Package: robotstxtr (via r-universe)

October 24, 2026

Type Package

Title Engine-Aware 'robots.txt' Policy with Google's Matcher Algorithm

Version 0.3.1

Language en-US

Description Evaluates whether URLs may be crawled according to 'robots.txt' rules. A versioned engine-aware facade keeps neutral fetch evidence, per-engine status policy, robots product tokens, and matcher capabilities separate, returning explicit non-decisions when policy is undocumented or a matcher is unavailable. The available matcher is a bundled, upstream-test-validated snapshot of Google's open-source C++ 'robots.txt' matcher, preserving 'Googlebot' behavior. Fetches use a deterministic, conservative policy built on 'httr2', with fetch origins constructed by 'rurl'. All network access in tests, examples, and vignettes is mocked or served locally, so no internet connection is required to install, test, or check the package. This matcher derives from Google's 'robotstxt' project <https://github.com/google/robotstxt>, distributed under the Apache License 2.0; see the package NOTICE.

License MIT + file LICENSE

Copyright file inst/COPYRIGHTS

URL https://bart-turczynski.gitlab.io/robotstxtr/, https://gitlab.com/bart-turczynski/robotstxtr, https://bart-turczynski.r-universe.dev/robotstxtr

BugReports https://gitlab.com/bart-turczynski/robotstxtr/-/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports httr2 (>= 1.2.0), rurl (>= 3.0.1)

LinkingTo cpp11

SystemRequirements C++17

Suggests callr, httpuv, jsonlite, knitr, oysteR, rmarkdown, rosv, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Bart Turczynski [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-8788-7980>), Google LLC [cph] (Bundled 'robots.txt' matcher: src/robots.cc, src/robots.h, src/reporting_robots.cc, src/reporting_robots.h)

Maintainer Bart Turczynski <bartek@turczynski.pl>

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

Date/Publication 2026-10-24 16:10:08 UTC

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

RemoteRef HEAD

RemoteSha 98da8765bfe1de613797c53f416199f703c0b882

Contents

allowed_by_robots_text . . . 3
allowed_by_robots_url . . . 4
as_legacy_robots_decisions_v1 . . . 6
print.robots_decisions . . . 7
print.robots_fetches . . . 8
print.robots_validations . . . 9
robots_body . . . 9
robots_engine_contract_v1 . . . 11
robots_engine_contract_v2 . . . 12
robots_evaluate_text_v1 . . . 13
robots_evaluate_url_v1 . . . 14
robots_fetch . . . 16
robots_resolve_matcher_profile_v1 . . . 19
robots_validate_text . . . 20
robots_validate_url . . . 21

Index **24**

`allowed_by_robots_text` *Match URLs against a supplied robots.txt body*

Description

Matches one supplied robots.txt body against one or more URLs using the vendored, upstream-test-validated Google robots.txt matcher. No HTTP request is performed; the caller supplies the robots.txt text directly.

Usage

```
allowed_by_robots_text(robots_txt, url, user_agent, source_id = "supplied")
```

Arguments

`robots_txt`	A single, non-missing character scalar holding the robots.txt body. An empty body is valid. It is converted to UTF-8 once and the exact bytes are stored in the returned `robots$body` and passed to the matcher; a body already carrying `Encoding = "bytes"` is used verbatim.
`url`	A character vector of URL strings to match, passed to the matcher unchanged. May have length zero.
`user_agent`	A matcher user-agent string, length one or `length(url)`, passed to the matcher unchanged (upstream extraction semantics apply, so e.g. `"Googlebot/2.1"` is not pre-trimmed by R).
`source_id`	A single non-empty character scalar identifying the supplied body within the returned object. Defaults to `"supplied"`.

Details

The URL string is passed to the matcher unchanged. The upstream matcher requires an appropriately encoded, %-escaped full URL (per RFC 3986); callers are responsible for supplying it in that form. R does not clean, decode, canonicalize, or reserialize the URL before matching.

`url` may have length zero (an empty result is returned). `user_agent` must be a character vector of length one or `length(url)`: a scalar user agent expands across every URL, and no other recycling is allowed. A wrong R type for any argument, a `user_agent` length mismatch, a non-scalar or missing `robots_txt`, or an empty/missing `source_id` are call-level errors.

A missing (`NA`) or empty (`""`) URL or user-agent element does not abort the call: that row yields a per-row `input_unknown` decision with `allowed = NA`. Such an invalid row is detached from the supplied body: its `source_id`, `robots_url`, and `http_status` are `NA` and its `fetch_outcome` is `"input_invalid"`. When both the URL and the user agent of a row are invalid, the URL error is reported (URL validity is checked first).

