Package: abstractExtractR (via r-universe)

November 8, 2026

Type Package

Title Retrieve and Audit Scholarly Metadata from Multiple Sources

Version 0.3.0

Description Retrieves and normalizes scholarly article metadata and abstracts from the 'Crossref' REST API <https://www.crossref.org/documentation/retrieve-metadata/rest-api/>, the 'OpenAlex' API <https://help.openalex.org/> and the 'Scopus' Search API <https://dev.elsevier.com/sc_apis.html>. Provides bibliographic searches, cursor pagination, publication date and author filtering, abstract retrieval by Digital Object Identifier (DOI), cached batch enrichment, merging of duplicate records, metadata-quality audits, reproducible search snapshots, local archives with integrity verification, and export to comma-separated values (CSV), 'BibTeX' and 'RIS' citation files. Network requests support configurable timeouts and test callbacks. 'Scopus' searches require an 'Elsevier' API key; 'Crossref' queries do not. 'OpenAlex' supports limited keyless searches and authenticated pagination.

License MIT + file LICENSE

Depends R (>= 4.2.0)

Imports httr2, jsonlite, openssl

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

NeedsCompilation no

Author Diogo Ribeiro [aut, cre, cph] (ORCID: <https://orcid.org/0009-0001-2022-7072>, affiliation: Faculty of Media Arts and Design of the Technical University of Porto)

Maintainer Diogo Ribeiro <dfr@esmad.ipp.pt>

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

Date/Publication 2026-11-08 19:51:04 UTC

RemoteUrl https://github.com/cran/abstractExtractR

RemoteRef HEAD

RemoteSha 399d350f7568953a62a46b2886303ec31a2b22c8

Contents

append_search_archive . . . 3
Article . . . 4
article_metadata_timeline . . . 4
articles_to_bibtex . . . 6
articles_to_data_frame . . . 6
articles_to_ris . . . 7
assess_article_quality . . . 8
compare_recent_archive_runs . . . 9
compare_search_archive_runs . . . 9
compare_search_history . . . 10
compare_search_snapshots . . . 11
create_search_archive . . . 12
deduplicate_articles . . . 13
enrich_articles_with_crossref . . . 14
export_articles . . . 15
fetch_all_from_scopus . . . 16
fetch_from_crossref . . . 17
fetch_from_openalex . . . 18
fetch_from_scopus . . . 19
get_abstract_from_doi . . . 20
get_abstracts_from_dois . . . 21
list_search_archive_runs . . . 22
new_crossref_cache . . . 23
new_search_plan . . . 24
normalize_doi . . . 25
process_crossref_entries . . . 26
process_crossref_response . . . 26
process_openalex_entries . . . 27
process_openalex_response . . . 27
process_scopus_entries . . . 28
process_scopus_response . . . 29
read_search_archive . . . 29
read_search_archive_run . . . 30
read_search_snapshot . . . 31
reconstruct_openalex_abstract . . . 32
recover_search_archive . . . 32
refresh_search_archive . . . 33
refresh_search_snapshot . . . 34
run_search_plan . . . 36
save_search_snapshot . . . 37

search_articles 38
search_crossref 40
search_openalex 42
search_snapshot_manifest 43
trace_archive_dois 44
verify_search_archive 45
write_search_history_report 46
write_search_manifest 47

Index **48**

`append_search_archive` *Append an observation to a research archive*

Description

Appends a SearchSnapshot to a new consecutively numbered run directory. The SearchPlan must match the archived plan, and completed timestamps must be nondecreasing. RDS and JSON evidence are written to a temporary sibling folder before the new run is published. Prior observations are unchanged. All existing runs have their checksums and JSON search-plan metadata inspected, but only the latest archived RDS is deserialized to validate chronology and plan equality. Call `read_search_archive()` to validate every historical RDS and generate the full SearchHistory. Cooperative writer locking is held during the operation; do not force-unlock an active process.

Usage

```
append_search_archive(snapshot, directory)
```

Arguments

`snapshot`	New SearchSnapshot or trusted RDS file path.
`directory`	Existing archive directory.

Value

New run directory path invisibly.

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snapshot <- run_search_plan(plan, requester = mock)
path <- tempfile("research-archive-")
create_search_archive(snapshot, path)
append_search_archive(snapshot, path)
unlink(path, recursive = TRUE)
```

`Article` *Construct a normalized scholarly article*

Description

Create a validated S3 Article record. Optional metadata is stored as NULL when absent, while authors are stored in source order as a character vector.

Usage

```
Article(
  title = NULL,
  abstract = NULL,
  publication_date = NULL,
  authors = character(),
  doi = NULL
)
```

Arguments

`title`, `abstract`, `publication_date`, `doi`	Optional scalar strings or NULL.
`authors`	Character vector of author names in source order.

Value

A validated list with class Article.

Examples

```
article <- Article(
  title = "Example paper",
  abstract = "Example abstract",
  authors = c("Ada Lovelace", "Grace Hopper"),
  doi = "10.1000/example"
)
article$title
```

`article_metadata_timeline` *Follow scholarly article metadata across archived searches*

Description

Uses canonical DOI identity and archive manifest indexes to find observed runs, loading only the matching RDS files. Tracks title, publication date, authors, abstract presence and SHA-256 fingerprint, source attribution, and metadata changes between successive *observations*. A missing DOI in an intervening run is described as an observation gap, not a retraction.

Usage

```
article_metadata_timeline(
  directory,
  doi,
  include_abstract = FALSE,
  on_unindexed = c("read", "error"),
  verify_integrity = TRUE
)
```

Arguments

directory	Existing research archive directory.
doi	A single valid DOI or doi.org resolver URL.
include_abstract	Logical; include full abstract text when TRUE.
on_unindexed	Either "read" (default) or "error" for older manifests.
verify_integrity	Verify archive checksums, TRUE by default.

Details

By default full abstract text is excluded. SHA-256 fingerprints indicate exact textual changes, not their significance or scientific validity. Provider failures and varying coverage limit interpretation.

Value

List with doi, observations, changes, gaps, summary and integrity_checked. Timestamps are search-observation times.

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list(
    list(title = list("Study"), DOI = "10.1000/example")
  )))
}
snap <- run_search_plan(plan, requester = mock)
path <- tempfile("doi-history-")
create_search_archive(snap, path)
article_metadata_timeline(path, "10.1000/example")$summary
unlink(path, recursive = TRUE)
```

articles_to_bibtex *Render article records as BibTeX*

Description

Returns UTF-8 BibTeX using one article entry per record. Titles, author names, DOIs and abstracts are escaped for LaTeX; missing fields are omitted. Citation keys derive from DOIs where possible and are made unique. Classical BibTeX may need LaTeX accent macros for non-ASCII characters; this function intentionally preserves the original Unicode characters.

Usage

```
articles_to_bibtex(articles, keys = NULL)
```

Arguments

articles	List of `Article()` records.
keys	Optional unique vector of BibTeX citation identifiers.

Value

A length-one character string containing BibTeX text, or an empty string for zero articles.

Examples

```
articles_to_bibtex(list(Article(
  title = "An example", authors = "Ada Lovelace", doi = "10.1000/example"
)))
```

articles_to_data_frame *Convert article records to a data frame*

Description

Converts normalized records from Scopus or Crossref into a rectangular representation. Missing scalar fields and missing authors become NA, and authors are joined in source order with a configurable separator. Embedded XML markup in abstracts is preserved.

