Package: researchR (via r-universe)

November 3, 2026

Title Client for a Self-Hosted 'researchD' Literature Surveillance Server

Version 0.1.2

Date 2026-10-31

Description An R client for 'researchD', a self-hosted literature and signal surveillance server. Configures surveillance questions and reads stored literature records, signals, source health and service heartbeats through a token-authenticated REST API. Returns typed 'tibble' objects, including source provenance where supported by the server. Collection, classification, source access and delivery are managed by the server.

License MIT + file LICENSE

URL https://github.com/CTTIR/researchR

BugReports https://github.com/CTTIR/researchR/issues

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

Depends R (>= 4.1)

Imports cli, httr2, jsonlite, rlang, tibble, utils

Suggests knitr, rmarkdown, testthat (>= 3.2.0), withr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/Needs/website CTTIR/themakR, pkgdown

VignetteBuilder knitr

Config/pak/sysreqs libssl-dev

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

Date/Publication 2026-10-31 12:09:31 UTC

RemoteUrl https://github.com/CTTIR/researchR

RemoteRef HEAD

RemoteSha 9f4bb6daac9bcf232a5012e036e91589e43976f7

Contents

rr_concept . . . 2
rr_connect . . . 3
rr_delete_question . . . 4
rr_feed_url . . . 4
rr_health . . . 5
rr_heartbeat . . . 6
rr_meta . . . 7
rr_question . . . 7
rr_questions . . . 8
rr_records . . . 8
rr_set_signal_config . . . 10
rr_signal_config . . . 11
rr_signals . . . 12
rr_source_health . . . 13
rr_upsert_question . . . 14

Index **16**

`rr_concept` *Describe one concept block of a question*

Description

A question is a list of concepts, AND-joined at query time; the terms and MeSH headings within one concept are OR-joined. So a two-concept question `(semaglutide OR "GLP-1") AND (MASH OR steatohepatitis)` is two calls to this function.

Usage

```
rr_concept(name, terms = character(0), mesh = character(0))
```

Arguments

`name`	Human-readable concept name, e.g. `"drug"`.
`terms`	Character vector of free-text synonyms.
`mesh`	Character vector of MeSH headings.

Value

A list of class `researchr_concept`.

Examples

```
rr_concept("drug", terms = c("semaglutide", "GLP-1"))
```

`rr_connect` *Connect to a researchD server*

Description

Creates the connection object every other function in this package takes as its first argument. Nothing is sent at connection time unless `check = TRUE`. The default checks the server immediately; use `check = FALSE` to construct an offline connection.

Usage

```
rr_connect(
  base_url = Sys.getenv("RESEARCHR_URL"),
  token = Sys.getenv("RESEARCHR_TOKEN"),
  timeout = 60,
  check = TRUE
)
```

Arguments

`base_url`	Base URL of the researchD web service, e.g. `"https://feeds.example.org"`. Defaults to the `RESEARCHR_URL` environment variable.
`token`	API bearer token. Defaults to the `RESEARCHR_TOKEN` environment variable. Never hard-code it in a script.
`timeout`	Request timeout in seconds.
`check`	Whether to contact the server and verify the token now.

Value

An object of class `researchr_conn`.

Authentication

researchD exposes two disjoint authentication surfaces. The browser dashboard is behind a password plus a TOTP second factor; the `/api/v1` surface this package uses is behind a bearer token. A dashboard session is deliberately *not* accepted as an API token, and vice versa.

Set the token on the server in its `.env` as `API_TOKEN`, and locally as the `RESEARCHR_TOKEN` environment variable (in `~/.Renviron`, so it never reaches your scripts or your history). If the server has no `API_TOKEN` set, the whole surface answers 503 rather than serving unauthenticated — this package reports that as a `researchR_disabled` condition.

A checked connection requires valid API v1 metadata. Malformed responses raise `researchR_protocol_error`. Request errors redact the bearer token and submitted source keys; do not include credentials in diagnostic reports.

Examples

```
conn <- rr_connect()
rr_questions(conn)
```

`rr_delete_question` *Delete a question*

Description

Deleting a question cascades: its concepts, stored records, signals and signal configuration go with it. This cannot be undone.