Match metadata is fully populated on the text path. `matched_line` carries the upstream one-based matching line (or `NA` when no rule matched). For a matched row, `matched_rule_type` and `matched_rule_value` are correlated from the parse callbacks: the type (`"allow"`/`"disallow"`) and

the canonical rule value the matcher actually used, i.e. the value after upstream `MaybeEscapePattern` canonicalization (non-ASCII bytes percent-escaped, existing `%xx` escapes upper-cased). R does not reconstruct a pre-escape form. A non-UTF-8 callback value is returned with `Encoding = "bytes"`. Rows where no rule matched keep `matched_rule_value = NA` and a decision-derived `matched_rule_type` (`"none"` for `default_allow`, `"unknown"` for `input_unknown`).

Value

An S3 object of class `robots_decisions`: a named list with two components, `results` (one row per input URL, in input order) and `robots` (one row for the supplied source body). The supplied body is stored once as a raw vector in `robots$body`.

Examples

```
allowed_by_robots_text("user-agent: *\ndisallow: /x", "http://a/x", "bot")
```

---

allowed_by_robots_url *Fetch robots.txt and match URLs under the deterministic fetch policy*

Description

Fetches the governing `/robots.txt` for one or more target URLs and matches each URL against the fetched body using the vendored, upstream-test-validated Google robots.txt matcher. This is the URL-first headline path: it combines the grouped fetch stage (`robots_fetch()`) with matching and rich per-row decisions, returning a `robots_decisions` object.

Usage

```
allowed_by_robots_url(
  url,
  user_agent,
  timeout = 10,
  max_bytes = 524288L,
  fetch_user_agent = NULL,
  ssrf_guard = TRUE
)
```

Arguments

`url`	A character vector of target URLs. May have length zero (an empty result is returned). The original string is passed to the matcher unchanged. A wrong R type is a call-level error.
`user_agent`	A matcher user-agent string, length one or `length(url)`, passed to the matcher unchanged (upstream extraction semantics apply, so e.g. `"Googlebot/2.1"` is not pre-trimmed by R).

`timeout` A single positive, finite numeric total timeout in seconds per origin, including redirects. Defaults to `10`.

`max_bytes` A single positive, finite, whole-number decoded-body byte limit no greater than `.Machine$integer.max`, coerced to integer. A whole-number double (for example `524288`) is accepted; fractional, non-positive, out-of-range, or non-scalar values are call-level errors. Defaults to `524288L`.

`fetch_user_agent`
`NULL` (use the package fetch user agent) or a single non-empty character HTTP user agent to send instead. This is never the matcher `user_agent`.

`ssrf_guard` A single logical. When `TRUE` (the default) a structural SSRF guard refuses, before any request, both the initial origin and every redirect target that resolves to a private, loopback, link-local, or cloud-metadata address (outcome `ssrf_blocked`, no socket opened). Set to `FALSE` to deliberately opt out — for example to fetch `robots.txt` from a trusted intranet host. Opting out removes that protection; only disable it for hosts you control and trust.

Details

Each URL's fetch-origin grouping key `scheme://host[:port]/robots.txt` is built from the URL alone. The URL must begin with an explicit `http://` or `https://` (in any letter case); a scheme-relative (`//host/...`) or scheme-less URL is invalid, never defaulted to HTTP. The scheme and host are put in lower case, an IDN host is converted to its ASCII form, a default port (80 for `http`, 443 for `https`) is dropped while any other port is kept, an IPv6 literal stays bracketed, userinfo is dropped, and the path, query, and fragment are replaced by `/robots.txt`. Within one call each distinct grouping key is fetched exactly once and every fetch-eligible row sharing that key reuses the same source body; source IDs are assigned `robots_1`, `robots_2`, and so on in first-occurrence order. There is no persistent, cross-call, or hidden HTTP cache.

The original input URL string is passed to the matcher unchanged. R does not clean, decode, canonicalize, or reserialize it: percent escapes, Unicode, path case, query strings, and fragments all reach the matcher as supplied. `rurl` is used only to construct the fetch-origin grouping key, never to transform the matcher input. The upstream matcher requires an appropriately encoded, `%`-escaped full URL (per RFC 3986); callers are responsible for supplying it in that form.

`url` may have length zero (an empty result is returned). `user_agent` must be a character vector of length one or `length(url)`: a scalar user agent expands across every URL, and no other recycling is allowed. A wrong R type, a `user_agent` length mismatch, or an invalid `timeout`, `max_bytes`, or `fetch_user_agent` are call-level errors.

A missing (`NA`), empty (`""`), malformed, or non-HTTP(S) URL, or a missing or empty user-agent element, does not abort the call: that row yields a per-row `input_unknown` decision with `allowed = NA`. URL validity is checked before user-agent validity, so each invalid row has one deterministic primary error class. An invalid row is fully detached: it never causes an HTTP request and its `source_id`, `robots_url`, and `http_status` are `NA` with `fetch_outcome = "input_invalid"`. A row with a valid URL but invalid user agent stays detached even when a valid sibling row causes the same origin to be fetched.

Per-row decisions follow the fetch outcome: a `200`-class body is matched (`rule_disallow`/`rule_allow`/`default_allow`; an empty body is valid and yields `default_allow`); a `404`/`410` yields `missing_allow`/`TRUE`; any fetch failure yields `fetch_unknown`/`NA`. For matched rows, `matched_line`, `matched_rule_type`,

and matched_rule_value are correlated from the parse callbacks of the same source body (the canonical value the matcher used, after upstream MaybeEscapePattern canonicalization).

