

Package: sitemapr (via r-universe)

October 12, 2026

Type Package

Title Parse and Validate XML Sitemaps

Version 0.0.0.9000

Description A deterministic toolkit for reading and validating XML, text, and index sitemaps against the Sitemap Protocol 0.9 and related W3C and RFC standards. It parses sitemap sources into tidy data frames, expands sitemap indexes, and reports schema and protocol findings with stable, reproducible output.

License MIT + file LICENSE

URL <https://bart-turczynski.gitlab.io/sitemapr/>,
<https://gitlab.com/bart-turczynski/sitemapr>,
<https://bart-turczynski.r-universe.dev/sitemapr>

BugReports <https://gitlab.com/bart-turczynski/sitemapr/-/issues>

Imports htmltools, httr2 (>= 1.1.1), rlang, rurl (>= 3.0.1), tibble, utils, xml2

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

Suggests cucumber (>= 2.0.0), knitr, oysteR, parallel, rmarkdown, robotstxt (>= 0.3.0), rosv, testthat (>= 3.0.0), tools, withr

Remotes gitlab::bart-turczynski/robotstxt@v0.3.0

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libicu-dev libxml2-dev libssl-dev

Repository <https://bart-turczynski.r-universe.dev>

Date/Publication 2026-10-12 22:00:11 UTC

RemoteUrl <https://gitlab.com/bart-turczynski/sitemapr>

RemoteRef HEAD

RemoteSha ff3e2add9339fbd422e1409b4abf4cdd802e0c47

Contents

audit_accessors	3
audit_sitemap	4
audit_unchanged	6
compare_sitemap_audits	7
discovery_limits	8
fetch_limits	9
gsc_submission	10
index_limits	11
probe_url	12
read_sitemap	14
report_sitemap	15
request_auth_basic	18
request_auth_bearer	19
request_policy	19
request_proxy	21
request_retry	22
request_throttle	23
robots_context	24
robots_context_preset	26
robots_context_presets	27
robots_cross_submission	27
ruleset_context	28
ruleset_context_for_child	29
ruleset_revision	31
sitemap_audit	31
sitemap_companions	33
sitemap_contract	34
sitemap_rulesets	35
sitemap_tree	36
sitemap_tree_from_bytes	38
validate_sitemap	39
validate_sitemap_robots	42
validate_sitemap_ruleset	45

Index

49

audit_accessors	<i>Access the components of a sitemap audit</i>
-----------------	---

Description

Component accessors for a `sitemap_audit()` object. Each returns the stable tidy shape the package already produces: `audit_urls()` the `read_sitemap()` row tibble, `audit_findings()` the `validate_sitemap()` findings contract, `audit_sources()` the per-source fetch-metadata records, `audit_problems()` the non-fatal parse problems table, and `audit_tree()` the `sitemap_tree()` discovery/index structure.

Usage

```
audit_urls(x)

audit_findings(x)

audit_sources(x)

audit_problems(x)

audit_tree(x)
```

Arguments

`x` A `sitemap_audit` object, as from `sitemap_audit()`.

Value

The requested component: a tibble or data frame with the documented columns (zero rows for an empty component).

Examples

```
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  '<url><loc>https://example.com/</loc></url></urlset>'
)
path <- tempfile(fileext = ".xml")
writeLines(xml, path)
audit <- sitemap_audit(urls = read_sitemap(path))

audit_urls(audit)
audit_findings(audit)
audit_sources(audit)
audit_problems(audit)
audit_tree(audit)
```

audit_sitemap

*Audit a sitemap in a single pass***Description**

Resolves each sitemap source — a sitemap URL or a local sitemap file — just ONCE and derives both the tidy URL table and the validation findings from that single resolved artifact, returning them together as a `sitemap_audit()` object. It is the single-pass equivalent of calling `read_sitemap()` and `validate_sitemap()` separately: it fetches every remote root (and every sitemap-index child) once instead of twice, and the two projections are guaranteed to describe the same snapshot of the source.

Usage

```
audit_sitemap(
  x,
  mode = c("strict", "non-strict"),
  user_agent = default_user_agent(),
  limits = fetch_limits(),
  index_limits = NULL,
  policy = request_policy(),
  collect = TRUE,
  on_urls = NULL,
  max_active = NULL
)
```

Arguments

<code>x</code>	One or more sitemap URLs or paths to local sitemap files (<code>.xml</code> , <code>.txt</code> , <code>.gz</code> , or <code>.tar.gz</code>).
<code>mode</code>	"strict" (the default) or "non-strict", passed through to the findings projection exactly as <code>validate_sitemap()</code> uses it.
<code>user_agent</code>	The User-Agent header for HTTP fetches. Defaults to the package User-Agent.
<code>limits</code>	Network limits for HTTP fetches, as from <code>fetch_limits()</code> .
<code>index_limits</code>	Sitemapindex-expansion bounds (recursion depth and per-index child-count cap), as from <code>index_limits()</code> . Defaults to <code>index_limits()</code> .
<code>policy</code>	A <code>request_policy()</code> applied to every HTTP hop (root, robots.txt, discovery, redirects, and index children) — configure custom headers, authentication, a proxy, TLS options, retry/backoff, and per-host throttling there. Defaults to the no-op policy.
<code>collect</code>	When TRUE (the default) the tidy URL rows from every source are collected into the returned <code>urls</code> component, exactly as before. When FALSE the audit runs in STREAMING mode: each completed leaf sitemap's rows are handed to <code>on_urls</code> (if supplied) and then discarded, so peak retained-row memory is bounded to a single leaf rather than the whole hierarchy. This lets a valid

	protocol-scale sitemap index be processed without materializing one combined in-memory table.
on_urls	Optional streaming callback function(<code>rows</code> , <code>source</code>) invoked ONCE per completed leaf sitemap with that leaf's tidy rows and its per-source fetch-metadata record. Supplying it activates streaming mode (as if <code>collect = FALSE</code>). An error thrown by the callback aborts the audit cleanly with a classed <code>sitemapr_stream_callback_error</code> condition naming the leaf; the accumulator is not left half-updated.
max_active	Optional worker cap for opt-in bounded-concurrency expansion of a sitemap index (ADR-008). <code>NULL</code> (the default) keeps child fetches sequential; a larger value (a small, polite bound such as 4–6 is typical) fetches up to that many independent child sitemaps at once. Concurrency changes only <i>when</i> a child's bytes arrive, never the row order or budget-truncation point — the result is byte-identical to sequential mode.

Details

`audit_urls()` on the result equals `read_sitemap()` on the same source, and `audit_findings()` equals `validate_sitemap()` on the same source and mode. The audit also carries the per-source fetch metadata, the non-fatal parse problems, and — for an expanded sitemap index — the discovery tree. A source-level failure (an unreachable source, a corrupt archive, an unsupported root) is recorded as a problem and a fetch-layer finding rather than aborting the whole audit, so every source stays attributable.

Value

A `sitemap_audit()` object: a classed list with the components `urls`, `findings`, `sources`, `problems`, and `tree`. Access them with `audit_urls()`, `audit_findings()`, `audit_sources()`, `audit_problems()`, and `audit_tree()`. In streaming mode (`collect = FALSE` or a supplied `on_urls`) the `urls` component is the empty row schema — the rows were streamed out per leaf — and carries the total number of streamed rows as its `"streamed_row_count"` attribute; `findings`, `sources`, `problems`, and `tree` are UNCHANGED from a collected audit, so index-protocol findings stay complete because they are derived incrementally per leaf.

See Also

`read_sitemap()` and `validate_sitemap()` for the equivalent two-call path, and `report_sitemap()`, which accepts the returned object.