Usage

```
articles_to_data_frame(articles, author_separator = "; ")
```

Arguments

`articles`	List of `Article()` records.
`author_separator`	Scalar character separator for the authors column.

Value

A base R data.frame containing title, abstract, publication_date, authors, and doi, in that order. Empty input produces zero rows with those same character columns.

Examples

```
x <- list(Article(title = "Paper", authors = c("Ada", "Grace"),
                  doi = "10.1000/example"))
articles_to_data_frame(x)
```

`articles_to_ris` *Render article records as RIS citations*

Description

Produces RIS records with TY/ER delimiters, one AU tag per author, and optional TI, PY, DA, DO, and AB tags. The format uses CRLF line endings, including after the final ER tag.

Usage

```
articles_to_ris(articles)
```

Arguments

`articles`	List of `Article()` records.

Value

A length-one character string containing RIS records, or an empty string for zero articles.

Examples

```
articles_to_ris(list(Article(title = "Paper", authors = "Ada")))
```

`assess_article_quality` *Evaluate structural quality of scholarly article metadata*

Description

Computes weighted completeness (fields present) and weighted validity (fields present and passing structural validation), plus a DOI duplicate indicator. This is a metadata audit, not a measure of scientific reliability, publication quality, or factual correctness. No HTTP requests are made.

Usage

```
assess_article_quality(
  articles,
  weights = c(title = 1, abstract = 1, publication_date = 1, authors = 1, doi = 1)
)
```

Arguments

`articles`	List of Article records, including an empty list.
`weights`	Named numeric vector with exactly the five field names; finite nonnegative values, at least one positive.

Details

The default weights are equal across title, abstract, publication_date, authors, and doi. Scores are normalized to the closed interval from 0 to 1. Blank values are missing; malformed DOI and dates are present but invalid. DOI duplication is reported separately and does not affect scores.

Value

A list with records (one row per article, scores and flags), summary (per-field counts and rates), issues (field-level missing/invalid diagnostics), and weights (normalized scoring weights).

Examples

```
papers <- list(Article(title = "A paper", doi = "10.1000/example"))
assess_article_quality(papers)$records
```

compare_recent_archive_runs

Compare only the most recent archive observations

Description

Load a bounded tail of completed runs and compare consecutive observations. This does not load or checksum earlier runs and is not a full archive audit.

Usage

```
compare_recent_archive_runs(directory, last_n = 2L)
```

Arguments

directory	Existing research archive.
last_n	Number of final runs to compare, at least two and no larger than the current archive length.

Value

A SearchHistory as produced by compare_search_history().

Examples

```
plan <- new_search_plan("topic", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snap <- run_search_plan(plan, requester = mock)
path <- tempfile("archive-")
create_search_archive(snap, path)
append_search_archive(snap, path)
compare_recent_archive_runs(path, last_n = 2L)$transitions
unlink(path, recursive = TRUE)
```

compare_search_archive_runs

Compare two selected archive observations

Description

Load and validate exactly the two requested observations, then perform the existing DOI-based comparison. Other archive RDS objects are not deserialized or checked. Use read_search_archive() for a full audit.

Usage

```
compare_search_archive_runs(directory, before, after = "latest")
```

Arguments

directory	Existing research archive path.
before	One-based index or run_NNNNNN name of the earlier run.
after	One-based index, run_NNNNNN name, or "latest".

Value

The result of compare_search_snapshots().

Examples

```
plan <- new_search_plan("topic", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snap <- run_search_plan(plan, requester = mock)
path <- tempfile("archive-")
create_search_archive(snap, path)
append_search_archive(snap, path)
compare_search_archive_runs(path, 1L, "latest")$summary
unlink(path, recursive = TRUE)
```

---

compare_search_history

Compare a chronological series of literature-search snapshots

---

Description

Compare successive SearchSnapshot observations using canonical DOI equality. Each transition reports newly observed and disappeared DOIs, modified metadata fields, changed source provenance and records that cannot be matched because they have no recognizable DOI.

Usage

```
compare_search_history(snapshots, labels = NULL)
```

Arguments

snapshots	An ordered list of at least two SearchSnapshot objects or trusted RDS paths from save_search_snapshot(). Do not read untrusted serialized files. Runs must be in nondecreasing finished_at UTC order.
labels	Optional unique nonempty names for the runs. Defaults to list names, or run_001, run_002, etc. when unnamed.

Details

Alongside the change events, each run records the provider retrieval counts and mean metadata completeness/structural-validity scores. These are properties of retrieved metadata, not measures of scientific quality.

Search settings and provider completeness are assessed for each transition. A disappeared DOI does *not* establish withdrawal, retraction or deletion. Settings and provider coverage may explain observed differences.

Value

A SearchHistory containing schema_version, runs, providers, transitions, events, and unmatched tables.

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snapshot <- run_search_plan(plan, requester = mock)
compare_search_history(list(snapshot, snapshot))$transitions
```

---

compare_search_snapshots

Compare two literature-search snapshots conservatively

Description

Uses only normalized DOI identity to compare scholarly Article records. Missing-DOI records are listed separately as unmatched, not paired by title or author. Reports added and removed DOIs and field-level changes for title, abstract, publication_date, authors and DOI.

Usage

```
compare_search_snapshots(before, after)
```

Arguments

`before`, `after`	Valid SearchSnapshot objects (or trusted RDS paths returned by save_search_snapshot()).

Details

Provenance changes compare provider-name sets, not row positions or counts. Comparison is descriptive: disappearance does not imply retraction, since query settings, coverage, availability and provider failures can differ.

Value

List containing summary, added, removed, modified, provenance, unmatched, and comparability diagnostics.

Examples

```
plan <- new_search_plan("test", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snap <- run_search_plan(plan, requester = mock)
compare_search_snapshots(snap, snap)$summary
```

---

create_search_archive *Create an append-only research archive from a saved search*

Description

Initializes a new local research collection with a first SearchSnapshot observation. The archive contains separate run directories with RDS evidence and credential-free JSON manifests, and does not call APIs. Files are staged in a temporary sibling directory before publication.

Usage

```
create_search_archive(snapshot, directory)
```

Arguments

snapshot	Valid SearchSnapshot or a trusted RDS snapshot path.
directory	Path to a new archive directory; parent must exist.

Details

Existing directories are never intentionally overwritten. Archive files are not cryptographically protected against later manual modification. Do not read untrusted RDS files.

Value

Archive directory path invisibly.

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snapshot <- run_search_plan(plan, requester = mock)
path <- tempfile("research-archive-")
create_search_archive(snapshot, path)
unlink(path, recursive = TRUE)
```

`deduplicate_articles` *Deduplicate and reconcile scholarly articles using exact DOI identity*

Description

Combines Scopus and Crossref Article records by normalized DOI, preserving first-occurrence order and list names. Records with missing or malformed DOIs remain separate, even when titles match.

Usage

```
deduplicate_articles(
  articles,
  on_conflict = c("keep_first", "warn", "error"),
  report = FALSE
)
```

Arguments

`articles`	List of Article records, such as the concatenation of normalized Scopus and Crossref results.
`on_conflict`	One of "keep_first" (default), "warn" or "error".
`report`	If TRUE, return an audit report as well as the articles.