Value

An S3 object of class robots_decisions: a named list with two components, results (one row per input URL, in input order) and robots (one row per fetched source body). Each fetched body is stored once as a raw vector in robots$body. Large body values never appear in results.

See Also

allowed_by_robots_text() to match a supplied body without HTTP, and robots_fetch() for the fetch stage on its own.

Examples

```
# allowed_by_robots_url() fetches /robots.txt over HTTP and matches in one
# step. The transport is mocked here so the example runs offline; a real call
# needs no such wrapper.
httr2::with_mocked_responses(
  function(req) {
    httr2::response(
      status_code = 200L, url = req$url,
      body = charToRaw("user-agent: *\nDisallow: /private\n")
    )
  },
  allowed_by_robots_url(
    c("https://example.com/page", "https://example.com/private"),
    "my-bot"
  )
)
```

---

as_legacy_robots_decisions_v1

Convert a v1 Google engine result to the legacy decision schema

---

Description

Explicit compatibility adapter for callers that still consume the original robots_decisions schema. It accepts only results whose ruleset and matcher backend are both "google"; unsupported legacy fetch outcomes remain unknown exactly as in the original API.

Usage

```
as_legacy_robots_decisions_v1(x)
```

Arguments

x	A robots_engine_decisions_v1 object using the Google policy and Google matcher backend for every row.

Value

A legacy robots_decisions object.

Examples

```
# Build a v1 engine result offline, then adapt it for a legacy consumer.
engine <- robots_evaluate_text_v1(
  "user-agent: *\nDisallow: /private\n",
  c("https://example.com/page", "https://example.com/private"),
  robots_product_token = "my-bot",
  robots_policy_ruleset = "google",
  matcher_backend = "google"
)
engine$results[c("url", "url_decision", "reason")]

# The adapter collapses the three-valued `url_decision` into the legacy
# logical `allowed`, and carries `reason` over as `decision_source`.
legacy <- as_legacy_robots_decisions_v1(engine)
class(legacy)
legacy$results[c("url", "allowed", "decision_source", "matched_rule_type")]
```

---

print.robots_decisions

Print a robots_decisions *object*

Description

Prints a compact one-line header (result and source counts), an allow/disallow/unknown tally, and a preview of up to ten result rows. Large body values are never shown. Returns the object invisibly.

Usage

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

Arguments

x	A robots_decisions object.
...	Ignored; present for S3 compatibility.

Value

x, invisibly.

Examples

```
# allowed_by_robots_text() matches a supplied body with no network access.
decisions <- allowed_by_robots_text(
  "user-agent: *\nDisallow: /private\n",
  c("https://example.com/page", "https://example.com/private"),
  "my-bot"
)
print(decisions)
```

---

print.robots_fetches *Print a* robots_fetches *object*

Description

Prints a compact one-line header (input and source counts) and a preview of up to ten input-map rows. Body values are never shown. Returns the object invisibly.

Usage

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

Arguments

x	A robots_fetches object.
...	Ignored; present for S3 compatibility.

Value

x, invisibly.

Examples

```
# A robots_fetches object comes from robots_fetch(); the transport is mocked
# here so the example runs offline.
fetches <- httr2::with_mocked_responses(
  function(req) {
    httr2::response(
      status_code = 200L, url = req$url,
      body = charToRaw("user-agent: *\nDisallow: /private\n")
    )
  },
  robots_fetch("https://example.com/page")
)
print(fetches)
```

print.robots_validations *Print a robots.txt validation result*

Description

Print a robots.txt validation result

Usage

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

Arguments

x	A robots_validations object.
...	Additional arguments, currently ignored.

Value

x, invisibly.

Examples

```
# A robots_validations object comes from robots_validate_url() or
# robots_validate_text(); the transport is mocked here so the example runs
# offline.
validations <- httr2::with_mocked_responses(
  function(req) {
    httr2::response(
      status_code = 200L, url = req$url,
      body = charToRaw("user-agent: *\nDisallow: /private\n")
    )
  },
  robots_validate_url("https://example.com/private")
)
print(validations)
```

robots_body *Preview or extract a stored robots.txt body*

Description

Returns the robots.txt body stored in a robots_decisions object. Each source body is stored once as a raw vector in the object's robots table; this helper selects one source and either renders a character preview (raw = FALSE, the default) or returns the stored raw vector byte-for-byte unchanged (raw = TRUE).

Usage

```
robots_body(x, source_id = NULL, n = 20, raw = FALSE)
```

Arguments

x	A robots_decisions object.
source_id	A single non-empty character string naming the source to read from x$robots$source_id, or NULL (the default) to select the only source when the object holds exactly one.
n	A single positive number giving the maximum number of body bytes to include in the preview. Defaults to 20. Use n = Inf for the full body. Ignored when raw = TRUE.
raw	A single logical. When TRUE, return the stored raw vector unchanged (byte-for-byte identical to x$robots$body[[i]]) instead of a rendered preview. Defaults to FALSE.