Examples

```
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  '<url><loc>https://example.com/</loc>',
  '<lastmod>2024-01-01</lastmod></url>',
  '<url><loc>https://example.com/about</loc></url>',
  '</urlset>'
)
path <- tempfile(fileext = ".xml")
writeLines(xml, path)
```

```
# One pass yields both the URL table and the findings.
audit <- audit_sitemap(path)
audit_urls(audit)
audit_findings(audit)

# Streaming: consume each leaf's rows via a callback instead of collecting
# them. `urls` comes back empty, with the streamed row count as an attribute.
seen <- 0L
streamed <- audit_sitemap(
  path,
  collect = FALSE,
  on_urls = function(rows, source) seen <-<= seen + nrow(rows)
)
seen
attr(audit_urls(streamed), "streamed_row_count")
```

audit_unchanged

Did a sitemap-audit comparison find any change?

Description

The "did anything change?" predicate for a [compare_sitemap_audits\(\)](#) result: TRUE when the two audits are identical (an empty diff), FALSE otherwise.

Usage

```
audit_unchanged(x)
```

Arguments

x A `sitemap_audit_diff`, as returned by [compare_sitemap_audits\(\)](#).

Value

A length-one logical: TRUE if no rows were added, removed, or changed in any component, else FALSE.

See Also

[compare_sitemap_audits\(\)](#).

Examples

```
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  '<url><loc>https://example.com/</loc></url></urlset>'
)
path <- tempfile(fileext = ".xml")
writeLines(xml, path)
audit <- sitemap_audit(urls = read_sitemap(path))
```

```
audit_unchanged(compare_sitemap_audits(audit, audit))
```

```
compare_sitemap_audits
```

Compare two sitemap audits

Description

Deterministically diffs two `sitemap_audit()` objects component-by-component and reports which rows were added, removed, or changed. Built for CI drift monitoring (e.g. a scheduled job that flags when a site's sitemap audit changes): comparing an audit to an identically-recomputed copy of itself always yields an empty diff.

Usage

```
compare_sitemap_audits(old, new)

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

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

Arguments

<code>old, new</code>	sitemap_audit objects (the baseline and the new capture).
<code>x, object, ...</code>	A sitemap_audit_diff object and ignored extra arguments, for the <code>print()</code> / <code>summary()</code> methods.

Details

Rows are matched by a stable identity key, never by position, so shuffling a component's rows never produces a diff. Keys are: `loc` for urls; `requested_url` for sources; `sitemap_url` + `parent_sitemap` + `depth` for tree; and the whole row for findings and problems (a finding present in one audit and not the other is added/removed).

Change detection ignores run-varying metadata so re-running the same audit does not show spurious diffs: the `elapsed-fetch` column `sources$timing` is excluded from the comparison. All other columns are deterministic content and are compared.

Value

A `sitemap_audit_diff`: a classed list whose components element holds, for each of `urls`, `findings`, `sources`, `problems`, and `tree`, a list of added, removed, and changed tibbles (each carrying the component's columns, deterministically ordered by key). Use `audit_unchanged()` to test whether anything changed, and `summary()` for a per-component count table.

See Also

`sitemap_audit()` for the compared container, and `audit_unchanged()` for the "did anything change?" predicate.

Examples

```
base <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  "<url><loc>https://example.com/</loc></url>",
  "<url><loc>https://example.com/a/</loc></url></urlset>"
)
moved <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  "<url><loc>https://example.com/</loc></url>",
  "<url><loc>https://example.com/b/</loc></url></urlset>"
)
p1 <- tempfile(fileext = ".xml")
p2 <- tempfile(fileext = ".xml")
writeLines(base, p1)
writeLines(moved, p2)

old <- sitemap_audit(urls = read_sitemap(p1))
new <- sitemap_audit(urls = read_sitemap(p2))
d <- compare_sitemap_audits(old, new)
d
summary(d)

# An audit compared with itself has an empty diff.
audit_unchanged(compare_sitemap_audits(old, old))
```

discovery_limits

Default limits for discovery

Description

Returns the configurable limits applied while discovering candidates. The candidate cap bounds how many distinct guessed-path candidates are evaluated against a single site root (after deduplication), so a large catalog can never trigger an unbounded burst of requests.

Usage

```
discovery_limits(max_candidates = getOption("sitemapr.max_candidates", 50L))
```

Arguments

`max_candidates` Maximum number of distinct candidates to evaluate. Resolves from the argument, then `getOption("sitemapr.max_candidates")`, then the default of 50.

Value

A named list of limits with coerced types.

See Also

`fetch_limits()` and `index_limits()` for the other bound constructors, and `sitemap_tree()` which accepts discovery limits.

Examples

```
discovery_limits()
discovery_limits(max_candidates = 10)
```

fetch_limits	<i>Default limits for a bounded HTTP fetch</i>
--------------	--

Description

Returns the configurable network limits the fetch engine applies per request. Each limit resolves from the matching argument, falling back to a `getOption("sitemapr.*")` value, then to the ADR-003 default. No limit is a non-overridable hard cap: a caller may raise any of them by passing an argument or setting the option.

Usage

```
fetch_limits(
  timeout = getOption("sitemapr.timeout", 30),
  max_redirects = getOption("sitemapr.max_redirects", 5L),
  max_bytes = getOption("sitemapr.max_bytes", 500L * 1024L^2)
)
```

Arguments

<code>timeout</code>	Per-request timeout in seconds (numeric).
<code>max_redirects</code>	Maximum number of redirects to follow (integer).
<code>max_bytes</code>	Per-resource safety ceiling in bytes (integer): the hard cap on the body read into memory. Exceeding it discards the body and raises a <code>sitemapr_body_ceiling</code> condition. Default 500 MB.

Details

Defaults (ADR-003 §3): request timeout 30 s, max redirects 5, per-resource safety ceiling 500 MB. The 50 MB sitemap-protocol size limit is NOT enforced here — it is a non-fatal validation finding (`PROTOCOL_SIZE_EXCEEDED`); the fetch layer only records body size and guards memory with the ceiling.

Value

A named list of limits with coerced types.

See Also

[index_limits\(\)](#) and [discovery_limits\(\)](#) for the other bound constructors, and [read_sitemap\(\)](#) which accepts limits.

Examples

```
fetch_limits()
fetch_limits(timeout = 10, max_redirects = 3, max_bytes = 1024^2)
```

gsc_submission	<i>Search Console submission context preset (ADR-009 spec 12.1)</i>
----------------	---

Description

A construction-time preset for a Google Search Console submission. It bundles the three per-source axes a GSC submission implies into one coherent context so a caller need not hand-set each one: `submission_channel = "search_console_api"`, `authority_evidence = "verified_property_set"`, and the verified `property_scope` the submission is bound to. It is a thin wrapper over [ruleset_context\(\)](#); every axis stays independently overridable by calling [ruleset_context\(\)](#) directly.

Usage

```
gsc_submission(property)
```

Arguments

property	The verified property/site URL the submission is bound to. Required: it is the irreducible input and cannot be inferred. Its shape is validated by ruleset_context() .
----------	--

Value

An object of class `sitemapr_ruleset_context`, the same shape [ruleset_context\(\)](#) returns.

See Also

[ruleset_context\(\)](#) for the general constructor and its axes.

Examples

```
gsc_submission("https://example.com/")
```

index_limits

*Default limits for sitemapindex expansion***Description**

Returns the configurable bounds the index-expansion engine applies while recursively expanding a sitemapindex. Both are safety bounds, not protocol rules: they keep a hostile or accidentally huge index tree from triggering an unbounded burst of requests or unbounded recursion.

Usage