Details

The first record has field priority. Later duplicates fill missing titles, abstracts and authors, and may refine compatible partial dates (2024 vs. 2024-06-03). Differing populated values are reported, not silently mixed. Returned valid DOI fields are normalized; input records remain unchanged.

Value

Reconciled list of Article records, or a list of three components when report is TRUE: articles, mapping and conflicts. The mapping includes input_index, output_index, original_doi, canonical_doi, status. Status is unique, duplicate or unkeyed. Conflicts identify which fields were retained and ignored, without losing the original input.

Examples

```
scopus <- list(Article(title = "Study", doi = "10.1000/Example"))
crossref <- list(Article(
  abstract = "Recovered", doi = "https://doi.org/10.1000/example"
))
deduplicate_articles(c(scopus, crossref), report = TRUE)
```

`enrich_articles_with_crossref` *Fill missing article abstracts from Crossref*

Description

Enrich normalized Scopus `Article()` records using batch DOI retrieval. Only missing or whitespace-only abstracts are fetched by default. Records, order, and list names are preserved, and duplicate DOI lookups are avoided. Crossref JATS/XML markup is preserved unmodified.

Usage

```
enrich_articles_with_crossref(
  articles,
  overwrite = FALSE,
  on_error = c("stop", "warn", "skip"),
  timeout = .DEFAULT_TIMEOUT,
  requester = NULL,
  cache = NULL,
  mailto = NULL,
  max_tries = 3L,
  min_interval = 0.25,
  sleep = Sys.sleep
)
```

Arguments

`articles`	List of Article objects.
`overwrite`	If TRUE, replace existing abstracts only when Crossref supplies a non-blank string.
`on_error`	Either "stop", "warn" or "skip" for Crossref failures.
`timeout`	Positive HTTP request timeout in seconds.
`requester`	Optional HTTP callback taking url, headers, params and timeout. NULL (default) enables bounded httr2 HTTP retries.
`cache`	Optional reusable environment from `new_crossref_cache()`.
`mailto`	Optional Crossref contact email.
`max_tries`	Total number of HTTP attempts (1 to 10); with custom requester callbacks, retries are controlled by that callback.
`min_interval`	Delay in seconds between distinct uncached DOI lookups.
`sleep`	Injectable delay function, defaults to `Sys.sleep()`.

Value

Updated list of Article records with existing metadata preserved.

Examples

```
articles <- list(Article(title = "Example", doi = "10.1000/example"))
mock <- function(url, headers, params, timeout) {
  list(message = list(abstract = "Example abstract"))
}
enrich_articles_with_crossref(articles, requester = mock)
```

`export_articles` *Export article records to CSV, BibTeX, or RIS*

Description

Writes one file in the selected format. CSV uses UTF-8 text, column names, and base R's standard CSV quoting rules. By default CSV fields beginning with potential spreadsheet formula operators are prefixed with a literal apostrophe for safer opening in spreadsheet applications. This protection affects CSV output only, not the original article records or data frame.

Usage

```
export_articles(
  articles,
  file,
  format = c("csv", "bibtex", "ris"),
  author_separator = "; ",
  keys = NULL,
  sanitize_csv = TRUE
)
```

Arguments

`articles`	List of `Article()` records.
`file`	Nonempty target file path. Existing files are replaced.
`format`	Either "csv", "bibtex", or "ris".
`author_separator`	Scalar authors separator used in CSV only.
`keys`	Optional BibTeX citation keys; ignored for other formats.
`sanitize_csv`	Logical. Protect cells beginning with =, +, -, or @.

Details

RIS files are written with CRLF line endings as prescribed by RIS. BibTeX is written as UTF-8, so classic BibTeX may need extra encoding conversion for accented characters.

Value

The file path invisibly.

Examples

```
example <- list(Article(title = "Example", doi = "10.1000/example"))
target <- tempfile(fileext = ".bib")
export_articles(example, target, format = "bibtex")
unlink(target)
```

`fetch_all_from_scopus` *Fetch all Scopus results through cursor pagination*

Description

Request successive Scopus pages until no next cursor is returned. A repeated cursor is rejected to prevent infinite loops.

Usage

```
fetch_all_from_scopus(
  query,
  api_key,
  start_date = NULL,
  end_date = NULL,
  author = NULL,
  timeout = .DEFAULT_TIMEOUT,
  requester = .http_get_json
)
```

Arguments

`query`	Non-empty Scopus search expression.
`api_key`	Non-empty Elsevier API key. Do not store keys in source code.
`start_date, end_date`	Optional ISO date bounds (YYYY-MM-DD). Both must be supplied together.
`author`	Optional non-empty author expression appended to the query.
`timeout`	Positive request timeout in seconds.
`requester`	Optional HTTP callback accepting url, headers, params, and timeout. Primarily useful for deterministic tests.

Value

A list of raw Scopus entry objects, in page order.

Examples

```
mock <- function(url, headers, params, timeout) {
  list("search-results" = list(
    entry = list(list("dc:title" = "Example")), cursor = list()
  ))
}
entries <- fetch_all_from_scopus(
  "forecasting", "example-key", requester = mock
)
length(entries)
```

---

fetch_from_crossref *Fetch one page of Crossref bibliographic search results*

Description

Search the public Crossref `/works` endpoint without an API key. Uses the supported `query.bibliographic` parameter, which searches bibliographic metadata (not full article text). Crossref can omit abstracts.

Usage

```
fetch_from_crossref(
  query,
  rows = 20L,
  start_date = NULL,
  end_date = NULL,
  work_type = NULL,
  cursor = NULL,
  mailto = NULL,
  timeout = .DEFAULT_TIMEOUT,
  requester = .http_get_json
)
```

Arguments

`query`	Non-empty bibliographic query, such as a paper title or topic.
`rows`	Number of records requested for the page, from 1 to 1000.
`start_date, end_date`	Optional inclusive publication date bounds, in `YYYY-MM-DD` format. Either or both bounds may be specified.
`work_type`	Optional Crossref work type such as "journal-article".
`cursor`	Optional Crossref cursor; use "*" for the first cursor page.
`mailto`	Optional contact email for Crossref's polite API pool.
`timeout`	Positive HTTP timeout in seconds.
`requester`	Optional callback accepting url, headers, params, timeout. Defaults to the package HTTP transport. Useful for offline tests.

Value

Raw Crossref response as a nested R list.

Examples

```
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list(list(title = list("Example")))))
}
fetch_from_crossref("forecasting", requester = mock)
```

`fetch_from_openalex` *Request one page of OpenAlex works*

Description

Search the Works endpoint with the current `search` and `per-page` parameters. Filters use OpenAlex work types and publication dates. Basic, keyless queries are supported; a free API key is necessary for cursor paging after the first page.

Usage

```
fetch_from_openalex(
  query,
  api_key = NULL,
  per_page = 25L,
  page = NULL,
  cursor = NULL,
  start_date = NULL,
  end_date = NULL,
  work_type = NULL,
  timeout = .DEFAULT_TIMEOUT,
  requester = .http_get_json
)
```

Arguments

`query`	Non-empty OpenAlex works search expression.
`api_key`	Optional OpenAlex API key; required for multiple cursor pages.
`per_page`	Number of results requested, between 1 and 100.
`page`	Optional one-based basic page; cannot combine with cursor.
`cursor`	Optional cursor. Use "*" to start cursor pagination.
`start_date, end_date`	Optional inclusive YYYY-MM-DD publication dates.
`work_type`	Optional OpenAlex type such as "article" or "review".
`timeout`	Positive request timeout in seconds.
`requester`	Injectable callback taking url, headers, params, timeout.

