUE` (the default), the `json` subfolder is deleted after successful conversion to Parquet.
`workers`	Number of parallel workers to use if `query_url` is a list. Defaults to 1.
`verbose`	Logical indicating whether to show verbose messages.

progress	Logical indicating whether to show a progress bar. Default TRUE.
enrich	Logical. When TRUE (the default) and the inferred schema contains abstract_inverted_index / authorships / publication_year, add abstract and citation computed columns.
count_only	Do not use it here. The function will abort if set to TRUE and give a warning if FALSE.
error_log	location of error log of API calls. (default: NULL (none)).

Details

The function

- downloads records from OpenAlex via pro_request() into a "json" subfolder of project_folder, and
- converts the JSON files to an Apache Parquet dataset via pro_request_parquet() into a "parquet" subfolder.

This function assumes count_only == FALSE

Value

Invisibly, the normalised path of the parquet subfolder inside project_folder.

See Also

pro_request() for the download step, pro_request_parquet() for the conversion step.

---

pro_query *Build an OpenAlex request (httr2)*

Description

Construct an httr2 request for the OpenAlex API. All filters must be supplied as named ... arguments (e.g., from_publication_date = "2020-01-01").

Usage

```
pro_query(
  entity = c("works", "authors", "venues", "institutions", "concepts", "publishers",
    "funders"),
  id = NULL,
  doi = NULL,
  search = NULL,
  search.exact = NULL,
  search.semantic = NULL,
  group_by = NULL,
  select = NULL,
```

```
  options = NULL,
  endpoint = "https://api.openalex.org",
  chunk_limit = 50L,
  ...
)
```

Arguments

entity	Character; one of "works", "authors", "venues", "institutions", "concepts", "publishers", "funders".
id	Optional ID or vector of IDs (e.g., "W1775749144"). If a single ID is provided, fetches one entity directly. If multiple IDs are provided, they are automatically moved into the ids.openalex filter.
doi	Optional DOI or vector of DOIs (e.g., "10.1038/s41586-021-03819-2"). Values are moved into the doi filter and automatically chunked into separate requests when the number of DOIs exceeds chunk_limit.
search	Optional full-text search string. Applies stemming and stop-word removal. Supports boolean operators (AND, OR, NOT in uppercase), quoted phrases ("exact phrase"), proximity ("word1 word2"~N), wildcards (*, ?), and fuzzy matching (term~N). Replaces the deprecated filter = field.search:keyword syntax.
search.exact	Optional full-text search without stemming or stop-word removal. Supports the same boolean/phrase/wildcard syntax as search. Use when you need to match exact word forms (e.g. "surgery" should not match "surgical").
search.semantic	Optional semantic (AI-powered) search string. Uses embeddings to match conceptual meaning rather than exact keywords. Limited to 1 request per second and returns at most 50 results per query.
group_by	Optional field to group by (facets), e.g. "type".
select	Optional character vector of fields to return.
options	Optional named list of additional query parameters (e.g., list(per_page = 200, sort = "cited_by_count:desc", cursor = "*", sample = 100)).
endpoint	Base API URL. Defaults to "https://api.openalex.org".
chunk_limit	Number of DOIS or ids per chunk if chunked. Default: 50
...	Filters as named arguments. Values may be scalars or vectors (vectors are collapsed with "|" to express OR).

Details

Filter names are validated via .validate_filter() using opt_filter_names(). select fields are validated via .validate_select() using `opt_select_fields()`.

If multiple more then 50 ‘doi‘ or openalex ‘id‘s are provided, the request is automatically split into chunks of 50 and a named list of URLs is returned.

Value

An individual URL or a list of URLs.

Search syntax

All three search parameters (`search`, `search.exact`, `search.semantic`) accept a query string. For `search` and `search.exact`, the following syntax is supported:

- Boolean: `biodiversity AND finance`, `climate OR weather`, `ocean NOT pollution` (operators must be uppercase).
- Exact phrase: `"biodiversity finance"` (double quotes).
- Proximity: `"biodiversity finance"~5` (words within 5 positions).
- Wildcard: `bio*` (zero or more characters), `organi?ation`.
- Fuzzy: `biodiversty~1` (allows 1 character edit).