```
index_limits(
  max_depth = getOption("sitemapr.max_index_depth", 3L),
  max_children = getOption("sitemapr.max_index_children", 50000L),
  max_total_sitemaps = getOption("sitemapr.max_total_sitemaps", 50000L),
  max_total_urls = getOption("sitemapr.max_total_urls", 2.5e+07)
)
```

Arguments

max_depth	Maximum recursion depth below the root index (integer). Resolves from the argument, then <code>getOption("sitemapr.max_index_depth")</code> , then the default of 3.
max_children	Maximum number of distinct child entries expanded per index (integer). Resolves from the argument, then <code>getOption("sitemapr.max_index_children")</code> , then the default of 50 000 (the sitemap-protocol per-index entry limit).
max_total_sitemaps	Maximum number of child sitemaps fetched across the entire traversal (numeric). Resolves from the argument, then <code>getOption("sitemapr.max_total_sitemaps")</code> , then the default of 50 000 (the sitemap-protocol per-index entry limit, reused as the traversal-wide ceiling). Kept numeric so Inf is representable.
max_total_urls	Maximum number of URL rows gathered across the entire traversal (numeric). Resolves from the argument, then <code>getOption("sitemapr.max_total_urls")</code> , then the default of 25 000 000. Kept numeric so Inf is representable.

Details

max_depth counts levels below the root index: the root index is depth 0, its children are depth 1, and an index whose children would land beyond max_depth is not descended (an INDEX_DEPTH_EXCEEDED event). max_children caps how many child entries one index contributes after dedup; entries beyond the cap are dropped (an INDEX_CHILD_COUNT_EXCEEDED event). max_total_sitemaps and max_total_urls are TRAVERSAL-WIDE aggregate budgets spanning the whole recursive expansion: the first caps how many child sitemaps are fetched in total (an INDEX_TOTAL_SITEMAPS_EXCEEDED event), the second caps how many URL rows are gathered in total (an INDEX_TOTAL_URLS_EXCEEDED event). Both are FINITE by default — 50 000 child sitemaps and 25 000 000 URL rows, the ceilings a traversal of protocol-legal sitemaps cannot exceed in practice — so an embedding caller is never

exposed to an unbounded traversal over a pathological index graph. Pass `Inf` explicitly to opt back out. When a budget is reached the traversal stops and returns the accumulated `PARTIAL` result.

Value

A named list of limits with coerced types.

See Also

[fetch_limits\(\)](#) and [discovery_limits\(\)](#) for the other bound constructors, and [read_sitemap\(\)](#) which accepts `index_limits`.

Examples

```
index_limits()
index_limits(max_depth = 2, max_children = 1000)
index_limits(max_total_sitemaps = 500, max_total_urls = 1e6)
```

probe_url	<i>Probe a URL to inspect what it is, without resolving it</i>
-----------	--

Description

`probe_url()` is a lightweight, **non-resolving** inspection: it fetches exactly one URL (or reads one local file) and reports what it is — a sitemap, a sitemap index, a feed, robots.txt, an HTML page, or an error state — **without** expanding indexes or following children. It is the diagnostic counterpart to [read_sitemap\(\)](#), which resolves a sitemap to its final URL rows. Use `probe_url()` to diagnose first and [read_sitemap\(\)](#) to resolve.

Usage

```
probe_url(
  url,
  limits = fetch_limits(),
  user_agent = default_user_agent(),
  ssrf_guard = TRUE,
  policy = request_policy()
)
```

Arguments

<code>url</code>	A single sitemap URL or local file path to inspect.
<code>limits</code>	Network limits for the fetch, as from fetch_limits() .
<code>user_agent</code>	The User-Agent header for the fetch. Defaults to the package User-Agent.
<code>ssrf_guard</code>	Logical; when <code>TRUE</code> (default) the structural SSRF guard runs on every hop.
<code>policy</code>	A request policy applied to the fetch, as from request_policy() . Defaults to the no-op policy.

Details

The fetch goes through the same bounded, SSRF-safe engine as the rest of the package, so `probe_url()` inherits the network-safety policy (per-hop SSRF guard, redirect and body-size ceilings) and the limits / policy overrides. It fetches **only** the given URL: for a sitemap index it *counts* the direct `<sitemap>` children by parsing the index locally, but it never fetches them.

A fetch or parse failure is represented in the returned record (`detected_type` becomes `"fetch_error"`, `"not_found"`, or `"parse_error"`, with an explanatory problems row), not thrown — `probe_url()` is diagnostic. Only invalid input (a non-string url) raises an error.

Value

A `sitemapr_probe` object: a classed list with the fields `url`, `final_url`, `status_code`, `content_type`, `detected_type`, `xml_root`, `is_compressed`, `child_count`, `sample`, `problems`, and `suggested_next`. `detected_type` is one of `"sitemap"`, `"sitemap_index"`, `"feed"`, `"robots_txt"`, `"html"`, `"xml_other"`, `"not_found"`, `"fetch_error"`, or `"parse_error"`.

See Also

[read_sitemap\(\)](#) to resolve a sitemap to its URL rows, and [sitemap_tree\(\)](#) to discover a site's sitemaps from its root.

Examples

```
# Probe a local sitemap file: detected as a urlset, with a child count.
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  "<url><loc>https://example.com/</loc></url>",
  "<url><loc>https://example.com/about</loc></url>",
  "</urlset>"
)
path <- tempfile(fileext = ".xml")
writeLines(xml, path)
probe_url(path)

# A sitemap index: children are COUNTED, never fetched.
index <- paste0(
  '<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  "<sitemap><loc>https://example.com/s1.xml</loc></sitemap>",
  "<sitemap><loc>https://example.com/s2.xml</loc></sitemap>",
  "</sitemapindex>"
)
index_path <- tempfile(fileext = ".xml")
writeLines(index, index_path)
probe_url(index_path)

# Probe a live URL (requires network):
# probe_url("https://example.com/sitemap.xml")
```

read_sitemap

*Read a sitemap into a tidy tibble of URLs***Description**

Parses one or more sitemap sources — sitemap URLs or local sitemap files — into the tidy row tibble: one row per URL with `loc`, `lastmod`, `changefreq`, `priority`, the `images/video/news/alternates` extension list-columns, and `source_sitemap` provenance. Supported formats are XML `urlset` and `sitemapindex`, RSS 2.0 and Atom 0.3/1.0 feeds, the plain-text format, transparent `gzip` (`.xml.gz/.txt.gz`), and — local files only — bounded `.tar.gz` archives. A feed is parsed into the same row schema, one row per item/entry.

Usage

```
read_sitemap(
  x,
  user_agent = default_user_agent(),
  limits = fetch_limits(),
  index_limits = NULL,
  policy = request_policy(),
  max_active = NULL
)

read_sitemaps(
  x,
  user_agent = default_user_agent(),
  limits = fetch_limits(),
  index_limits = NULL,
  policy = request_policy(),
  max_active = NULL
)
```

Arguments

<code>x</code>	One or more sitemap URLs or paths to local sitemap files (<code>.xml</code> , <code>.txt</code> , <code>.gz</code> , or <code>.tar.gz</code>).
<code>user_agent</code>	The User-Agent header for HTTP fetches. Defaults to the package User-Agent.
<code>limits</code>	Network limits for HTTP fetches, as from <code>fetch_limits()</code> .
<code>index_limits</code>	Sitemapindex-expansion bounds (recursion depth and per-index child-count cap), as from <code>index_limits()</code> . Defaults to <code>index_limits()</code> .
<code>policy</code>	A request_policy() applied to every HTTP hop (root, robots.txt, discovery, redirects, and index children) — configure custom headers, authentication, a proxy, TLS options, retry/backoff, and per-host throttling there. Defaults to the no-op policy.

`max_active` Optional worker cap for opt-in bounded-concurrency expansion of a sitemap index (ADR-008). NULL (the default) keeps child fetches sequential; a larger value (a small, polite bound such as 4–6 is typical) fetches up to that many independent child sitemaps at once. Concurrency changes only *when* a child's bytes arrive, never the row order or budget-truncation point — the result is byte-identical to sequential mode.

Details

The result carries a `sources` attribute (the per-source fetch-metadata records) and a `problems` attribute (a tibble of non-fatal issues such as skipped archive members, unfetchable index children, or failed members of a submitted vector). `read_sitemap()` never returns a validation findings tibble; use `validate_sitemap()` for that.

A top-level sitemap index is expanded recursively (cycle-safe, depth- and count-capped) so every reachable child sitemap's rows carry per-child provenance; the bounds are configurable via `index_limits`.

When `x` contains more than one source, inputs are normalized, deduplicated, and capped using the submitted-list source-record policy. Per-source failures are recorded in the `problems` attribute and successful sources still contribute rows. Scalar calls keep the stricter historical behavior: an entry-point fetch failure or unsupported content raises a classed condition.

Value

A tibble of URL rows with `sources` and `problems` attributes. An entry-point fetch failure or unsupported content raises a classed error condition.

Examples