Details

Credentials, if provided, are sent in the Authorization header rather than the query string. The requester callback supports offline testing.

Value

Raw OpenAlex response as a nested list.

Examples

```
mock <- function(url, headers, params, timeout) {
  list(meta = list(count = 0), results = list())
}
fetch_from_openalex("gaussian processes", requester = mock)
```

---

fetch_from_scopus *Fetch one page of Scopus search results*

Description

Query the Scopus Search API. The function returns the raw JSON response as nested lists and does not normalize individual article records.

Usage

```
fetch_from_scopus(
  query,
  api_key,
  max_results = .MAX_PAGE_SIZE,
  timeout = .DEFAULT_TIMEOUT,
  requester = .http_get_json
)
```

Arguments

query	Non-empty Scopus search expression.
api_key	Non-empty Elsevier API key. Do not store keys in source code.
max_results	Integer page size, between 1 and 25.
timeout	Positive request timeout in seconds.
requester	Optional HTTP callback accepting url, headers, params, and timeout. Primarily useful for deterministic tests.

Value

Raw Scopus JSON object decoded into an R list.

Examples

```
mock <- function(url, headers, params, timeout) {
  list("search-results" = list(entry = list(list("dc:title" = "Example"))))
}
fetch_from_scopus("forecasting", "example-key", requester = mock)
```

`get_abstract_from_doi` *Retrieve an abstract from Crossref using its DOI*

Description

Looks up a DOI via the Crossref works endpoint. Abstracts can include JATS/XML markup, which is returned without modification. Crossref does not have abstracts for every indexed article.

Usage

```
get_abstract_from_doi(
  doi,
  timeout = .DEFAULT_TIMEOUT,
  requester = .http_get_json,
  mailto = NULL
)
```

Arguments

`doi`	Non-empty DOI, without the doi.org URL prefix.
`timeout`	Positive request timeout in seconds.
`requester`	Optional HTTP callback accepting url, headers, params, and timeout. Primarily useful for deterministic tests.
`mailto`	Optional contact email for Crossref's polite API pool.

Value

An abstract string as supplied by Crossref, or NULL if absent.

Examples

```
mock <- function(url, headers, params, timeout) {
  list(message = list(abstract = "Example abstract"))
}
get_abstract_from_doi("10.1000/example", requester = mock)
```

`get_abstracts_from_dois` *Retrieve a batch of Crossref abstracts by DOI*

Description

Looks up each distinct DOI at most once, preserving the input order. Both present and absent abstracts are cached, so repeated queries can be avoided by passing the same cache to multiple calls.

Usage

```
get_abstracts_from_dois(
  dois,
  cache = NULL,
  mailto = NULL,
  max_tries = 3L,
  min_interval = 0.25,
  on_error = c("stop", "warn", "skip"),
  timeout = .DEFAULT_TIMEOUT,
  requester = NULL,
  sleep = Sys.sleep
)
```

Arguments

`dois`	Character vector of DOIs. Missing or blank values yield NULL without a request.
`cache`	Optional environment, such as `new_crossref_cache()`. If NULL, a fresh per-call cache is used.
`mailto`	Optional email address identifying the caller to Crossref.
`max_tries`	Integer 1 to 10, including the initial HTTP request.
`min_interval`	Nonnegative minimum delay in seconds between distinct uncached DOI lookups. Defaults to 0.25 (four requests per second).
`on_error`	One of "stop", "warn", or "skip". Failed lookups are not cached across calls.
`timeout`	Positive HTTP request timeout in seconds.
`requester`	Optional callback accepting url, headers, params, timeout. NULL uses httr2 with retry policy; callbacks bypass retries.
`sleep`	Function used for delays; defaults to `Sys.sleep()`. Injectable for deterministic, network-free tests.

Details

Requests are sequential and spaced with min_interval seconds between uncached lookups. With the default HTTP requester, httr2 retries transient status codes (408, 429, 500, 502, 503, 504) and connection failures, honoring Retry-After or exponential backoff. Injected request callbacks are called directly without automatic HTTP retries.

Value

Named list with one entry per input DOI in input order. Entries contain a character string, or NULL for missing/failed abstracts. Names retain the original DOI values, including duplicates.

Examples

```
mock <- function(url, headers, params, timeout) {
  list(message = list(abstract = "Sample abstract"))
}
get_abstracts_from_dois(c("10.1000/example", "10.1000/example"),
                        requester = mock, min_interval = 0)
```

`list_search_archive_runs` *List research-archive runs without loading their saved RDS evidence*

Description

Read the small JSON manifests to inspect run dates, article counts, provider-record counts, and completeness across the archive. Only the manifests are decoded; snapshots and the full SearchHistory are not. Run ordering and plan consistency are validated.

Usage

```
list_search_archive_runs(directory, verify_integrity = TRUE)
```

Arguments

`directory`	Existing SearchArchive directory.
`verify_integrity`	Logical flag. TRUE (default) verifies available MD5/SHA-256 records and refuses invalid hashes. FALSE checks metadata, filenames and integrity-policy presence without reading RDS contents.

Details

With verify_integrity = TRUE (default), checksums for all runs are verified against the RDS and manifest files without deserializing RDS. Set this to FALSE for a faster metadata-only listing. An unchecked run is labeled "unchecked", not "verified"; missing integrity records on protected runs are rejected in either mode. The fast mode does not establish that manifests agree with the contents of the RDS files.

Use `read_search_archive()` for comprehensive evidence validation and full chronological history; use `read_search_archive_run()` to load one observation with its checksum and manifest verification.

Value

A data.frame with index, run, started_at, finished_at, articles, provider_records, complete, and integrity for each archived observation.

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
path <- tempfile("search-archive-")
create_search_archive(run_search_plan(plan, requester = mock), path)
list_search_archive_runs(path)
unlink(path, recursive = TRUE)
```

---

new_crossref_cache *Create a reusable in-memory Crossref abstract cache*

Description

Reusing the cache across calls avoids requesting the same DOI repeatedly. Both found and missing abstracts are cached. Network failures are not persisted in the cache.

Usage

```
new_crossref_cache()
```

Value

An environment suitable for the cache argument of `get_abstracts_from_dois()` and `enrich_articles_with_crossref()`

Examples

```
cache <- new_crossref_cache()
is.environment(cache)
```

new_search_plan *Create a reproducible multi-provider search plan*

Description

Capture the arguments of `search_articles()` without any credentials or executable request callbacks. A plan can be reused to run a new search; results are retrieved only when `run_search_plan()` is called.

Usage