`search.semantic` does not use keyword syntax; pass a natural-language phrase or even a full abstract. It returns at most 50 results per call.

Deprecated search filters

Filter arguments with a `.search` suffix (e.g. `title_and_abstract.search = "biodiversity"`) are deprecated by the OpenAlex API. They still work but emit a warning. Use the `search`, `search.exact`, or `search.semantic` parameters instead. See https://developers.openalex.org/guides/searching for details.

Examples

```
## Not run:

req <- oa_build_req(
  entity = "works",
  search = "biodiversity",
  from_publication_date = "2020-01-01",
  language = c("en","de"),
  select = c("id","title","publication_year"),
  options = list(per_page = 5)
)
# resp <- api_call(req)
# httr2::resp_body_json(resp)

## End(Not run)
```

---

pro_rate_limit_status *Check OpenAlex rate limit status*

Description

Queries the OpenAlex rate-limit endpoint and returns current API usage and remaining budget as a parsed list.

Usage

```
pro_rate_limit_status(api_key = pro_api_key(), verbose = TRUE)
```

Arguments

api_key	API key (character string) or 'NULL'. Defaults to pro_api_key(). If 'NULL' or '""', this function returns FALSE with an informational message.
verbose	Logical. If TRUE (default), prints rate limit info via message().

Value

Invisibly, the parsed JSON list with all rate limit fields; FALSE if the API key is missing or invalid; or NULL if the request failed due to a network error.

---

pro_request *Fetch works from OpenAlex*

Description

All returned values from OpenAlex will be saved as json files in the output directory and the return value is the directory of the json files.

Usage

```
pro_request(
  query_url,
  pages = 1e+05,
  output = NULL,
  overwrite = FALSE,
  api_key = pro_api_key(),
  workers = 1,
  verbose = FALSE,
  progress = TRUE,
  count_only = FALSE,
  error_log = NULL
)
```

Arguments

query_url	The URL of the API query or a list of URLs returned from pro_query().
pages	The number of pages to be downloaded. The default is set to 10000, which would be 2,000,000 works. It is recommended to not increase it beyond 100000 due to server load and to use the snapshot instead. If NULL, all pages will be downloaded. Default: 100000.
output	directory where the JSON files are saved. Default is a temporary directory. Needs to be specified.

overwrite	Logical. If TRUE, output will be deleted before downloading. For a list query_url, the entire top-level output directory is removed upfront. If FALSE (the default) and output already exists, the function stops with an error.
api_key	Character string API key or NULL. Defaults to pro_api_key(). If NULL or "", requests are sent without an API key (subject to OpenAlex's unauthenticated limits).
workers	Number of parallel workers to use if query_url is a list. Defaults to 1.
verbose	Logical indicating whether to show verbose messages.
progress	Logical indicating whether to show a progress bar. Default TRUE.
count_only	return count only as a data.frame.
error_log	location of error log of API calls. (default: NULL (none)).

Details

If query_url is a list, the function is called for each element of the list in parallel using a maximum of workers parallel R sessions. The results from the individual URLs in the list are returned in a folder named after the names of the list elements in the output folder.

When starting the download, a file 00_in.progress which is deleted upon completion.

Value

If count_only is FALSE (the default) the complete path to the expanded and normalized output. If count_only is TRUE, a data.frame with metadata about the query (count, db_response_time_ms, page, per_page, error). When query_url is a list, an additional query column identifies each query.

---

pro_request_jsonl_parquet

Convert JSON files to Apache Parquet files

Description

The function takes a directory of JSONL files as written from a call to pro_request_jsonl_R(...) and converts each file individually to a Parquet file. The subfolder structure from the input is preserved in the output, so files in Chunk_1/ will be written to Chunk_1/ in the output directory.