```
# Read a local sitemap file into a tidy tibble of URLs.
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  '<url><loc>https://example.com/</loc>',
  '<lastmod>2024-01-01</lastmod></url>',
  '<url><loc>https://example.com/about</loc></url>',
  '</urlset>'
)
path <- tempfile(fileext = ".xml")
writeLines(xml, path)
read_sitemap(path)

# Read directly from a sitemap URL; a top-level index expands recursively.
# read_sitemap("https://example.com/sitemap.xml")
```

Description

Produces a single, fully self-contained HTML report for a sitemap source, modeled on the sitemap-validator reference renderer. Unlike [read_sitemap\(\)](#) (which returns a tidy tibble) and [validate_sitemap\(\)](#) (which returns the findings contract), [report_sitemap\(\)](#) is the human-readable surface: it *consumes* those existing outputs and renders them, and never re-implements any parsing or validation itself.

Usage

```
report_sitemap(
  x,
  output = NULL,
  mode = c("strict", "non-strict"),
  urls = NULL,
  findings = NULL,
  audit = NULL,
  title = NULL,
  user_agent = default_user_agent(),
  limits = fetch_limits(),
  index_limits = NULL,
  policy = request_policy()
)
```

Arguments

x	A single source: a sitemap URL (character) or a path to a local sitemap file. When both <code>urls</code> and <code>findings</code> are supplied, <code>x</code> is used only as the displayed source label.
output	Optional path to write the HTML file to. When supplied, the report is written there (UTF-8) and the path is returned invisibly; otherwise the HTML is returned as an <code>htmltools::HTML</code> string.
mode	"strict" (the default) or "non-strict", passed to validate_sitemap() when <code>findings</code> is not supplied.
urls	Optional precomputed read_sitemap() result (a tibble with the <code>sources</code> attribute). When <code>NULL</code> (the default) it is computed from <code>x</code> .
findings	Optional precomputed validate_sitemap() findings tibble. When <code>NULL</code> (the default) it is computed from <code>x</code> .
audit	Optional audit_sitemap() result supplying both the URL rows and the findings from a single pass (so a URL source is fetched once, not twice). It fills whichever of <code>urls/findings</code> you leave <code>NULL</code> ; an explicit <code>urls</code> or <code>findings</code> still wins. <code>x</code> is then used only as the source label.
title	The HTML document <code><title></code> . Defaults to a title derived from <code>x</code> .
user_agent	The User-Agent header for HTTP fetches. Defaults to the package User-Agent.
limits	Network limits for HTTP fetches, as from fetch_limits() .
index_limits	Sitemapindex-expansion bounds (recursion depth and per-index child-count cap), as from index_limits() . Defaults to index_limits() .

`policy` A `request_policy()` applied to every HTTP hop (root, robots.txt, discovery, redirects, and index children) — configure custom headers, authentication, a proxy, TLS options, retry/backoff, and per-host throttling there. Defaults to the no-op policy.

Details

The report contains a hero banner (source, overall status, URL/index/sitemap counts), a per-source sitemap table (format, HTTP status, URL count, and lastmod/priority/changefreq presence), lastmod coverage cards with a by-month histogram, a collapsible URL folder tree grouped by path segment, a severity dashboard, the findings grouped by validation layer (deduplicated by code, with evidence excerpts), the checks that ran and passed, prescriptive recommendations, and a searchable, sortable, CSV-exportable URL table.

Two sections read the run rather than the findings. The **Checks** section is driven by the finding-code registry: it enumerates the checks this package actually implements, marks those that ran and did not fire as passed, and names the layers the run did not exercise as neither passed nor failed — so an empty findings table can be told apart from a validation that never ran. A layer counts as having run only on positive evidence, which understates rather than overstates: `check_robots = TRUE` leaves no trace when nothing is disallowed, so a clean robots run reads as "not exercised". A check is also gated on the registry's `ruleset` column, independently of its layer: a baseline call cannot reach an engine-gated emitter however thoroughly it exercised the surrounding layer, so those checks are reported as unknown rather than clean. Pass the result of `validate_sitemap_ruleset()` as findings to have the selected engine's checks counted. The **Recommendations** section is prescriptive rather than diagnostic — stale or absent lastmod, priority/changefreq that the major engines ignore, and the 50,000-URL / 50 MB / 50,000-child bounds as they are approached — each carrying its source and one provenance tag.

The output is entirely self-contained: all CSS, JavaScript (search, sort, CSV export, tree toggle, and a light/dark theme toggle), and data are inlined, so the file references no external hosts and works offline. The palette follows the viewer's `prefers-color-scheme` by default; the in-page toggle stamps a `data-theme` attribute on the root element that wins in both directions.

By default the source `x` is both read (via `read_sitemap()`, for the URL rows and per-source meta-data) and validated (via `validate_sitemap()`, for the findings). To avoid re-fetching a URL source, or to render results you have already computed, pass them via `urls` and/or `findings`; in that case `x` is used only as the report's source label.

Value

If output is supplied, the output path, invisibly. Otherwise, the report HTML as an `htmltools::HTML` character string.

See Also

`read_sitemap()` and `validate_sitemap()` for the underlying data.

Examples

```
# Render a report for a local sitemap file to a temporary HTML file.
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
```

```

'<url><loc>https://example.com/</loc>',
'<lastmod>2024-01-01</lastmod></url>',
'<url><loc>https://example.com/about</loc></url>',
'</urlset>'
)
path <- tempfile(fileext = ".xml")
writeLines(xml, path)
out <- tempfile(fileext = ".html")
report_sitemap(path, output = out)

# Render directly from a sitemap URL (fetches twice: read + validate).
# report_sitemap("https://example.com/sitemap.xml", output = "report.html")

# Reuse results you have already computed to avoid re-fetching.
# u <- read_sitemap("https://example.com/sitemap.xml")
# f <- validate_sitemap("https://example.com/sitemap.xml")
# report_sitemap(
#   "example.com", urls = u, findings = f, output = "report.html"
# )

```

request_auth_basic	<i>Basic-authentication credentials for a request-policy</i>
--------------------	--

Description

Basic-authentication credentials for a request-policy

Usage

```
request_auth_basic(username, password)
```

Arguments

username, password
 Credential strings, applied via `httr2::req_auth_basic()`.

Value

A `sitemapr_request_auth` object for `request_policy(auth = )`.

See Also

Other request-policy: [request_auth_bearer\(\)](#), [request_policy\(\)](#), [request_proxy\(\)](#), [request_retry\(\)](#), [request_throttle\(\)](#)

Examples

```
request_policy(auth = request_auth_basic("user", "secret"))
```

request_auth_bearer	<i>Bearer-token authentication for a request-policy</i>
---------------------	---

Description

Bearer-token authentication for a request-policy

Usage