```
new_search_plan(
  query,
  sources = c("crossref", "openalex"),
  max_results_per_source = 25L,
  page_size = 25L,
  start_date = NULL,
  end_date = NULL,
  crossref_work_type = NULL,
  openalex_work_type = NULL,
  scopus_author = NULL,
  on_error = c("stop", "warn", "skip"),
  on_conflict = c("keep_first", "warn", "error"),
  timeout = .DEFAULT_TIMEOUT,
  max_tries = 3L,
  min_interval = 0.25
)
```

Arguments

`query`	Nonempty scholarly search query.
`sources`	Unique provider names, in precedence order.
`max_results_per_source`	Maximum results per provider (1 to 500).
`page_size`	Maximum results per page (1 to 25).
`start_date, end_date`	Optional YYYY-MM-DD publication date limits.
`crossref_work_type`	Optional Crossref work type.
`openalex_work_type`	Optional OpenAlex work type.
`scopus_author`	Optional Scopus author expression.
`on_error`	Provider failure policy: stop, warn, or skip.
`on_conflict`	Reconciliation policy: keep_first, warn, or error.

timeout	HTTP timeout in seconds, positive and finite.
max_tries	Retry attempts for the built-in transport (1 to 10).
min_interval	Nonnegative request spacing in seconds.

Value

A validated SearchPlan list containing no credentials.

Examples

```
plan <- new_search_plan("causal inference", sources = "crossref")
plan$query
```

---

normalize_doi *Normalize DOI identifiers for equality comparisons*

Description

Accepts bare DOIs, doi: and urn:doi: identifiers and doi.org resolver URLs. URI representations are decoded once; bare percent signs are preserved. Matching is case-insensitive only for ASCII A-Z per DOI rules. The function performs no network calls and does not verify DOI registration.

Usage

```
normalize_doi(dois)
```

Arguments

dois	Character vector of DOI values, including NA when missing.

Value

Character vector of canonical DOI strings, NA for invalid values.

Examples

```
normalize_doi(c(" DOI:10.1000/Example ",
                "https://doi.org/10.1000/example"))
```

process_crossref_entries *Normalize raw Crossref works into Article objects*

Description

Convert Crossref metadata into the same `Article()` format as Scopus. Incomplete publication dates retain their original precision (e.g. `"2024"` or `"2024-08"`). Non-record entries are ignored.

Usage

```
process_crossref_entries(entries)
```

Arguments

`entries`	A list of Crossref works metadata objects.

Value

A list of `Article` records.

Examples

```
works <- list(list(title = list("Example"), DOI = "10.1000/example"))
process_crossref_entries(works)
```

process_crossref_response *Normalize a Crossref works search response*

Description

Normalize a Crossref works search response

Usage

```
process_crossref_response(payload)
```

Arguments

`payload`	A Crossref response containing `message.items`.

Value

A list of normalized `Article` records.

Examples

```
payload <- list(message = list(items = list(
  list(title = list("Example"), DOI = "10.1000/example")
)))
process_crossref_response(payload)
```

`process_openalex_entries`

Normalize OpenAlex works as Article records

Description

Converts titles, inverted-index abstracts, dates, authorship display names and DOI resolver URLs into the package's standard Article representation. Missing fields remain NULL. A present malformed abstract index fails visibly to avoid silently returning truncated or invented text.

Usage

```
process_openalex_entries(entries)
```

Arguments

`entries`	List of OpenAlex works objects.

Value

List of normalized Article records, in source order.

Examples

```
works <- list(list(
  title = "Example", doi = "https://doi.org/10.1000/example",
  authorships = list(list(author = list(display_name = "Ada")))
))
process_openalex_entries(works)
```

`process_openalex_response`

Normalize an OpenAlex search response

Description

Normalize an OpenAlex search response

Usage

```
process_openalex_response(payload)
```

Arguments

payload	Raw OpenAlex response including a results array.

Value

List of Article records.

Examples

```
process_openalex_response(list(results = list(
  list(title = "A work", publication_date = "2025-01-02")
)))
```

---

process_scopus_entries

Normalize raw Scopus search entries

Description

Converts selected Dublin Core and PRISM fields into validated Article objects. Malformed entry and author objects are ignored.

Usage

```
process_scopus_entries(entries)
```

Arguments

entries	List of raw Scopus entry objects.

Value

A list of Article objects.

Examples

```
entries <- list(list("dc:title" = "Example", "prism:doi" = "10.1000/example"))
process_scopus_entries(entries)
```

process_scopus_response *Normalize a complete Scopus search response*

Description

Normalize a complete Scopus search response

Usage

```
process_scopus_response(payload)
```

Arguments

payload	Raw Scopus JSON object containing search-results.entry.

Value

A list of Article objects.

Examples

```
payload <- list("search-results" = list(
  entry = list(list("dc:title" = "Example"))
))
process_scopus_response(payload)
```

read_search_archive *Inspect an existing append-only research archive*

Description

Reads each run's trusted RDS evidence and checks the accompanying JSON manifest for matching timestamps, query, sources and article count. Rejects missing or nonsequential run folders and nonchronological results. An archive with at least two runs includes an automatically generated SearchHistory for all successive observations.

Usage

```
read_search_archive(directory)
```

Arguments

directory	Existing research archive directory.

Value

A SearchArchive containing directory, inventory, snapshots, and history (NULL when the archive contains only its initial observation).

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snapshot <- run_search_plan(plan, requester = mock)
path <- tempfile("research-archive-")
create_search_archive(snapshot, path)
read_search_archive(path)$inventory
unlink(path, recursive = TRUE)
```

`read_search_archive_run` *Read a single research-archive observation*

Description

Load and validate one run instead of deserializing the entire archive. The selected run's checksum and manifest are validated as with `read_search_archive()`. Other runs are not checked; use `list_search_archive_runs()` to audit their checksums or `read_search_archive()` to load and validate all evidence.

Usage

```
read_search_archive_run(directory, run = "latest")
```

Arguments

`directory`	Existing SearchArchive directory.
`run`	"latest" (default), a positive one-based integer run index, or a canonical name such as "run_000002".

Value

A validated SearchSnapshot for the selected run.

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
path <- tempfile("search-archive-")
```

```
snapshot <- run_search_plan(plan, requester = mock)
create_search_archive(snapshot, path)
identical(read_search_archive_run(path), snapshot)
unlink(path, recursive = TRUE)
```

`read_search_snapshot` *Read and validate a previously saved search snapshot*

Description

Reloads an RDS search snapshot, validating its schema version and evidence structure. Use only snapshot files from trusted sources. RDS is an R-native serialized format, not a safe interchange format for untrusted files.

Usage

```
read_search_snapshot(file)
```

Arguments

`file`	Path to an existing RDS search snapshot.

Value

A validated SearchSnapshot preserving articles and provenance.

Examples

```
plan <- new_search_plan("test", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snap <- run_search_plan(plan, requester = mock)
path <- tempfile(fileext = ".rds")
save_search_snapshot(snap, path)
restored <- read_search_snapshot(path)
unlink(path)
restored$context$complete
```

reconstruct_openalex_abstract

Restore an OpenAlex abstract from its inverted index

Description

OpenAlex represents available abstracts as tokens mapped to their zero-based positions. This function sorts those positions and joins the tokens with spaces. Missing or empty indexes yield NULL. Malformed positions, duplicate positions, or gaps cause an explicit error rather than silently fabricating text.

Usage

```
reconstruct_openalex_abstract(index)
```

Arguments

index	Named list of tokens mapped to integer position vectors, as returned in abstract_inverted_index, or NULL.

Value

A character abstract or NULL when no index is supplied.

Examples