Usage

```
pro_request_jsonl_parquet(
  input_jsonl = NULL,
  output = NULL,
  overwrite = FALSE,
  verbose = TRUE,
  delete_input = FALSE,
  sample_size = 1000,
  workers = NULL
)
```

Arguments

input_jsonl	The directory of JSON files returned from pro_request(..., json_dir = "FOLDER").
output	output directory for the parquet dataset; default: temporary directory.
overwrite	Logical indicating whether to overwrite output.
verbose	Logical indicating whether to show verbose information. Defaults to TRUE
delete_input	Determines if the input_jsonl should be deleted afterwards. Defaults to FALSE.
sample_size	Number of records to sample from each file when inferring the unified schema. Higher values give more accurate schema inference but use more memory. Default is 1000. Set to -1 to read all records (may be slow for large files).
workers	Number of parallel workers for file conversion via future.apply::future_lapply(). Default is NULL (sequential processing).

Details

The page column (added by pro_request_jsonl_R()) is preserved as a regular column in the Parquet data.

When starting the conversion, a file 00_in.progress is created which is deleted upon completion.

The function uses DuckDB to read the JSON files and to create the Apache Parquet files. Each JSON file is converted individually using its own DuckDB connection, which enables parallel processing via future.apply::future_lapply().

To ensure consistent schemas across all Parquet files, the function first infers a unified schema by sampling records from all JSONL files. This prevents type mismatches (e.g., a column being struct in one file but string in another) that would cause errors when reading the combined Parquet dataset.

Value

The function returns the output path invisibly.

---

pro_request_jsonl_R *Convert JSON files to jsonl files*

Description

The function takes a directory of JSON files as written from a call to pro_request(...) and is preparing the json files to be processed further using DuckDB by converting them to jsonl files. The subfolders in input_json are preserved in output, i.e. results of a list of initial queries passed to pro_request() are maintained.

Usage

```
pro_request_jsonl_R(
  input_json = NULL,
  output = NULL,
  add_columns = list(),
  overwrite = FALSE,
  verbose = TRUE,
  progress = TRUE,
  delete_input = FALSE,
  workers = 1
)
```

Arguments

input_json	The directory of JSON files returned from pro_request(..., json_dir = "FOLDER").
output	output directory for the jsonl files as created by calls to 'jq_execute().
add_columns	List of additional fields to be added to the output. They nave to be provided as a named list, e./g. list(column_1 = "value_1", column_2 = 2). Only Scalar values are supported.
overwrite	Logical indicating whether to overwrite output.
verbose	Logical indicating whether to show a verbose information. Defaults to TRUE
progress	Logical indicating whether to show a progress bar. Default TRUE.
delete_input	Determines if the input_json should be deleted afterwards. Defaults to FALSE.
workers	Number of parallel workers to use. Defaults to 1.

Details

See jq_execute or the vignette("jq", package = "openalexPro") for more information on the conversion of the JSON files. The folder/filename is converted to a value named page As an example:

1. the subfolder in the output folder is called Chunk_1
2. the page othe json file represents is 2
3. The resulting cvalus for page will be Chunk_1_2

When starting the conversion, a file 00_in.progress which is deleted upon completion.

The function uses DuckDB to read the JSON files and to create the Apache Parquet files. The function creates a DuckDB connection in memory and readsds the JSON files into DuckDB when needed. Then it creates a SQL query to convert the JSON files to Apache Parquet files and to copy the result to the specified directory.

Value

The function does returns the output invisibly.

Examples

```
## Not run:
  source_to_parquet(
  input_json = "json",
  source_type = "snapshot",
  output = "parquet"
)
## End(Not run)
```

`pro_request_parquet` *Convert JSON files from pro_request() directly to Apache Parquet*

Description

Single-step replacement for the two-step `pro_request_jsonl_R()` + `pro_request_jsonl_parquet()` pipeline. Reads the JSON files written by `pro_request()` and converts each one to a Parquet file using DuckDB, with no intermediate JSONL on disk.