```
request_auth_bearer(token)
```

Arguments

token	Bearer token string, applied via <code>httr2::req_auth_bearer_token()</code> .
-------	--

Value

A `sitemapr_request_auth` object for `request_policy(auth = )`.

See Also

Other request-policy: [request_auth_basic\(\)](#), [request_policy\(\)](#), [request_proxy\(\)](#), [request_retry\(\)](#), [request_throttle\(\)](#)

Examples

```
request_policy(auth = request_auth_bearer("a-token"))
```

request_policy	<i>Construct a request-policy for the fetch boundary</i>
----------------	--

Description

A request-policy is the single, constrained extension seam for customizing the `httr2` requests `sitemapr` issues for every hop (root, redirect, discovery, robots, and index-child requests). It carries typed fields for custom HTTP headers, authentication, a proxy, and TLS/curl options, plus a generic prepare hook for anything not covered by the typed fields. Every field defaults to `NULL`, so the default policy is a byte-identical no-op and existing callers behave exactly as before.

Usage

```
request_policy(
  prepare = NULL,
  headers = NULL,
  auth = NULL,
  proxy = NULL,
  tls = NULL,
  retry = NULL,
  throttle = NULL,
  max_active = NULL
)
```

Arguments

prepare	Optional function(req, ctx) receiving the built httr2 request plus a hop-context list (currently the resolved hop url) and returning the possibly-modified request. Applied last of the caller-facing steps, so it composes on top of the typed fields.
headers	Optional named list or named character vector of HTTP headers to add via <code>httr2::req_headers()</code> .
auth	Optional authentication object from request_auth_basic() or request_auth_bearer() , applied via <code>httr2::req_auth_basic()</code> / <code>httr2::req_auth_bearer_token()</code> .
proxy	Optional proxy: either a proxy URL string or a request_proxy() object, applied via <code>httr2::req_proxy()</code> .
tls	Optional named list of curl/TLS options (e.g. <code>list(ssl_verifypeer = 0L)</code>) passed through <code>httr2::req_options()</code> . Cannot override sitemapr's redirect controls, which are re-asserted afterwards.
retry	Optional request_retry() object enabling bounded retry with exponential back-off. NULL (the default) means a single attempt per hop — byte-identical to the pre-retry behavior. When set, only TRANSIENT failures retry: the retryable HTTP status set (default 429, 500, 502, 503, 504) and, when <code>retry_on_failure = TRUE</code> , transient transport errors. Retries happen inside a single hop's <code>req_perform()</code> , so they never inflate the redirect/hop count or consume the redirect budget. Deterministic failures are NEVER retried: SSRF rejections (raised before the request), malformed input (classified downstream), and resource-ceiling aborts (raised after the response is buffered) all sit outside the retried code path. Any <code>Retry-After</code> header is honored but BOUNDED by the configured max backoff.
throttle	Optional request_throttle() object enabling host-aware request pacing. NULL (the default) means no pacing, byte-identical to the pre-throttle behavior. When set, requests are keyed by canonical host:port, so different origins pace independently and a redirect onto another host pays that host's pace, not the origin's. Pacing runs around each request (after the per-hop SSRF guard and sitemapr's re-asserted transport controls), so it never weakens those safety semantics.
max_active	Optional worker cap enabling opt-in bounded-concurrency expansion of a sitemapr index (ADR-008). NULL (the default) keeps child fetches sequential — byte-identical to the pre-concurrency path — and <code>max_active = 1</code> is observably identical to that default. A larger value (a small, polite bound such as 4–6 is typical)

fetches up to that many independent child sitemaps at once while the per-host throttle still paces each origin. Concurrency changes only *when* a child's bytes arrive, never the row/finding/tree order or the budget-truncation point: the output is byte-identical to sequential mode.

Details

What callers may set. headers, auth, proxy, tls, and prepare are all applied to each hop's request AFTER the per-hop SSRF guard and BEFORE sitemapr asserts its own transport controls.

What sitemapr always owns (callers cannot override). Redirect control (followlocation = 0, maxredirects = 0) and the non-2xx error policy are re-applied *after* all caller customization, and the structural SSRF guard re-runs on every redirect hop before any network call. A policy therefore cannot re-enable automatic redirect following, defeat the per-hop SSRF re-check, or turn a non-2xx status into a transport error — even via tls/prepare — because sitemapr's controls win last.

Value

An object of class sitemapr_request_policy.

See Also

Other request-policy: [request_auth_basic\(\)](#), [request_auth_bearer\(\)](#), [request_proxy\(\)](#), [request_retry\(\)](#), [request_throttle\(\)](#)

Examples

```
# (1) A custom header (e.g. to satisfy a staging gateway).
request_policy(headers = list("X-Env" = "staging"))

# (2) Basic and bearer authentication.
request_policy(auth = request_auth_basic("user", "secret"))
request_policy(auth = request_auth_bearer("a-token"))

# (3) A corporate proxy.
request_policy(proxy = "http://proxy.internal:3128")
request_policy(proxy = request_proxy("proxy.internal", port = 3128))

# (4) Bounded-concurrency index expansion (opt-in; sequential by default).
request_policy(max_active = 6)
```

request_proxy

Proxy settings for a request-policy

Description

Proxy settings for a request-policy

Usage

```
request_proxy(
  url,
  port = NULL,
  username = NULL,
  password = NULL,
  auth = "basic"
)
```

Arguments

url	Proxy host or URL.
port	Optional proxy port.
username, password	Optional proxy credentials.
auth	Proxy authentication scheme (httr2::req_proxy() default "basic").

Value

A `sitemapr_request_proxy` object for `request_policy(proxy = )`.

See Also

Other request-policy: [request_auth_basic\(\)](#), [request_auth_bearer\(\)](#), [request_policy\(\)](#), [request_retry\(\)](#), [request_throttle\(\)](#)

Examples

```
request_policy(proxy = request_proxy("proxy.internal", port = 3128))
```

request_retry

Retry-and-backoff settings for a request-policy

Description

Configures bounded retry with exponential backoff for a request-policy. Pass the result as `request_policy(retry = )`. Only TRANSIENT failures retry: the retryable HTTP statuses set, plus transient transport errors when `retry_on_failure = TRUE`. Deterministic failures (SSRF, malformed input, resource-ceiling aborts) are never retried because they surface outside the single-hop `req_perform()` that `retry` wraps.

Usage

```
request_retry(
  max_tries = 3L,
  statuses = c(429L, 500L, 502L, 503L, 504L),
  backoff_min = 1,
  backoff_max = 30,
  retry_on_failure = FALSE
)
```

Arguments

max_tries Total attempts per hop (≥ 1). 1 means no retry.

statuses Integer vector of retryable HTTP statuses. Defaults to the transient set 429, 500, 502, 503, 504.

backoff_min, backoff_max Lower/upper bounds (seconds) for the exponential backoff between attempts. The delay for attempt i is $\text{backoff_min} * 2^{(i - 1)}$, clamped to backoff_max . A Retry-After header is honored but likewise bounded by backoff_max , so no unbounded sleep.

retry_on_failure When TRUE, transient transport errors (curl failures) are also retried. Defaults to FALSE (status-driven retry only).

Value

A `sitemapr_request_retry` object for `request_policy(retry = )`.

See Also

Other request-policy: [request_auth_basic\(\)](#), [request_auth_bearer\(\)](#), [request_policy\(\)](#), [request_proxy\(\)](#), [request_throttle\(\)](#)

Examples