```
reconstruct_openalex_abstract(list(
  evidence = 2L, Research = 0L, needs = 1L
))
```

recover_search_archive

Inspect and explicitly clean up interrupted archive staging directories

Description

By default no files are changed. With dry_run = FALSE, remove only hidden staging directories created by this package. If a writer lock remains, deletion is refused unless force_unlock = TRUE; that option is dangerous if any writer is still active.

Usage

```
recover_search_archive(directory, dry_run = TRUE, force_unlock = FALSE)
```

Arguments

directory	Existing search archive directory.
dry_run	TRUE (default) for inspection only.
force_unlock	Explicitly remove a writer lock when dry_run is FALSE.

Details

All staging paths are validated before any lock removal or deletion. Completed archive runs are never removed. Suspected checksum corruption must be investigated using a trusted backup instead of this function.

Value

List with pending, locked, removed, and unlocked fields.

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
path <- tempfile("archive-recovery-")
create_search_archive(run_search_plan(plan, requester = mock), path)
recover_search_archive(path)$pending
unlink(path, recursive = TRUE)
```

---

refresh_search_archive

Refresh an archive on demand and append the new search evidence

---

Description

Executes the SearchPlan from the most recent archived observation, using credentials and HTTP request callbacks supplied at invocation time. The new observation is appended only after search and snapshot validation. Provider errors obey the SearchPlan policy. This function does not create timers, recurring jobs, or background processes.

Usage

```
refresh_search_archive(
  directory,
  scopus_api_key = NULL,
  openalex_api_key = NULL,
  crossref_mailto = NULL,
  requester = NULL,
  sleep = Sys.sleep,
  clock = Sys.time,
  return_mode = c("full", "recent")
)
```

Arguments

directory	Existing search archive.
scopus_api_key	Optional runtime Scopus API key.
openalex_api_key	Optional runtime OpenAlex API key.
crossref_mailto	Optional runtime contact email for Crossref.
requester	Optional HTTP requester callback.
sleep	Injectable request-spacing function.
clock	Injectable zero-argument time function.
return_mode	"full" (default) preserves the original SearchArchive result, loading all snapshots and building a full SearchHistory after the append. "recent" returns a SearchArchiveRefresh containing the archive inventory, previous and new snapshots, and their two-run comparison. It does not deserialize older historical observations.

Value

A complete SearchArchive by default, or SearchArchiveRefresh with directory, inventory, previous, snapshot and comparison when return_mode is "recent".

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snapshot <- run_search_plan(plan, requester = mock)
path <- tempfile("research-archive-")
create_search_archive(snapshot, path)
updated <- refresh_search_archive(path, requester = mock)
updated$inventory
unlink(path, recursive = TRUE)
```

---

refresh_search_snapshot

Refresh a literature search and preserve its evidence

Description

Execute the SearchPlan contained in an earlier SearchSnapshot, compare the results with that observation and write a self-contained report directory. This is an explicit, synchronous operation; it does not install a scheduler or contact providers outside the requested search.

Usage

```
refresh_search_snapshot(
  previous,
  directory,
  scopus_api_key = NULL,
  openalex_api_key = NULL,
  crossref_mailto = NULL,
  include_values = FALSE,
  requester = NULL,
  sleep = Sys.sleep,
  clock = Sys.time
)
```

Arguments

previous	An existing SearchSnapshot or a trusted RDS path produced by save_search_snapshot(). Never read RDS from untrusted sources.
directory	Destination directory, which must not already exist. The parent directory must exist.
scopus_api_key	Optional Elsevier API key supplied at execution.
openalex_api_key	Optional OpenAlex API key supplied at execution.
crossref_mailto	Optional Crossref contact email.
include_values	If TRUE, include changed metadata text in CSV events; defaults to FALSE to exclude potentially long abstracts.
requester	Optional HTTP callback taking url, headers, params, timeout. Useful for deterministic offline tests.
sleep	Injectable delay callback (default Sys.sleep()).
clock	Injectable zero-argument time callback (default Sys.time()).

Details

The output contains the previous and current snapshots in RDS format, credential-free JSON manifests for both, and a CSV drift report. All files are staged in a temporary sibling directory and published only when every write succeeds. Existing output directories are never intentionally overwritten. The previous snapshot or its source RDS file is unchanged.

If the plan allows partial provider failures, the new snapshot still captures them; its comparison is marked incomparable when either run is incomplete. A missing DOI in the new results does not establish article withdrawal or retraction.

API keys and Crossref contact details are supplied only at execution time, and are excluded from the plan and generated manifests. The full RDS files intentionally preserve article text and provider error evidence: store them privately. Changed field values (including abstracts) are omitted from the CSV report unless include_values is TRUE.

Value

A SearchRefresh list containing directory, snapshot and history. The saved files are named before.rds, after.rds, before-manifest.json, after-manifest.json, and report/ (CSV drift tables and interpretation notes).

Examples

```
plan <- new_search_plan("test", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
before <- run_search_plan(plan, requester = mock)
path <- tempfile("search-refresh-")
refreshed <- refresh_search_snapshot(
  before, path, requester = mock, sleep = function(...) NULL
)
refreshed$history$transitions
unlink(path, recursive = TRUE)
```

---

run_search_plan *Execute a search plan and capture a versioned evidence snapshot*

Description

Executes search_articles() using a validated SearchPlan and saves the resulting normalized articles, provider counts and errors, DOI provenance and field-level conflicts. Records UTC start/end timestamps and runtime version information. Credentials are accepted at execution time only: values of API keys and Crossref contact addresses are not saved.

Usage

```
run_search_plan(
  plan,
  scopus_api_key = NULL,
  openalex_api_key = NULL,
  crossref_mailto = NULL,
  requester = NULL,
  sleep = Sys.sleep,
  clock = Sys.time
)
```

Arguments

plan	A SearchPlan from new_search_plan().
scopus_api_key	Optional Elsevier API key.
openalex_api_key	Optional OpenAlex API key.

crossref_mailto
: Optional Crossref contact address.

requester
: Optional shared HTTP callback for offline tests.

sleep
: Optional delay function, by default `Sys.sleep()`.

clock
: Zero-argument function returning the current time, by default `Sys.time()`. Injectable for deterministic tests.

Details

A snapshot is an observation of one search run, not a claim that a provider will return the same results later. Use `save_search_snapshot()` to persist the complete result for reproducible downstream analysis.

Value

A SearchSnapshot with schema_version, plan, started_at, finished_at, context (runtime information and completion status), and result (the five `search_articles()` output tables/objects).

Examples

```
plan <- new_search_plan("forecasting", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list(
    list(title = list("Example"), DOI = "10.1000/example")
  )))
}
snap <- run_search_plan(plan, requester = mock, sleep = function(...) NULL)
snap$result$sources
```

---

save_search_snapshot *Persist a search snapshot without overwriting existing evidence*

Description

Writes the full SearchSnapshot to an RDS file, using a temporary file in the same directory before renaming. Existing files are never intentionally overwritten. RDS preserves Article classes and empty table schemas. No network calls or additional R packages are required.

Usage

```
save_search_snapshot(snapshot, file)
```

Arguments

snapshot
: A valid SearchSnapshot from `run_search_plan()`.

file
: Nonempty path to a new RDS file. The parent directory must already exist. Existing files cause an error.

Value

The file path invisibly.

Examples