Usage

```
rr_delete_question(conn, question_id)
```

Arguments

`conn`	A `rr_connect()` connection.
`question_id`	Question id.

Value

Invisibly, `TRUE`.

Examples

```
## Not run:
rr_delete_question(rr_connect(), "obsolete_question")

## End(Not run)
```

`rr_feed_url` *Feed URLs for a question*

Description

The feeds are open — no token — so these URLs can go straight into Zotero (My Library, right-click, New Feed, From URL) or any reader.

Usage

```
rr_feed_url(
  conn,
  question_id,
  study_type = NULL,
  all = FALSE,
  preprints = FALSE,
  signals = FALSE
)
```

Arguments

conn	A rr_connect() connection.
question_id	Question id.
study_type	Optional study type, or signal type when signals = TRUE, for a filtered feed. URL construction does not query server metadata.
all	If TRUE, ignore the question's digest threshold. All feeds still contain at most the server's configured FEED_LIMIT recent items; use rr_records() or rr_signals() for the retained collection.
preprints	If TRUE, the preprints-only literature feed. Cannot be combined with signals = TRUE.
signals	If TRUE, the signals stream rather than the literature one.

Value

A single URL string.

Examples

```
conn <- structure(list(base_url = "https://feeds.example.org"),
                  class = "researchr_conn")
rr_feed_url(conn, "glp1_mash", study_type = "rct")
```

---

rr_health *Is the server up?*

Description

Hits the unauthenticated /healthz endpoint, so this answers "is the web service running" independently of whether the token is any good.

Usage

```
rr_health(conn)
```

Arguments

conn	A `rr_connect()` connection.

Value

`TRUE` if the server answered `ok`, otherwise `FALSE`.

Examples

```
rr_health(rr_connect(check = FALSE))
```

`rr_heartbeat` *Is each service still alive?*

Description

`rr_source_health()` answers "is this *source* still returning anything". This answers the prior question: "did the engine and the scheduler run at all". The distinction matters because a crashed engine writes no source heartbeats either, so without this the two failures look identical — and both look like a quiet week in the literature.

Usage

```
rr_heartbeat(conn)
```

Arguments

conn	A `rr_connect()` connection.

Value

A tibble with one row per service: when it last ran, when it last ran *successfully*, a short detail string, the most recent error, and a `stale` flag.

Examples

```
rr_heartbeat(rr_connect())
```

rr_meta *Server vocabulary and capabilities*

Description

Returns the study-type taxonomy, signal types, confidence tiers and source list the server is actually using. Worth calling after a server upgrade: it is the authoritative vocabulary, so validating against it is how the client avoids drifting from the server's classifier.

Usage

```
rr_meta(conn)
```

Arguments

conn	A rr_connect() connection.

Value

A named list.

Examples

```
rr_meta(rr_connect())$study_types
```

rr_question *Fetch one question by id*

Description

Fetch one question by id

Usage

```
rr_question(conn, question_id)
```

Arguments

conn	A rr_connect() connection.
question_id	Question id (the slug).

Value

A one-row tibble, as in rr_questions().

Examples

```
rr_question(rr_connect(), "glp1_mash")
```

`rr_questions` *List the surveillance questions*

Description

List the surveillance questions

Usage

```
rr_questions(conn)
```

Arguments

`conn`	A `rr_connect()` connection.

Value

A tibble, one row per question. The list-valued fields (`concepts`, `sources`, `email_to`, `study_filter`, `delivery`, `api_key_present`) come back as list columns.

API keys are never returned by the server: `api_key_present` reports only *which* sources have a key stored, never the key itself.

Examples

```
rr_questions(rr_connect())
```

`rr_records` *Fetch stored literature records*

Description

Pulls what researchD has already found and stored — this never triggers a fetch from the upstream databases, which is the pipeline's own job on its cron schedule.

Usage

```
rr_records(
  conn,
  question_id,
  study_type = NULL,
  is_preprint = NULL,
  since = NULL,
  until = NULL,
  min_score = NULL,
  emailed = NULL,
  apply_threshold = FALSE,
  n_max = Inf,
  page_size = 200
)
```

Arguments

conn	A rr_connect() connection.
question_id	Question id.
study_type	Optional single study type. Must be one of rr_meta(conn)$study_types.
is_preprint	Optional logical: preprints only, or non-preprints only.
since, until	Optional publication-date bounds. Date or "YYYY-MM-DD".
min_score	Optional score floor.
emailed	Optional logical: filter on whether the record has already gone out in a digest.
apply_threshold	Apply the question's own digest threshold.
n_max	Maximum records to return.
page_size	Records per request.