```
# Up to 4 attempts per hop, retrying the transient status set.
request_policy(retry = request_retry(max_tries = 4L))
```

request_throttle	<i>Host-aware request-throttle settings for a request-policy</i>
------------------	--

Description

Configures per-host request pacing for a request-policy. Pass the result as `request_policy(throttle = )`. Requests are keyed by canonical host:port, so different origins pace independently and a redirect onto another host pays that host's pace, not the origin's. Supply EITHER a minimum interval between consecutive requests to one host (`min_interval`, seconds) OR a request budget per time window (requests per window seconds, evenly paced at $\text{window} / \text{requests}$). Pacing is shared

across every hop of one logical operation (roots, redirects, robots, discovery candidates, and index children), so an index traversal's many children pace against one set of host buckets. NULL (the default policy field) means no pacing.

Usage

```
request_throttle(min_interval = NULL, requests = NULL, window = NULL)
```

Arguments

`min_interval` Minimum seconds between consecutive requests to one host.

`requests, window` A request budget: at most requests requests per window seconds to one host, evenly paced (interval window / requests). Both must be supplied together and take precedence over `min_interval`.

Value

A `sitemapr_request_throttle` for `request_policy(throttle = )`.

See Also

Other request-policy: [request_auth_basic\(\)](#), [request_auth_bearer\(\)](#), [request_policy\(\)](#), [request_proxy\(\)](#), [request_retry\(\)](#)

Examples

```
# At most one request per host every 2 seconds.
request_throttle(min_interval = 2)

# A budget of 10 requests per 60 seconds per host (evenly paced).
request_throttle(requests = 10, window = 60)
```

robots_context	<i>Construct a robots evaluation context (ADR-009 §1, sitemap-spec §13.0)</i>
----------------	---

Description

Bundles the three **independent** robots axes one allow/disallow evaluation runs under. It is the carrier [validate_sitemap_robots\(\)](#) accepts, and it is deliberately separate from [ruleset_context\(\)](#): selecting a sitemap ruleset does not select a robots policy (ADR-009 §1). A Bing *sitemap* ruleset does not imply a Bing *robots* policy — the two are chosen independently.

Usage

```
robots_context(
  product_token = "*",
  policy_ruleset = "google",
  matcher_backend = "google"
)
```

Arguments

product_token The robots.txt group to match against; defaults to the catch-all "*".

policy_ruleset The robots status-policy ruleset; defaults to "google", the historical behavior of `validate_sitemap()`.

matcher_backend The robots matcher backend; defaults to "google".

Details

- **product_token** — the robots.txt group used for **matching**: "*" for the catch-all group, or a crawler token such as "Googlebot". This is not the HTTP request user-agent `sitemapr` fetches with.
- **policy_ruleset** — whose HTTP-status → policy semantics govern (what a 404, a 403 or a 5xx robots.txt *means*).
- **matcher_backend** — which matcher decides a rule against a URL.

The vocabulary for the last two belongs to the sibling `robotstxttr` package, not to `sitemapr`, and each value is validated against the set the **installed** build publishes on its engine contract — so an axis this build cannot honor is rejected here rather than deep inside the engine. `robotstxttr` is an optional dependency; when it is absent only the shape of each axis is checked.

A backend whose published `token_policy` is `bounded_profiles` (Bing, Yandex) accepts only its own vendor profile tokens. When `robotstxttr` is installed the token is checked here with `robotstxttr::robots_resolve_matcher_profile_v1()`, and a token the backend refuses is an error (class `robotstxttr_unresolvable_matcher_profile`) naming the accepted selectors. Without that check every URL would come back undecided and surface as `ROBOTS_INDETERMINATE`. A selector is a robots.txt `User-agent:` group label, not a crawler identity: Yandex accepts "Yandex" but not "YandexBot". [robots_context_preset\(\)](#) carries tokens known to be accepted.

Value

An object of class `sitemapr_robots_context`: a named list of the three axes.

See Also

[robots_context_preset\(\)](#) for the per-engine presets, [validate_sitemap_robots\(\)](#) for the entry point that accepts one, and [ruleset_context\(\)](#) for the independent `sitemap`-source axes.

Examples

```
robots_context()
robots_context(product_token = "Googlebot")
```

robots_context_preset *Per-engine robots context preset (sitemap-spec §13.0)*

Description

A construction-time preset selecting one engine's robots product token, status policy and matcher together, so a caller need not know which token a bounded matcher backend accepts. It is a thin wrapper over [robots_context\(\)](#); every axis stays independently overridable by calling that constructor directly.

Usage

```
robots_context_preset(preset = robots_context_presets())
```

Arguments

preset A single preset name; see [robots_context_presets\(\)](#). Defaults to "google".

Details

The preset's values are **expanded onto the returned context** rather than re-derived at use time, so the result records exactly which product token, policy ruleset and matcher backend the preset selected. The name is retained on \$preset for provenance.

A preset selects a robots engine only. It does **not** select a sitemap ruleset, and no sitemap ruleset selects it (ADR-009 §1): the bridge between the two value sets is this documented preset, never a silent derivation.

The bing preset is the Bing matcher (token "Bingbot") under RFC 9309 fetch rules, **not** a policy Bing documents: Bing publishes no robots.txt status policy, so robotstxttr's "bing" policy ruleset declines to decide every fetched robots.txt, and the preset uses the explicit application policy "assumed_rfc9309" instead. The context records that value, so results never present the assumption as a Bing policy. The yandex preset keeps "yandex" as its policy ruleset because Yandex does document one: a 200 uses the rules, and any other status allows everything.

Not every preset is runnable on every install. robotstxttr publishes a matcher_availability per backend, and one reporting capability_unavailable decides nothing — a context on it evaluates cleanly but reports every URL as ROBOTS_INDETERMINATE rather than guessing.

Value

An object of class sitemapr_robots_context, the same shape [robots_context\(\)](#) returns, carrying the expanded axis values plus the preset name.

See Also

[robots_context\(\)](#) for the general constructor and [validate_sitemap_robots\(\)](#) for the entry point that accepts the result.

Examples

```
robots_context_preset("google")

# The EXPANDED values are readable back off the context.
yandex <- robots_context_preset("yandex")
yandex$product_token
yandex$matcher_backend
```

 robots_context_presets

Supported [robots_context_preset\(\)](#) names

Description

The engine presets of the naming bridge between the sitemap-ruleset and robots-policy value sets (docs/sitemap-spec.md §13.0). "google" is deliberately first so it is the default choice, matching the axis defaults of [robots_context\(\)](#).

Usage

```
robots_context_presets()
```

Value

A character vector of the supported preset names.

See Also

[robots_context_preset\(\)](#) to build one and [robots_context\(\)](#) for the axes a preset expands to.

Examples

```
robots_context_presets()
```

 robots_cross_submission

Cross-host robots.txt submission context preset (ADR-009 spec 12.1)

Description

A construction-time preset for a sitemap found via, and trusted via, a target host's robots.txt. It bundles both senses into one coherent context: the trust (authority) sense on `authority_evidence = "target_host_robots_reference"` and the discovery sense on `discovery_provenance = "robots_txt_reference"`, keeping the two distinct. No `property_scope` is set: robots.txt reference authority is blanket, not property-bound. It is a thin wrapper over [ruleset_context\(\)](#); every axis stays independently overridable by calling [ruleset_context\(\)](#) directly.

Usage

```
robots_cross_submission()
```

Value

An object of class `sitemapr_ruleset_context`, the same shape `ruleset_context()` returns.

See Also

`ruleset_context()` for the general constructor and its axes.

Examples

```
robots_cross_submission()
```

ruleset_context	Construct a per-source validation context (ADR-009 §1, §12.1)
-----------------	---

Description

Bundles the four **independent** per-source context axes into a validated object. There is deliberately no single profile scalar (ADR-009 §1); profile stays reserved for the XSD schema-profile sense. Each axis carries its own exact, extensible value set (sitemapr owns them):

Usage