```
plan <- new_search_plan("test", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snap <- run_search_plan(plan, requester = mock)
path <- tempfile(fileext = ".rds")
save_search_snapshot(snap, path)
unlink(path)
```

---

search_articles *Search Scopus, Crossref, and OpenAlex with provenance*

Description

Fetches bounded result sets from the selected providers, normalizes them to Article records, and reconciles exact DOI matches. Each provider is queried in the requested order. Where populated fields disagree, the earlier provider's value is kept and the disagreement is reported.

Usage

```
search_articles(
  query,
  sources = c("crossref", "openalex"),
  max_results_per_source = 25L,
  page_size = 25L,
  start_date = NULL,
  end_date = NULL,
  scopus_api_key = NULL,
  openalex_api_key = NULL,
  crossref_mailto = NULL,
  crossref_work_type = NULL,
  openalex_work_type = NULL,
  scopus_author = NULL,
  on_error = c("stop", "warn", "skip"),
  on_conflict = c("keep_first", "warn", "error"),
  timeout = .DEFAULT_TIMEOUT,
  max_tries = 3L,
  min_interval = 0.25,
  requester = NULL,
  sleep = Sys.sleep
)
```

Arguments

`query`	Non-empty scholarly search expression.
`sources`	Non-empty vector containing unique elements of "crossref", "openalex", and "scopus". Order determines metadata precedence.
`max_results_per_source`	Maximum works fetched from each source, between 1 and 500. This is not a global result limit.
`page_size`	Maximum works per request (1 to 25); 25 respects the Scopus API page-size restriction.
`start_date, end_date`	Optional YYYY-MM-DD publication date bounds. Scopus searches require either both or neither.
`scopus_api_key`	Optional Elsevier API key; required for Scopus.
`openalex_api_key`	Optional OpenAlex API key; required when OpenAlex needs more than one cursor page.
`crossref_mailto`	Optional contact email for Crossref.
`crossref_work_type`	Optional Crossref type such as "journal-article".
`openalex_work_type`	Optional OpenAlex type such as "article".
`scopus_author`	Optional Scopus author expression.
`on_error`	Provider failure policy: "stop" (default), "warn" or "skip". A failed source is recorded in the results for warn/skip.
`on_conflict`	How to handle populated fields differing for one DOI: "keep_first" (default), "warn", or "error".
`timeout`	Positive timeout per HTTP request in seconds.
`max_tries`	Attempts for the built-in Crossref/OpenAlex retry transport (1 to 10). Scopus uses its standard request transport.
`min_interval`	Nonnegative delay between requests for each source.
`requester`	Optional shared HTTP callback receiving url, headers, params, timeout. Useful for deterministic network-free testing.
`sleep`	Injectable delay function, default Sys.sleep.

Details

The result includes a row for each retrieved record in provenance, showing its original source, position, canonical DOI, reconciliation status and final output index. DOI-free articles remain separate. No fuzzy matching or unsupported source-to-source field inferences are performed.

By default Crossref and OpenAlex are selected because they allow small queries without credentials. Scopus requires scopus_api_key. OpenAlex requires openalex_api_key for more than one cursor page. Source-specific type filters are separate because the provider taxonomies differ.

Value

List with articles (reconciled Article records), provenance (input-to-output mapping with source labels), conflicts (disagreements found during reconciliation), sources (one status row per requested provider), and errors (provider-level failure messages).

Examples

```
mock <- function(url, headers, params, timeout) {
  if (grepl("crossref", url, fixed = TRUE)) {
    return(list(message = list(items = list(
      list(title = list("Example"), DOI = "10.1000/a")
    ))))
  }
  list(results = list(
    list(title = "Example", doi = "https://doi.org/10.1000/a")
  ), meta = list())
}
result <- search_articles("example", requester = mock, min_interval = 0)
result$provenance
```

---

search_crossref *Discover articles from Crossref without known DOIs*

Description

Search Crossref bibliographic metadata and normalize the matching records. Uses cursor pagination to retrieve up to `max_results`; unlike a search that retrieves the entire Crossref index, this limit is finite.

Usage

```
search_crossref(
  query,
  max_results = 50L,
  page_size = 50L,
  start_date = NULL,
  end_date = NULL,
  work_type = NULL,
  mailto = NULL,
  timeout = .DEFAULT_TIMEOUT,
  max_tries = 3L,
  min_interval = 0.25,
  requester = NULL,
  sleep = Sys.sleep
)
```

Arguments

query	Non-empty bibliographic query, such as a paper title or topic.
max_results	Maximum normalized records to return. Defaults to 50.
page_size	Page size from 1 to 1000; defaults to 50.
start_date, end_date	Optional inclusive publication date bounds, in YYYY-MM-DD format. Either or both bounds may be specified.
work_type	Optional Crossref work type such as "journal-article".
mailto	Optional contact email for Crossref's polite API pool.
timeout	Positive HTTP timeout in seconds.
max_tries	Maximum HTTP attempts per page, 1 to 10, when using the default transport. Injected callback requesters bypass automatic retries.
min_interval	Nonnegative delay between page requests in seconds.
requester	Optional request callback; defaults to NULL, meaning bounded httr2 transient-error retries.
sleep	Injectable delay callback for offline tests.

Details

The cursor endpoint is paged until the requested number of works is collected, a short page is received, or no next cursor is available. Repeated cursors cause an error rather than an infinite loop. Existing Crossref abstracts are preserved as supplied, including JATS/XML.

Value

A list of normalized Article records in result order.

Examples

```
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list(
    list(title = list("Example paper"), DOI = "10.1000/example")
  )))
}
papers <- search_crossref("gaussian processes", requester = mock,
                          min_interval = 0)
papers[[1L]]$title
```

`search_openalex` *Discover scholarly articles from OpenAlex*

Description

Search OpenAlex with finite result limits and cursor pagination. For multi-page searches an OpenAlex API key is required; single-page searches can use the smaller keyless budget. Cursor loops and malformed pages raise errors. Request spacing avoids bursts of API traffic.

Usage

```
search_openalex(
  query,
  api_key = NULL,
  max_results = 25L,
  page_size = 25L,
  start_date = NULL,
  end_date = NULL,
  work_type = NULL,
  timeout = .DEFAULT_TIMEOUT,
  max_tries = 3L,
  min_interval = 0.25,
  requester = NULL,
  sleep = Sys.sleep
)
```

Arguments

`query`	Non-empty OpenAlex works search expression.
`api_key`	Optional OpenAlex API key; required for multiple cursor pages.
`max_results`	Maximum normalized results to return, at least 1.
`page_size`	Requested works per cursor page, between 1 and 100.
`start_date, end_date`	Optional inclusive YYYY-MM-DD publication dates.
`work_type`	Optional OpenAlex type such as "article" or "review".
`timeout`	Positive request timeout in seconds.
`max_tries`	Retry attempts for transient failures using httr2.
`min_interval`	Delay in seconds between page requests.
`requester`	Optional callback; NULL enables retry-capable httr2.
`sleep`	Injectable delay function for offline tests.

Value

List of normalized Article records.

Examples

```
mock <- function(url, headers, params, timeout) {
  list(results = list(list(title = "Example")), meta = list())
}
search_openalex("forecasting", requester = mock, min_interval = 0)
```

`search_snapshot_manifest` *Summarize a scholarly search as a structured manifest*

Description

Creates a small, credential-free summary of one validated SearchSnapshot: the search plan, UTC execution times, runtime versions, provider status counts, DOI coverage, and metadata reconciliation counts. Original article abstracts and provider error messages are excluded.

Usage