Details

Selection follows the robots$source_id column. If source_id is NULL (the default) and the object holds exactly one source, that source is used. If source_id is NULL and the object holds more than one source, the call is ambiguous and raises a package error rather than choosing silently. An unknown source_id is also a package error.

When raw = FALSE, the selected body's bytes are rendered to a character string. Robots.txt bodies may be arbitrary bytes (for example non-UTF-8 or a value carrying Encoding = "bytes"); rendering never errors on invalid encoding. A body whose bytes form valid UTF-8 is returned as a UTF-8 string; otherwise the result carries Encoding = "bytes" so R does not attempt to reinterpret it. n limits the preview to the first n bytes of the body; n = Inf returns the full body. n is ignored when raw = TRUE, which always returns the complete stored raw vector unchanged.

Value

When raw = FALSE, a length-one character string (a preview, or the full body when n = Inf), marked Encoding = "UTF-8" or Encoding = "bytes". When raw = TRUE, the stored raw vector unchanged (or NULL when the selected source stored no body).

Examples

```
x <- allowed_by_robots_text("user-agent: *\ndisallow: /x", "http://x/x", "b")
robots_body(x)
robots_body(x, n = Inf)
robots_body(x, raw = TRUE)
```

robots_engine_contract_v1 *Inspect the versioned engine-aware robots contract*

Description

Returns the stable identifiers, value sets, backend capability states, the matcher token/semantics capability boundary (`matcher_capability`, which for a `bounded_profiles` backend also publishes the accepted matcher-profile selectors in `supported_profiles`), the separately inspectable vendored-matcher identity fields (`matcher_identity`, e.g. library/payload/profile/corpus/evidence/profile-source for the Yandex backend), status-policy table, and supported sibling-package ranges for the v1 engine-aware contract. It performs no fetch or matching.

Usage

```
robots_engine_contract_v1()
```

Value

A named list of contract metadata with class `robots_engine_contract_v1`.

Examples

```
# A plain offline accessor: it opens no socket and matches nothing.
contract <- robots_engine_contract_v1()
contract$contract_id
contract$schema_revision

# Which matcher backends this installation can actually run.
contract$matcher_availability

# The token/semantics boundary each backend is authoritative for.
contract$matcher_capability$yandex$token_policy

# Which selectors a bounded backend accepts, and how each one selects a
# group. `Yandex` falls back to `User-agent: *`; `YandexAdditionalBot`
# never does, so the two are not interchangeable.
contract$matcher_capability$yandex$supported_profiles

# Status policy is a lookup keyed by acquisition category and ruleset:
# how each ruleset treats a 4xx response to the robots.txt request.
policy <- contract$policy_table
policy[
  policy$category == "status_4xx",
  c("ruleset", "policy_status", "policy_action", "policy_provenance")
]
```

robots_engine_contract_v2 *Inspect the engine-aware/v2 robots contract (Bing activation)*

Description

Returns the v2 facade metadata introduced by the atomic Bing matcher activation. It has the same shape as robots_engine_contract_v1() but publishes the v2 contract id (robotstxtr.engine-aware/v2), the v2 schema revision, the full eight-member v2 matcher-status set (matcher_status_set), and the separately inspectable vendored-matcher identity fields for both the Yandex and Bing backends (matcher_identity$bing, with library/payload/contract/parser/profile/ manifest components). It reads the same shared matcher registry as the v1 accessor, so the Bing backend reports available here and there alike; the v1 accessor's own contract_id and schema_revision stay unchanged. It does no fetch or matching.

Usage

```
robots_engine_contract_v2()
```

Value

A named list of contract metadata with class robots_engine_contract_v2.

See Also

robots_engine_contract_v1()

Examples

```
v2 <- robots_engine_contract_v2()
v1 <- robots_engine_contract_v1()

# v2 publishes its own identifiers; the v1 accessor keeps its own.
c(v1 = v1$contract_id, v2 = v2$contract_id)
c(v1 = v1$schema_revision, v2 = v2$schema_revision)

# The component v2 adds is the full eight-member matcher-status set.
setdiff(names(v2), names(v1))
v2$matcher_status_set

# Everything else is read from the same shared matcher registry, so the
# Bing identity and backend availability agree across both accessors.
v2$matcher_identity$bing$contract_id
identical(v2$matcher_availability, v1$matcher_availability)
```

`robots_evaluate_text_v1` *Evaluate URLs against supplied robots.txt under an explicit engine profile*

Description

Versioned, no-network entry point for the engine-aware v1 contract. The robots status-policy ruleset and matcher backend are selected independently; an unavailable backend produces `matcher_status = "capability_unavailable"` rather than silently using Google's matcher.