```
ruleset_context(  
  submission_channel = "absent",  
  discovery_provenance = "organic",  
  property_scope = NA_character_,  
  authority_evidence = "absent"  
)
```

Arguments

- submission_channel
One of `search_console_api`, `webmaster_tools`, `user`, `import`, or `absent` (default). `"discovered"` is rejected.
- discovery_provenance
One of `organic` (default), `robots_txt_reference`, `supplied`, `guessed_path`, `index_child`, or `archive_derived`.
- property_scope
The verified property/site string (a URL) or NA (default).
- authority_evidence
One of `absent` (default), `verified_property_set`, `target_host_robots_reference`, `same_location_default`, or `conflicting`.

Details

- `submission_channel` — how the artifact was submitted. "discovered" is **not** a submission value and is rejected; a child that inherits no submission facts is "absent".
- `discovery_provenance` — how the artifact was found. A robots.txt reference used as cross-site trust is distinct from robots.txt *discovery*.
- `property_scope` — the verified property/site a submission is bound to (a URL/site string or NA); it does not by itself grant authority beyond that property.
- `authority_evidence` — **structured, never boolean**: the kind of evidence that establishes cross-site authority for this source.

These are **per-source** facts: a sitemap-index child inherits nothing implicitly (§12.1). Build a child's context from its own evidence with `ruleset_context_for_child()`, which takes no parent context. This constructor is the pure data layer; it does not evaluate scope or authority.

Value

An object of class `sitemapr_ruleset_context`: a named list of the four axes, the shape the findings context list-column carries (`docs/findings-contract.md`).

See Also

`ruleset_context_for_child()` for the per-source invariant helper and `sitemap_rulesets()` for the engine axis.

Examples

```
ruleset_context()
ruleset_context(
  submission_channel = "search_console_api",
  property_scope = "https://example.com/",
  authority_evidence = "verified_property_set"
)
```

`ruleset_context_for_child`

Per-source context for a sitemap-index child (ADR-009 §1, §12.1 invariant)

Description

Expresses the per-source invariant that **a sitemap-index child inherits nothing implicitly**: this helper takes **no parent context**, so a child's context can only be established by its own evidence. The defaults encode a child with no submission facts (`submission_channel = "absent"`), found as an index child (`discovery_provenance = "index_child"`), no bound property, and no authority evidence (`authority_evidence = "absent"`). A caller supplies the child's own evidence via the arguments.

Usage

```
ruleset_context_for_child(
  submission_channel = "absent",
  discovery_provenance = "index_child",
  property_scope = NA_character_,
  authority_evidence = "absent"
)
```

Arguments

submission_channel
One of search_console_api, webmaster_tools, user, import, or absent (default). "discovered" is rejected.

discovery_provenance
One of organic (default), robots_txt_reference, supplied, guessed_path, index_child, or archive_derived.

property_scope The verified property/site string (a URL) or NA (default).

authority_evidence
One of absent (default), verified_property_set, target_host_robots_reference, same_location_default, or conflicting.

Details

This is the invariant-respecting context primitive only; it does **not** evaluate scope or authority (§12.2; S4/S5 own those). Because there is no parent parameter, a submitted or verified parent index cannot confer scope or authority on a child by construction.

Value

An object of class `sitemapr_ruleset_context` built solely from the child's own evidence.

See Also

[ruleset_context\(\)](#) for the general constructor.

Examples

```
# A child with no evidence of its own: nothing is inherited from the parent.
ruleset_context_for_child()

# A child that carries its own verified-property evidence.
ruleset_context_for_child(
  property_scope = "https://cdn.example.com/",
  authority_evidence = "verified_property_set"
)
```

ruleset_revision	<i>Published revision of a sitemap_ruleset</i>
------------------	--

Description

Returns the published revision string for a ruleset (ADR-009 §1/§7) so cross-repo consumers can pin a compatible version. The revisions live in one documented constant map (a single source of truth); a ruleset's revision advances only when that ruleset's own rules change.

Usage

```
ruleset_revision(ruleset = sitemap_rulesets())
```

Arguments

ruleset	A single sitemap_ruleset name; see sitemap_rulesets() . Defaults to the baseline "sitemaps.org".
---------	--

Value

A length-1 character revision string.

See Also

[sitemap_rulesets\(\)](#) for the value set.

Examples

```
ruleset_revision()  
ruleset_revision("yandex")
```

sitemap_audit	<i>Assemble a sitemap audit result container</i>
---------------	--

Description

Bundles precomputed sitemapr outputs into a single validated sitemap_audit object with stable, named components. It performs no fetching, parsing, or validation of its own: it *assembles* shapes the package already produces (see [read_sitemap\(\)](#), [validate_sitemap\(\)](#), and [sitemap_tree\(\)](#)) into one first-class result and validates their column contracts.

Usage

```
sitemap_audit(
  urls = NULL,
  findings = NULL,
  sources = NULL,
  problems = NULL,
  tree = NULL
)
```

Arguments

urls	A read_sitemap() tidy row tibble (optionally carrying the sources/problems attributes). Defaults to the empty row schema.
findings	A validate_sitemap() findings tibble. Defaults to the empty findings contract.
sources	The per-source fetch-metadata records. Defaults to the sources attribute of urls when present, otherwise the empty schema.
problems	The non-fatal parse problems companion table. Defaults to the problems attribute of urls when present, else the empty schema.
tree	A sitemap_tree() discovery/index structure. Defaults to the empty tree schema.

Details

Because a [read_sitemap\(\)](#) result already carries its per-source metadata and non-fatal parse problems as the sources and problems attributes, those two components are pulled from urls when not passed explicitly, and then promoted to first-class components (the stored urls component is the plain tidy tibble, without those attributes). Any component left NULL defaults to its empty (zero-row) schema, so `sitemap_audit()` with no arguments is a valid empty audit and any subset of components yields a valid partial audit.

Value

A validated `sitemap_audit` object: a classed list with the components `urls`, `findings`, `sources`, `problems`, and `tree`. Access them with [audit_urls\(\)](#), [audit_findings\(\)](#), [audit_sources\(\)](#), [audit_problems\(\)](#), and [audit_tree\(\)](#).

See Also

[read_sitemap\(\)](#), [validate_sitemap\(\)](#), and [sitemap_tree\(\)](#) for the component producers.

Examples

```
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  '<url><loc>https://example.com/</loc>',
  '<lastmod>2024-01-01</lastmod></url>',
  '<url><loc>https://example.com/about</loc></url>',
  '</urlset>'
)
```

```

)
path <- tempfile(fileext = ".xml")
writeLines(xml, path)

# Assemble precomputed read + validate results into one container.
audit <- sitemap_audit(
  urls = read_sitemap(path),
  findings = validate_sitemap(path, mode = "non-strict")
)
audit
summary(audit)

# An empty audit is valid (all components are zero-row schemas).
sitemap_audit()

```

sitemap_companions	<i>Access the sources and problems companions of a sitemap result</i>
--------------------	---

Description

`sources()` and `problems()` return the two companion tables the package produces alongside its tidy URL rows: the per-source fetch-metadata records and the non-fatal parse problems table. They dispatch on the object type, so the same call works on a `read_sitemap()` result and on a `sitemap_audit()` object:

Usage

```

sources(x, ...)

## Default S3 method:
sources(x, ...)

## S3 method for class 'sitemap_audit'
sources(x, ...)

problems(x, ...)