Usage

```
pro_request_parquet(
  input_json = NULL,
  output = NULL,
  add_columns = list(),
  overwrite = FALSE,
  verbose = TRUE,
  progress = TRUE,
  delete_input = FALSE,
  sample_size = 1000,
  workers = NULL,
  enrich = TRUE,
  schema = "auto"
)
```

Arguments

`input_json`	Directory of JSON files returned by `pro_request()`.
`output`	Output directory for the Parquet dataset.
`add_columns`	Named list of scalar constant columns to embed in every output record (e.g. `list(query = "my_filter")`). Values are embedded as SQL string literals; only character scalars are supported.
`overwrite`	Logical. Overwrite `output` if it already exists. Default `FALSE`.
`verbose`	Logical. Show progress messages. Default `TRUE`.
`progress`	Logical. Show a progress bar. Default `TRUE`.

`delete_input` Logical. Delete `input_json` after a successful conversion. Default `FALSE`.

`sample_size` Integer. Number of records per file passed to DuckDB's `sample_size` option during schema inference. Use `-1` to read all records (accurate but slow for large files). Default `1000`.

`workers` Integer. Number of parallel workers. `NULL` or `1` runs sequentially. Default `NULL`.

`enrich` Logical. When `TRUE` (the default) and the inferred schema contains `abstract_inverted_index` / `authorships` / `publication_year`, add `abstract` and `citation` computed columns.

`schema` Controls use of a pre-built baseline schema for type resolution. Possible values:

`"auto"` **(default)** Auto-detect the OpenAlex entity type from the inferred columns, then load the matching schema from the user cache (populated by `oa_cache_schema()`) or the schemas bundled with the package. For each column where DuckDB runtime inference produced the ambiguous `JSON` fallback type, the baseline type is used instead. Falls back silently to runtime-only inference when the entity cannot be detected or no schema is found.

`"none"` **or** `NULL` Skip the baseline entirely; behaviour is identical to package versions before this feature was added.

A file path Path to a CSV with columns `col_name` / `col_type`. Used directly as the baseline.

A directory path Auto-detect entity, then look for `<entity>.csv` inside that directory. Useful when pointing directly at a snapshot-metadata schemata directory.

Details

For works entities the function detects the presence of `abstract_inverted_index`, `authorships`, and `publication_year` in the inferred schema and, when `enrich = TRUE` (the default), adds two computed columns:

- `abstract` — plain text reconstructed from `abstract_inverted_index`.
- `citation` — `"Author (year)"` / `"A & B (year)"` / `"A et al. (year)"`.

These expressions are identical to those used by the `openalex-snapshot` CLI binary, so the Parquet output matches the snapshot pipeline column for column.

Value

Output directory path (invisibly).

File format

`pro_request()` writes one JSON file per API page. For paginated queries each file has the structure `{"results": [...], "meta": {...}}`. For group-by queries the array field is `"group_by"`. For single-record lookups the file is a bare JSON object. All three formats are handled automatically.

Output layout

The subdirectory structure of `input_json` is preserved, with hive-partition naming (`query=<name>/`, `query_l2=<name>/`, ...) so that Arrow/DuckDB can read the result as a partitioned dataset. A `page` column is added to each record with a value derived from the source filename (or subdirectory for multi-query inputs).

See Also

`pro_request()` to download the JSON files, `pro_request_jsonl_R()` and `pro_request_jsonl_parquet()` for the older two-step pipeline (now deprecated).

`pro_validate_credentials` *Validate OpenAlex credentials*

Description

Makes a minimal API request to verify that the api_key is valid.

Usage

```
pro_validate_credentials(api_key = pro_api_key(), show_credentials = FALSE)
```

Arguments

`api_key`	API key to validate (character string) or 'NULL'.
`show_credentials`	shows the api_key using 'message()'. USE WITH CAUTION!

Value

TRUE if credentials work, FALSE otherwise

`read_corpus` *Read corpus from Parquet Dataset*

Description

This function reads a corpus in Apache Parquet format and returns an `ArrowObject` representing the corpus which can be fed into a `dplyr` pipeline or a `tibble` which contains all the data.

Usage

```
read_corpus(corpus, return_data = FALSE)
```

Arguments

corpus	The directory of the Parquet files.
return_data	Logical indicating whether to return an ArrowObject representing the corpus (default) or a tibble containing the whole corpus shou,d be returned.

Value

An ArrowObject representing the corpus or a tibble.