Usage

```
robots_evaluate_text_v1(
  robots_txt,
  url,
  robots_product_token,
  robots_policy_ruleset,
  matcher_backend,
  source_id = "supplied"
)
```

Arguments

`robots_txt`	A single, non-missing character value containing the robots.txt body.
`url`	A character vector of URLs to evaluate.
`robots_product_token`	A matcher-profile selector: a robots.txt `User-agent:` group label, NOT an HTTP `User-Agent` crawler identity. It is never sent as a header. A character vector of length one or `length(url)`. The Google backend accepts any valid Google robots product token and yields Google semantics for it, not a prediction of the named crawler. The `"yandex"` and `"bing"` backends accept only a closed set of selectors, published on the engine contract as `matcher_capability$<backend>$supported_prof` and checkable up front with `robots_resolve_matcher_profile_v1()`; an unaccepted selector leaves its rows undecided rather than raising.
`robots_policy_ruleset`	An explicit ruleset, length one or `length(url)`: `"google"`, `"yandex"`, `"rfc9309"`, `"bing"`, or `"assumed_rfc9309"`.
`matcher_backend`	An explicit matcher backend, length one or `length(url)`: `"google"`, `"yandex"`, `"rfc9309"`, or `"bing"`.
`source_id`	A non-empty identifier for the supplied body.

Details

The robots_product_token is used ONLY for robots user-agent group selection; it is never sent as an HTTP header. Each matcher backend is authoritative only for its own vendor semantics: google for Google, yandex for Yandex, bing for Bing, and RFC 9309 behavior only under the rfc9309 backend. The Google backend additionally accepts any valid Google robots product token and returns Google parsing/matching semantics for it; that is NOT a claim of compatibility with, or a prediction of, the crawler the token names. The Yandex and Bing backends are bounded to their supported vendor profiles only and never generalize to arbitrary tokens. This boundary is published as robots_engine_contract_v1()$matcher_capability.

The Yandex backend is active (since schema revision 2026-07-18.2). An evaluated Yandex row publishes reason as one of default_allow, rule_allow, rule_disallow, or effective_empty_disallow; a non-evaluated Yandex row reports matcher_status = "not_evaluated" with reason = "unsupported_product_token" (error_class = "robots_unsupported_product_token") or reason = "invalid_request_target" (error_class = "robots_invalid_request_target") at error_stage = "input". The activation adds the public matched_rule_value_raw list column (one element per row: exact owning-rule bytes, raw(0) for an effective-empty Disallow, or NULL for an absent rule); it is populated for Yandex-evaluated rows and NULL elsewhere. The vendored-matcher identity is published as robots_engine_contract_v1()$matcher_identity$yandex.

Value

A robots_engine_decisions_v1 object with results, neutral evidence, and contract components.

Examples

```
# A non-Google-named token evaluated through the default Google backend
# yields Google parsing/matching semantics for that token. This is NOT a
# prediction of the named crawler's behavior.
decisions <- robots_evaluate_text_v1(
  "user-agent: *\ndisallow: /private",
  "https://example.com/private",
  robots_product_token = "Yandex",
  robots_policy_ruleset = "google",
  matcher_backend = "google"
)
decisions$results$matcher_backend  # "google" -- Google semantics, not Yandex
decisions$results$url_decision     # "disallow"
```

---

robots_evaluate_url_v1

Fetch and evaluate robots.txt under an explicit engine profile

---

Description

Versioned URL-first entry point for the engine-aware v1 contract. It records neutral acquisition and safety evidence before applying the explicitly selected policy ruleset and matcher backend. fetch_user_agent is the HTTP request User-Agent and is never used as robots_product_token.

Usage

```
robots_evaluate_url_v1(
  url,
  robots_product_token,
  robots_policy_ruleset,
  matcher_backend,
  timeout = 10,
  max_bytes = 524288L,
  fetch_user_agent = NULL,
  ssrf_guard = TRUE
)
```

Arguments

`url`	A character vector of URLs to evaluate.
`robots_product_token`	A matcher-profile selector: a robots.txt `User-agent:` group label, NOT an HTTP `User-Agent` crawler identity. It is never sent as a header. A character vector of length one or `length(url)`. The Google backend accepts any valid Google robots product token and yields Google semantics for it, not a prediction of the named crawler. The `"yandex"` and `"bing"` backends accept only a closed set of selectors, published on the engine contract as `matcher_capability$<backend>$supported_prof` and checkable up front with `robots_resolve_matcher_profile_v1()`; an unaccepted selector leaves its rows undecided rather than raising.
`robots_policy_ruleset`	An explicit ruleset, length one or `length(url)`: `"google"`, `"yandex"`, `"rfc9309"`, `"bing"`, or `"assumed_rfc9309"`.
`matcher_backend`	An explicit matcher backend, length one or `length(url)`: `"google"`, `"yandex"`, `"rfc9309"`, or `"bing"`.
`timeout`	A single positive, finite numeric total timeout in seconds per origin, including redirects. Defaults to `10`.
`max_bytes`	A single positive, finite, whole-number decoded-body byte limit no greater than `.Machine$integer.max`, coerced to integer and recorded on each source row. A whole-number double (for example `524288`) is accepted; fractional, non-positive, out-of-range, or non-scalar values are call-level errors. Defaults to `524288L`.
`fetch_user_agent`	`NULL` (use the package fetch user agent) or a single non-empty character HTTP user agent to send instead.
`ssrf_guard`	A single logical. When `TRUE` (the default) a structural SSRF guard refuses, before any request, both the initial origin and every redirect target that resolves to a private, loopback, link-local, or cloud-metadata address (outcome `ssrf_blocked`, no socket opened). Set to `FALSE` to deliberately opt out — for example to fetch `robots.txt` from a trusted intranet host. Opting out removes that protection; only disable it for hosts you control and trust.