Details

Results are paginated by the server; this function follows the pages for you up to n_max. Offset pagination is not a database snapshot. If the result count changes or a record repeats during paging, the request stops with researchR_pagination_changed; retry after collection has finished. Malformed responses or structured metadata raise researchR_protocol_error rather than being silently replaced by empty results.

Value

A tibble of records, one row each, with pub_date as Date and first_seen as UTC POSIXct. provenance is a list column of source contributors; it is empty for servers that do not expose provenance.

Recall versus precision

Scoring orders results without deleting below-threshold records. `apply_threshold = TRUE` reproduces what the curated feed and the digest email show for questions in `digest_mode = "threshold"`; the default `FALSE` gives you the complete stored set. RSS feeds, including `all.xml`, have a server-side item limit. Configured server retention can remove old records.

Examples

```
conn <- rr_connect()
rr_records(conn, "glp1_mash", study_type = "rct", since = Sys.Date() - 90)
```

`rr_set_signal_config` *Update a question's signals configuration*

Description

Update a question's signals configuration

Usage

```
rr_set_signal_config(
  conn,
  question_id,
  sources = NULL,
  min_confidence = NULL,
  instant_alerts = NULL,
  feed_categories = NULL,
  scrape_targets = NULL,
  api_keys = NULL
)
```

Arguments

`conn`	A `rr_connect()` connection.
`question_id`	Question id.
`sources`	Named list of signal-source toggles.
`min_confidence`	Minimum confidence for the signals digest and the default signals feed. One of `"unverified"`, `"reported"`, `"authoritative"`.
`instant_alerts`	If `TRUE`, high-priority signals are emailed on the next scheduler tick rather than waiting for the digest cadence.
`feed_categories`	Named list of curated-feed category toggles.
`scrape_targets`	List of scrape target definitions. Optional list names are ignored.
`api_keys`	Named list of plaintext signal API keys. Write-only, as with `rr_upsert_question()`.

Value

Invisibly, the configuration as the server now holds it.

Examples

```
## Not run:
rr_set_signal_config(rr_connect(), "glp1_mash",
                     min_confidence = "reported", instant_alerts = TRUE)

## End(Not run)
```

`rr_signal_config` *Read a question's signals configuration*

Description

Read a question's signals configuration

Usage

```
rr_signal_config(conn, question_id)
```

Arguments

`conn`	A `rr_connect()` connection.
`question_id`	Question id.

Value

A named list: enabled signal sources, minimum confidence, whether instant alerts are on, enabled feed categories, scrape targets, and which signal API keys are stored (never the keys themselves).

Examples

```
rr_signal_config(rr_connect(), "glp1_mash")
```

`rr_signals` *Fetch stored signals*

Description

The signals track carries non-publication intelligence — grant awards, trial registrations and results postings, regulatory decisions, SEC filings, patents, press and investor-relations items. It runs off the same questions as the literature track but is a deliberately separate stream, because these objects are earlier and less reliable than papers.

Usage

```
rr_signals(
  conn,
  question_id,
  signal_type = NULL,
  min_confidence = NULL,
  since = NULL,
  n_max = Inf,
  page_size = 200
)
```

Arguments

`conn`	A `rr_connect()` connection.
`question_id`	Question id.
`signal_type`	Optional single signal type; see `rr_meta(conn)$signal_types`.
`min_confidence`	Optional minimum confidence tier.
`since`	Optional lower bound on `signal_date`.
`n_max`	Maximum signals to return.
`page_size`	Signals per request.

Value

A tibble of signals. `extra` is parsed from JSON into a list column of type-specific fields. Malformed `extra` JSON raises `researchR_protocol_error`; absent or null metadata becomes an empty list.