---

sample_parquet_n *Sample rows from Parquet files using DuckDB reservoir sampling*

Description

Draw a uniform random sample of n rows from one or more Parquet files using DuckDB's SQL USING SAMPLE reservoir(n ROWS) clause. The sampling is performed entirely inside DuckDB, so the full dataset is never loaded into R.

This is well-suited for large Parquet corpora (e.g. OpenAlex works) where you want a random subset of rows without materialising the whole table.

Usage

```
sample_parquet_n(path, n, seed = NULL, con = NULL, select = NULL)
```

Arguments

path	Character scalar. Path or glob pointing to one or more Parquet files, as understood by DuckDB's parquet_scan() table function. For example, "spc_corpus/output/chapter_3/co
n	Integer scalar. Number of rows to sample. If n is larger than the total number of rows in the dataset, DuckDB returns all rows.
seed	Optional integer scalar. If supplied, a REPEATABLE(seed) clause is added to the DuckDB query so that repeated calls with the same input data and seed return the same sample. If NULL (default), the sample is not forced to be reproducible at the DuckDB level.
con	Optional DBIConnection to an existing DuckDB database. If NULL (the default), the function creates a temporary in-memory DuckDB instance, uses it for the query, and shuts it down before returning. If a connection is supplied, it is left open and not modified beyond running the sampling query.
select	Optional character vector of column names to return. If NULL (default), all columns are returned (equivalent to SELECT *). Column names are passed through DBI::dbQuoteIdentifier() to safely handle special characters. If any requested column does not exist in the Parquet schema, DuckDB will raise an error.

Details

The function delegates to the following SQL pattern (simplified):

```
SELECT [columns]
FROM parquet_scan('path/to/files/*.parquet')
USING SAMPLE reservoir(n ROWS)
[REPEATABLE (seed)]
```

Using `reservoir(n ROWS)` gives an exact uniform sample of size `n` from all rows in the dataset (unless `n` exceeds the total row count, in which case all rows are returned).

Note that the `path` argument is passed directly to DuckDB's `parquet_scan()` function, so you can use:

- A single Parquet file:
  - `"works.parquet"`
- A glob for many files:
  - `"works/*.parquet"`
- A directory, depending on your DuckDB version/configuration.

When `con` is `NULL`, the function creates an in-memory DuckDB database. If you want to reuse the same DuckDB instance for multiple queries (for performance reasons or to control pragmas), you can create a DuckDB connection yourself and pass it via `con`.

Value

A `data.frame` with up to `n` rows, containing a uniform random sample from the union of all Parquet files matched by `path`, restricted to the columns specified in `select` (or all columns if `select` is `NULL`).

Examples

```
## Not run:
# Sample 1,000 rows from a directory of Parquet files
sample_df <- sample_parquet_duckdb(
  path = "spc_corpus/output/chapter_3/corpus/*.parquet",
  n = 10000L,
  seed = 1234
)

# Sample only a subset of columns
sample_df_small <- sample_parquet_duckdb(
  path = "spc_corpus/output/chapter_3/corpus/*.parquet",
  n = 10000L,
  seed = 1234,
  select = c("id", "doi", "citation", "author_abbr", "display_name", "ab")
)

# Reuse a DuckDB connection for multiple samples
```

```
con <- DBI::dbConnect(duckdb::duckdb())
on.exit(DBI::dbDisconnect(con, shutdown = TRUE), add = TRUE)

s1 <- sample_parquet_duckdb(
  path = "openalex_works/*.parquet",
  n = 500L,
  seed = 42,
  con = con
)

s2 <- sample_parquet_duckdb(
  path = "openalex_works/*.parquet",
  n = 500L,
  seed = 777,
  con = con
)

## End(Not run)
```

`snapshot_to_parquet` *Convert OA snapshot to Parquet format*

Description

Moved to the **openalexSnapshot package.**

This function has been removed from **openalexPro**. Please install the **openalexSnapshot** package and call ‘openalexSnapshot::snapshot_to_parquet()‘ instead.

Usage

```
snapshot_to_parquet(...)
```

Arguments

`...`	Ignored.

See Also

https://github.com/rkrug/openalexSnapshot

Index