Details

As with robots_evaluate_text_v1(), the robots_product_token drives ONLY user-agent group selection. Each matcher backend is authoritative only for its own vendor semantics; the Google backend accepts any valid Google robots product token and returns Google semantics for it, which is not a claim of compatibility with the named crawler, while the Yandex and Bing backends are bounded to their supported vendor profiles and RFC 9309 behavior is available only under the rfc9309 backend. See robots_engine_contract_v1()$matcher_capability.

Value

A robots_engine_decisions_v1 object with results, neutral evidence, and contract components.

Examples

```
# This entry point fetches /robots.txt over HTTP. The transport is mocked
# here so the example runs offline; a real call needs no such wrapper.
decisions <- httr2::with_mocked_responses(
  function(req) {
    httr2::response(
      status_code = 200L, url = req$url,
      body = charToRaw("user-agent: *\nDisallow: /private\n")
    )
  },
  robots_evaluate_url_v1(
    c("https://example.com/page", "https://example.com/private"),
    robots_product_token = "my-bot",
    robots_policy_ruleset = "google",
    matcher_backend = "google"
  )
)
decisions$results[c("url", "policy_status", "url_decision", "reason")]

# Acquisition evidence is recorded neutrally, apart from the decision, so
# the policy applied to the response stays auditable.
decisions$evidence[c("source_id", "final_http_status", "evidence_status")]
```

---

robots_fetch *Fetch robots.txt documents for URLs under the deterministic fetch policy*

Description

Performs the grouped robots.txt fetch stage for one or more target URLs and returns a robots_fetches object. No matching is performed (that is the allowed_by_robots_url() path, a later slice); this function only computes each URL's fetch-origin, fetches each distinct /robots.txt once, and classifies the HTTP outcome.

Usage

```
robots_fetch(
  url,
  timeout = 10,
  max_bytes = 524288L,
  fetch_user_agent = NULL,
  ssrf_guard = TRUE
)
```

Arguments

`url`	A character vector of target URLs. May have length zero (an empty result is returned). A wrong R type is a call-level error.
`timeout`	A single positive, finite numeric total timeout in seconds per origin, including redirects. Defaults to `10`.
`max_bytes`	A single positive, finite, whole-number decoded-body byte limit no greater than `.Machine$integer.max`, coerced to integer and recorded on each source row. A whole-number double (for example `524288`) is accepted; fractional, non-positive, out-of-range, or non-scalar values are call-level errors. Defaults to `524288L`.
`fetch_user_agent`	`NULL` (use the package fetch user agent) or a single non-empty character HTTP user agent to send instead.
`ssrf_guard`	A single logical. When `TRUE` (the default) a structural SSRF guard refuses, before any request, both the initial origin and every redirect target that resolves to a private, loopback, link-local, or cloud-metadata address (outcome `ssrf_blocked`, no socket opened). Set to `FALSE` to deliberately opt out — for example to fetch `robots.txt` from a trusted intranet host. Opting out removes that protection; only disable it for hosts you control and trust.

Details

For every input URL the fetch-origin grouping key `scheme://host[:port]/robots.txt` is built from the URL alone. The URL must begin with an explicit `http://` or `https://` (in any letter case); a scheme-relative (`//host/...`) or scheme-less URL is invalid, never defaulted to HTTP. The scheme and host are put in lower case, an IDN host is converted to its ASCII form, a default port (80 for `http`, 443 for `https`) is dropped while any other port is kept, an IPv6 literal stays bracketed, userinfo is dropped, and the path, query, and fragment are replaced by `/robots.txt`. Within one call each distinct grouping key is fetched exactly once and every input row sharing that key references the same source row and stored body. Grouping is keyed on the requested robots URL, not on the final redirect destination. Input order is preserved in `map`, and fetch groups run sequentially in first-occurrence order; source IDs are assigned `robots_1`, `robots_2`, and so on in that order. There is no persistent, cross-call, or hidden HTTP cache.

A URL whose fetch-origin cannot be constructed (missing, empty, malformed, or non-HTTP(S)) is invalid: it never triggers an HTTP request and appears in `map` with `fetch_outcome = "input_invalid"`, no `source_id`, and invalid-URL error metadata.