Confidence tiers

Every signal carries a confidence: `"authoritative"` (registries, regulators, SEC), `"reported"` (official but preliminary, e.g. a topline readout), or `"unverified"` (scraped press or IR — a lead, not a fact). `min_confidence` is a **floor** over that ordering, not an equality test: `"reported"` returns reported *and* authoritative.

Examples

```
rr_signals(rr_connect(), "glp1_mash", min_confidence = "reported")
```

`rr_source_health` *Per-source fetch health*

Description

A surveillance system's worst failure is a source that quietly stopped returning anything, because the symptom — fewer results — looks exactly like a quiet week in the literature. Every pipeline run writes a heartbeat row per source; this reports the last successful fetch, the last one that actually returned rows, and the error count.

Usage

```
rr_source_health(conn, stale_days = 3)
```

Arguments

`conn`	A `rr_connect()` connection.
`stale_days`	A source with no successful fetch in this many days is flagged `stale`.

Details

Watch `last_nonempty` as much as `stale`: a source can be answering happily while its query has silently stopped matching anything.

Value

A tibble, one row per source and track.

Examples

```
rr_source_health(rr_connect())
```

`rr_upsert_question` *Create or update a question*

Description

Upserts by `question_id`: an existing question with that id is updated in place, otherwise a new one is created. Concepts are replaced wholesale, so pass the full set you want.

Usage

```
rr_upsert_question(
  conn,
  question_id,
  label,
  concepts,
  sources = list(),
  api_keys = list(),
  study_filter = character(0),
  from_days_back = 7L,
  enabled = TRUE,
  digest_mode = c("all", "threshold"),
  min_score = 0,
  delivery = "file",
  email_enabled = FALSE,
  email_cron = "0 7 * * 1",
  email_to = character(0)
)
```

Arguments

`conn`	A `rr_connect()` connection.
`question_id`	Question id (a slug, e.g. `"glp1_mash"`): starts with a letter or digit, followed by letters, digits, underscores or hyphens; at most 128 characters.
`label`	Human-readable label.
`concepts`	List of `rr_concept()` blocks. Optional list names are ignored.
`sources`	Named list of per-source toggles, e.g. `list(pubmed = TRUE, biorxiv = FALSE)`. Sources you do not name fall back to the server's global defaults.
`api_keys`	Named list of plaintext per-source API keys. Write-only.
`study_filter`	Character vector of publication types to restrict to. Empty means maximum recall.
`from_days_back`	Lookback window, in days.
`enabled`	Whether the question is fetched at all.

digest_mode	"all" or "threshold". Under "threshold" the curated feed and the email show only records scoring at least min_score. Below-threshold records remain queryable through the API; all.xml also omits the threshold but keeps the server's feed item limit.
min_score	Score cutoff used when digest_mode = "threshold".
delivery	Character vector of delivery channels, e.g. c("file", "zotero").
email_enabled	Whether this question emails a digest.
email_cron	Cron expression for this question's own email cadence.
email_to	Character vector of recipient addresses.

Value

Invisibly, the question as the server now holds it.

API keys

api_keys takes *plaintext* keys and is write-only: the server encrypts them at rest and never returns them. Leave a source out to keep its stored key untouched; pass the literal "__CLEAR__" to delete one.

Examples

```
## Not run:
rr_upsert_question(
  rr_connect(), "glp1_mash", "GLP-1 receptor agonists in MASH",
  concepts = list(
    rr_concept("drug", c("semaglutide", "tirzepatide", "GLP-1")),
    rr_concept("disease", c("MASH", "NASH", "steatohepatitis"))),
  sources = list(pubmed = TRUE, europepmc = TRUE),
  email_enabled = TRUE, email_cron = "0 7 * * 1",
  email_to = "me@example.org")

## End(Not run)
```

Index

rr_concept, 2
rr_concept(), *14*
rr_connect, 3
rr_connect(), *4–14*
rr_delete_question, 4
rr_feed_url, 4
rr_health, 5
rr_heartbeat, 6
rr_meta, 7
rr_question, 7
rr_questions, 8
rr_questions(), *7*
rr_records, 8
rr_records(), *5*
rr_set_signal_config, 10
rr_signal_config, 11
rr_signals, 12
rr_signals(), *5*
rr_source_health, 13
rr_upsert_question, 14
rr_upsert_question(), *10*