```
search_snapshot_manifest(snapshot)
```

Arguments

`snapshot`	SearchSnapshot from run_search_plan().

Details

A manifest describes retrieved evidence, not a reproducible guarantee that external providers will return the same results in a later search. For a complete record, preserve the source SearchSnapshot RDS as well.

Value

A SearchManifest list with schema, plan, context, counts, and provider summary.

Examples

```
plan <- new_search_plan("topic", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snapshot <- run_search_plan(plan, requester = mock)
search_snapshot_manifest(snapshot)$counts
```

`trace_archive_dois` *Trace DOI observations across a research archive*

Description

Locate normalized DOIs across all archived search runs using compact JSON manifest indexes rather than loading every RDS snapshot. The result records which run first and last observed each DOI, its position in that run, and the contributing provider names. Dates are search-observation timestamps, not publication, withdrawal or retraction dates.

Usage

```
trace_archive_dois(
  directory,
  dois,
  on_unindexed = c("read", "error"),
  verify_integrity = TRUE
)
```

Arguments

`directory`	Existing research archive directory.
`dois`	Character vector of one or more valid DOI representations. Equivalent DOIs in the input are deduplicated after normalization.
`on_unindexed`	Either "read" (default) or "error" for older manifests.
`verify_integrity`	Logical. TRUE (default) verifies all stored file checksums; FALSE performs a faster metadata-only scan.

Details

Manifests created before DOI indexes are supported: when on_unindexed is "read" (default), only those older RDS snapshots are loaded and validated to recover their DOI positions. Use "error" to forbid fallback reads.

The archive inventory is checked for chronological consistency. By default all archived RDS/manifest file checksums are verified without deserializing indexed runs. Setting verify_integrity = FALSE skips those checks for indexed runs and labels the returned result as unverified. Legacy RDS fallback always performs its available integrity checks.

No network queries are performed. Records without valid DOIs cannot be traced using this function.

Value

An ArchiveDOITrace list with occurrences (one row per matching archive run), summary (one row per requested DOI, including absent identifiers), legacy_runs (runs requiring RDS fallback), and the integrity_checked flag.

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list(
    list(title = list("Paper"), DOI = "10.1000/example")
  )))
}
snapshot <- run_search_plan(plan, requester = mock)
path <- tempfile("archive-")
create_search_archive(snapshot, path)
trace_archive_dois(path, "https://doi.org/10.1000/example")$summary
unlink(path, recursive = TRUE)
```

---

verify_search_archive *Inspect checksums and interrupted writes in a research archive*

Description

Inspects every run without deserializing RDS. New runs are verified with SHA-256, while older MD5 runs remain readable and are labeled verified_md5. Pre-checksum archives receive legacy_unverified status. A protected run that loses its integrity.json is reported as invalid, not as legacy.

Usage

```
verify_search_archive(directory)
```

Arguments

directory	Existing local archive directory.

Details

The archive root policy is local and unsigned: checksums do not establish authenticity against deliberate manipulation of both data and metadata.

Value

List of runs (run, integrity, issue), pending staging paths, and a logical locked flag.

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
path <- tempfile("archive-audit-")
create_search_archive(run_search_plan(plan, requester = mock), path)
verify_search_archive(path)$runs
unlink(path, recursive = TRUE)
```

`write_search_history_report` *Export an auditable search-history report to CSV files*

Description

Write six UTF-8 CSV tables (runs, providers, transitions, events, unmatched) and a short README describing interpretation limits. Provider error texts, API keys, and contact addresses are not included. By default, metadata before/after values are suppressed in the event table; opt in using include_values = TRUE. CSV cells are sanitized against spreadsheet formula injection. A new directory is required and existing reports are never intentionally overwritten.

Usage

```
write_search_history_report(history, directory, include_values = FALSE)
```

Arguments

`history`	A SearchHistory created by `compare_search_history()`.
`directory`	Path to a new report directory, within an existing parent.
`include_values`	Whether to include changed field values (which may include full abstracts) in events.csv. Defaults to FALSE.

Value

The report directory path invisibly.

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snapshot <- run_search_plan(plan, requester = mock)
history <- compare_search_history(list(snapshot, snapshot))
directory <- tempfile("search-history-")
write_search_history_report(history, directory)
unlink(directory, recursive = TRUE)
```

`write_search_manifest` *Write a structured JSON search manifest*

Description

Writes a small, human-readable UTF-8 JSON summary of a SearchSnapshot. The manifest intentionally excludes API keys, contact emails, article abstracts, and raw error messages. A temporary file is written in the destination directory before renaming; existing files are not intentionally overwritten.

Usage

```
write_search_manifest(snapshot, file)
```

Arguments

`snapshot`	Valid SearchSnapshot.
`file`	Path to a new JSON file. Its parent directory must exist.

Value

File path invisibly.

Examples

```
plan <- new_search_plan("example", sources = "crossref")
mock <- function(url, headers, params, timeout) {
  list(message = list(items = list()))
}
snapshot <- run_search_plan(plan, requester = mock)
target <- tempfile(fileext = ".json")
write_search_manifest(snapshot, target)
unlink(target)
```

Index

append_search_archive, 3
Article, 4
Article(), *6, 7*, *14, 15*, *26*
article_metadata_timeline, 4
articles_to_bibtex, 6
articles_to_data_frame, 6
articles_to_ris, 7
assess_article_quality, 8

compare_recent_archive_runs, 9
compare_search_archive_runs, 9
compare_search_history, 10
compare_search_history(), *9*, *46*
compare_search_snapshots, 11
compare_search_snapshots(), *10*
create_search_archive, 12

deduplicate_articles, 13

enrich_articles_with_crossref, 14
enrich_articles_with_crossref(), *23*
export_articles, 15

fetch_all_from_scopus, 16
fetch_from_crossref, 17
fetch_from_openalex, 18
fetch_from_scopus, 19

get_abstract_from_doi, 20
get_abstracts_from_dois, 21
get_abstracts_from_dois(), *23*

list_search_archive_runs, 22
list_search_archive_runs(), *30*

new_crossref_cache, 23
new_crossref_cache(), *14*, *21*
new_search_plan, 24
new_search_plan(), *36*
normalize_doi, 25

process_crossref_entries, 26
process_crossref_response, 26
process_openalex_entries, 27
process_openalex_response, 27
process_scopus_entries, 28
process_scopus_response, 29

read_search_archive, 29
read_search_archive(), *3*, *9*, *23*, *30*
read_search_archive_run, 30
read_search_archive_run(), *23*
read_search_snapshot, 31
reconstruct_openalex_abstract, 32
recover_search_archive, 32
refresh_search_archive, 33
refresh_search_snapshot, 34
run_search_plan, 36
run_search_plan(), *24*, *37*

save_search_snapshot, 37
save_search_snapshot(), *10*, *35*, *37*
search_articles, 38
search_articles(), *24*, *36, 37*
search_crossref, 40
search_openalex, 42
search_snapshot_manifest, 43
Sys.sleep(), *14*, *21*, *35*, *37*
Sys.time(), *35*, *37*

trace_archive_dois, 44

verify_search_archive, 45

write_search_history_report, 46
write_search_manifest, 47