The request policy is deterministic and intentionally conservative: HTTP `GET`; TLS verified with platform defaults; a total timeout per origin including redirects; at most five redirects; redirects allowed only to `http`/`https`; HTTP-to-HTTPS upgrades allowed but every HTTPS-to-HTTP downgrade rejected as `redirect_error`; a structural SSRF guard that refuses, before any request, both the initial origin and every redirect target that resolves to a private, loopback, link-local, or cloud-metadata address (outcome `ssrf_blocked`, no socket opened); no URL userinfo, authorization, cookies, or caller session state forwarded; a package fetch user agent (`robotstxtr/<version>`) unless the caller supplies one; and no automatic retries. Each response is classified into a stable `fetch_outcome`. For a `fetched` outcome the exact decoded entity bytes are stored once as a raw vector in the source row's `body` (an empty body is valid); all other outcomes store no body.

`max_bytes` caps the decoded response body. The limit is enforced on decoded entity bytes (after any HTTP content decoding, such as gzip) by a streaming read that stops the moment the decoded byte count exceeds the limit; the compressed entity is never fully downloaded before the check. A body over the limit ends as `body_too_large` with no stored body (never a truncated body); a body at or under the limit is stored normally as a `fetched` result.

Value

An S3 object of class `robots_fetches`: a named list with two components, `map` (one row per input URL, in input order) and `robots` (one row per fetched source, with the fetched body stored once as a raw vector in `robots$body`).

See Also

`allowed_by_robots_url()` to fetch and match in one step.

Examples

```
# robots_fetch() performs live HTTP. To keep this example offline and
# deterministic, the transport is mocked with httr2 so every request returns
# the same robots.txt; a real call needs no such wrapper.
httr2::with_mocked_responses(
  function(req) {
    httr2::response(
      status_code = 200L, url = req$url,
      body = charToRaw("user-agent: *\nDisallow: /private\n")
    )
  },
  {
    # Both URLs share one fetch-origin, so /robots.txt is fetched once.
    robots_fetch(c("https://example.com/", "https://example.com/private"))
  }
)
```

`robots_resolve_matcher_profile_v1` *Resolve a matcher-profile selector before evaluating anything*

Description

Checks, at configuration-construction time, that a product token is one the named matcher backend will actually accept — and reports what it will DO, not merely that it is accepted. Performs no fetch and no matching.

Usage

```
robots_resolve_matcher_profile_v1(matcher_backend, robots_product_token)
```

Arguments

`matcher_backend`
: A character vector of matcher backends, length one or `length(robots_product_token)`: `"google"`, `"yandex"`, `"rfc9309"`, or `"bing"`.

`robots_product_token`
: A character vector of selectors, length one or `length(matcher_backend)`.

Details

A `bounded_profiles` backend (`"yandex"`, `"bing"`) accepts a closed set of selectors. Passing an unaccepted one is not an error during evaluation: the affected rows come back undecided, with `url_decision` `NA`, which reads like a transport failure rather than a configuration mistake. This function turns that quiet outcome into a loud one at the point where the mistake is actually made.

Checking the evaluation results instead is not equivalent. Profile support is a property of the matcher, and a row need not reach the matcher at all — an `allow_all` policy returns a decision for a nonsensical selector without ever consulting the profile.

The `robots_product_token` is a robots.txt `User-agent:` group selector, not an HTTP `User-Agent` crawler identity. The distinction is easy to miss because RFC 9309 treats them as one namespace, and vendors do not: Yandex's `"Yandex"` is an umbrella group label that no crawler ever sends, while `"YandexBot"` — a real crawler — is not an accepted selector at all.

Value

A data frame with one row per resolved selector and the columns `matcher_backend`, `robots_product_token` (the value as supplied), `token_policy`, `profile_id`, `accepted_token` (the canonical spelling), `group_label`, `group_selection`, and `profile_revision`. For a backend whose `token_policy` is not `"bounded_profiles"` there is no closed set to resolve against, so the profile columns are `NA` and the token is returned unchanged.

`group_selection` is `"exact_else_wildcard"` when the profile falls back to `User-agent: *` groups if no group names it, and `"exact_only"` when it never does. Two accepted selectors on one backend are therefore not interchangeable.

See Also

robots_engine_contract_v1(), whose matcher_capability publishes the same data as a table.

Examples

```
# The umbrella group label and the Additional crawler are both accepted, but
# they select groups differently.
robots_resolve_matcher_profile_v1(
  "yandex", c("Yandex", "YandexAdditionalBot")
)

# Selectors are compared ASCII case-insensitively; the canonical spelling
# comes back in `accepted_token`.
robots_resolve_matcher_profile_v1("bing", "BINGBOT")

# A real crawler identity is not a selector. This is the mistake the
# function exists to catch, so it errors rather than returning a row.
try(robots_resolve_matcher_profile_v1("yandex", "YandexBot"))

# An unbounded backend has no closed set to resolve against.
robots_resolve_matcher_profile_v1("google", "Googlebot")
```

---

robots_validate_text *Validate a supplied robots.txt document*

Description

Parses a supplied document with the package's pinned Google-compatible parser and returns byte evidence, directive counts, and stable per-line diagnostics. This validates document syntax and structure; it does not decide whether a URL is allowed, infer webmaster intent, or fetch sitemap resources declared by the document.

Usage

```
robots_validate_text(robots_txt, source_id = "supplied")
```

Arguments

robots_txt	A single non-missing character string or a raw vector.
source_id	A single non-empty character string identifying the supplied document. Defaults to "supplied".

Details

The validation profile is explicitly "google-parser-compatible". It reports what the vendored parser recognizes (including the parser's accepted typo and missing-colon variants) and adds conservative structural checks. It is not a claim of universal crawler behavior or complete RFC 9309 validity.

robots_txt may be text or a raw vector. Raw input is useful when validating acquisition bytes that contain malformed UTF-8, a byte-order mark, or NUL bytes. Character input marked with Encoding = "bytes" is used verbatim; other character input is converted to UTF-8 once.

The result contains three typed data frames: map (the validation target), documents (one summary row per document), and diagnostics (zero or more findings). Diagnostic severities are "info", "warning", and "error"; document statuses are "valid", "warning", "error", and "not_validated".

Value

An S3 object of class robots_validations, containing map, documents, and diagnostics data frames.

See Also

robots_validate_url() to fetch and validate, robots_fetch() for acquisition evidence, and allowed_by_robots_text() for URL decisions.

Examples

```
report <- robots_validate_text(
  "user-agent: *\ndisallow: /private\n"
)
report$documents
report$diagnostics

malformed <- robots_validate_text(charToRaw(
  "disallow: /before-agent\nunknown-field: value"
))
malformed$diagnostics
```

---

robots_validate_url *Fetch and validate robots.txt documents for URLs*

Description

Uses robots_fetch() unchanged, then validates each distinct stored raw body exactly once. Shared origins therefore remain shared and no second HTTP path or refetch is introduced. Acquisition failures are retained as evidence and produce "not_validated" document rows; a response exceeding max_bytes is explicitly reported as acquisition_limit_exceeded, and no truncated body is parsed.

Usage

```
robots_validate_url(
  url,
  timeout = 10,
  max_bytes = 524288L,
  fetch_user_agent = NULL,
  ssrf_guard = TRUE
)
```

Arguments

`url`	A character vector of target URLs. May have length zero (an empty result is returned). A wrong R type is a call-level error.
`timeout`	A single positive, finite numeric total timeout in seconds per origin, including redirects. Defaults to `10`.
`max_bytes`	A single positive, finite, whole-number decoded-body byte limit no greater than `.Machine$integer.max`, coerced to integer and recorded on each source row. A whole-number double (for example `524288`) is accepted; fractional, non-positive, out-of-range, or non-scalar values are call-level errors. Defaults to `524288L`.
`fetch_user_agent`	`NULL` (use the package fetch user agent) or a single non-empty character HTTP user agent to send instead.
`ssrf_guard`	A single logical. When `TRUE` (the default) a structural SSRF guard refuses, before any request, both the initial origin and every redirect target that resolves to a private, loopback, link-local, or cloud-metadata address (outcome `ssrf_blocked`, no socket opened). Set to `FALSE` to deliberately opt out — for example to fetch `robots.txt` from a trusted intranet host. Opting out removes that protection; only disable it for hosts you control and trust.

Details

This function validates the acquired robots.txt document. It does not answer whether any target URL is allowed; use `allowed_by_robots_url()` for that separate question. It also does not infer webmaster intent or check whether declared remote sitemaps exist.

Value

An S3 object of class `robots_validations`, containing `map` (one row per input URL), `documents` (one row per distinct fetch source), and `diagnostics` data frames.

See Also

`robots_validate_text()` for supplied bytes and `robots_fetch()` for acquisition without document validation.

Examples

```
fetched_validation <- httr2::with_mocked_responses(
  function(req) {
    httr2::response(
      status_code = 200L, url = req$url,
      body = charToRaw("user-agent: *\nDisallow: /private\n")
    )
  },
  robots_validate_url("https://example.com/private")
)
fetched_validation$documents
```

Index

allowed_by_robots_text, 3
allowed_by_robots_text(), *6*, *21*
allowed_by_robots_url, 4
allowed_by_robots_url(), *18*, *22*
as_legacy_robots_decisions_v1, 6

print.robots_decisions, 7
print.robots_fetches, 8
print.robots_validations, 9

robots_body, 9
robots_engine_contract_v1, 11
robots_engine_contract_v1(), *12*, *20*
robots_engine_contract_v2, 12
robots_evaluate_text_v1, 13
robots_evaluate_text_v1(), *16*
robots_evaluate_url_v1, 14
robots_fetch, 16
robots_fetch(), *6*, *21*, *22*
robots_resolve_matcher_profile_v1, 19
robots_resolve_matcher_profile_v1(), *13*, *15*
robots_validate_text, 20
robots_validate_text(), *22*
robots_validate_url, 21
robots_validate_url(), *21*