## Default S3 method:
problems(x, ...)

## S3 method for class 'sitemap_audit'
problems(x, ...)

```

Arguments

<code>x</code>	A <code>read_sitemap()</code> result (a tidy tibble carrying the sources/problems attributes) or a <code>sitemap_audit()</code> object.
<code>...</code>	Ignored; reserved for future methods.

Details

- On a `read_sitemap()` result, they read the `sources` / `problems` attributes the entry point attaches to the tidy tibble, so callers need not reach for `attr()`.
- On a `sitemap_audit()` object, they return the first-class `sources` / `problems` components, delegating to `audit_sources()` / `audit_problems()` for identical behavior.

The default methods return the requested attribute (or `NULL` when the object carries none), so they are safe to call on any object.

Value

For `sources()`, the per-source fetch-metadata records; for `problems()`, the non-fatal parse problems table. A `sitemap_audit()` object always yields the documented (possibly zero-row) component; a `read_sitemap()` result yields its attached companion table.

See Also

`audit_sources()` and `audit_problems()` for the `sitemap_audit` component accessors.

Examples

```
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  '<url><loc>https://example.com/</loc></url></urlset>'
)
path <- tempfile(fileext = ".xml")
writeLines(xml, path)
urls <- read_sitemap(path)

sources(urls)
problems(urls)

# The same accessors work on a sitemap_audit object.
audit <- sitemap_audit(urls = urls)
sources(audit)
problems(audit)
```

sitemap_contract

Published cross-port contract of this sitemapr build

Description

Returns the contract identity that sibling implementations pin against (ADR-009 §7): which findings-contract generation this build speaks, which revision of the finding-code registry it ships, the per-ruleset revisions, and the sibling-package version ranges it is known to work with.

Usage

```
sitemap_contract()
```

Details

The sibling_versions map has the same shape as robotstxt's robots_engine_contract_v1()\$sibling_versions and sitemap-validator's SIBLING_VERSIONS, so all three repos in the contract publish their ranges identically.

Value

- A named list:
- contract_id Length-1 character. Stable identifier for the findings-contract generation, e.g. "sitemapr.findings/v2".
- contract_version Length-1 character. The generation number.
- legacy_contract_version Length-1 character. The older generation still accepted by consumers.
- registry_revision Length-1 character. Date-stamped revision of findings-registry.csv.
- ruleset_revisions Named character. Per-ruleset revisions; the same values ruleset_revision() returns one at a time.
- sibling_versions Named character. Supported version range per sibling package.

See Also

ruleset_revision() for a single ruleset's revision.

Examples

sitemap_contract()\$sibling_versions

sitemap_rulesets	Supported sitemap_ruleset values
------------------	----------------------------------

Description

The engine axis of the ADR-009 validation context: the sitemaps.org baseline plus the opt-in google, bing, and yandex overlays (sitemap-spec §12.0). sitemapr owns this value set and it is extensible; this accessor is the one canonical source vector every ruleset-aware surface validates against. The baseline "sitemaps.org" is deliberately first so it is the default choice.

Usage

sitemap_rulesets()

Value

A character vector of the supported ruleset names, baseline first.

See Also

ruleset_revision() for the published revision of a ruleset and ruleset_context() for the per-source context axes.

Examples

```
sitemap_rulesets()
```

sitemap_tree	<i>Discover a site's sitemaps as a tree</i>
--------------	---

Description

Discovers a site's sitemaps and returns them as a tree: one row per evaluated sitemap, accepted or rejected, with the columns needed to explain the result. Two input modes select what x is:

Usage

```
sitemap_tree(  
  x,  
  from = c("root", "sitemap"),  
  use_robots = TRUE,  
  use_known_paths = TRUE,  
  user_agent = default_user_agent(),  
  limits = discovery_limits(),  
  net_limits = fetch_limits(),  
  index_limits = NULL,  
  policy = request_policy(),  
  max_active = NULL  
)
```

Arguments

x	A single URL (character). With from = "root" a site-root URL (a bare host is accepted and normalized to https:// via the shared site-entripoint policy); with from = "sitemap" an exact sitemap URL.
from	Input mode: "root" (default) treats x as a site root and runs discovery; "sitemap" treats x as an exact sitemap URL and fetches only that.
use_robots	When from = "root", fetch robots.txt and add every Sitemap: directive it lists (provenance "robots"), deduplicated against the guessed paths. Default TRUE. Robots rules (Disallow/Allow) are never applied — only the Sitemap: directive is read.
use_known_paths	When from = "root", try the guessed-path catalog (provenance "guessed-path"). Default TRUE. Set both use_robots and use_known_paths to FALSE for an empty tree.
user_agent	The User-Agent header for HTTP fetches. Defaults to the package User-Agent.
limits	Discovery limits (the candidate cap), as from discovery_limits(). Used only when from = "root".
net_limits	Network limits for the per-candidate fetches, as from fetch_limits().

index_limits	Sitemapindex-expansion bounds (recursion depth and per-index child-count cap), as from <code>index_limits()</code> . Defaults to <code>index_limits()</code> .
policy	A <code>request_policy()</code> applied to every HTTP hop (root, robots.txt, discovery, redirects, and index children) — configure custom headers, authentication, a proxy, TLS options, retry/backoff, and per-host throttling there. Defaults to the no-op policy.
max_active	Optional worker cap for opt-in bounded-concurrency expansion of a sitemap index (ADR-008). NULL (the default) keeps child fetches sequential; a larger value (a small, polite bound such as 4–6 is typical) fetches up to that many independent child sitemaps at once. Concurrency changes only <i>when</i> a child's bytes arrive, never the row order or budget-truncation point — the result is byte-identical to sequential mode.

Details

`from = "root"` (**default**) `x` is a site-root URL. `sitemap_tree()` reads robots.txt Sitemap: directives (provenance "robots") and tries the guessed-path catalog (every generic guess first, then WordPress/Shopify), classifying each candidate. A guessed path that 404s is a rejected not-found row, never a validation finding, and a single unreachable guess never fails the call. Robots rules (Disallow/Allow) are never applied — only the Sitemap: directive is read (ADR-006). Turn either source off with `use_robots` / `use_known_paths`.

`from = "sitemap"` `x` is an exact sitemap or sitemapindex URL. `sitemap_tree()` fetches that one URL (no catalog, no guessing) and, if it is a sitemapindex, expands it. The root row carries provenance "seed". A fetch or parse failure yields a single rejected seed row.

In both modes each accepted candidate's page count and gzip flag come from the body fetched during discovery, so no source is requested twice, and an accepted sitemapindex is expanded recursively (cycle-safe, depth- and count-capped), contributing deeper rows parented to the index with provenance "child-of-index". To discover from sitemap bytes you already fetched yourself (no network for the root), use `sitemap_tree_from_bytes()`.

Value

A tibble with one row per evaluated sitemap (and per expanded index child) and columns `depth`, `parent_sitemap`, `sitemap_url`, `page_count`, `gzip`, `status`, `reason`, and `provenance`. Accepted rows carry a `page_count/gzip`; rejected rows carry their rejection reason and NA page count.

See Also

`sitemap_tree_from_bytes()` to discover from already-fetched bytes.

Examples

```
# Discover a site's sitemaps from its root URL as a tree of candidates.
# sitemap_tree("https://example.com")

# A bare host is accepted and normalized to https://.
# sitemap_tree("example.com")

# Fetch and expand one exact sitemap URL, skipping the guessed-path catalog.
```

```
# sitemap_tree("https://example.com/sitemap_index.xml", from = "sitemap")
```

```
sitemap_tree_from_bytes
```

Discover a site's sitemaps from already-fetched bytes

Description

Builds the same discovery tree as `sitemap_tree()`, but from sitemap bytes you have already fetched yourself rather than from a site root. This is the explicit-seed entry point for sources `sitemap_tree()` cannot fetch directly — for example a site behind a TLS-fingerprint bot wall, whose sitemap you retrieved through a browser or proxy. The root document is parsed in memory (no network request for it); if it is a `sitemapindex`, its child sitemaps are still fetched and expanded over the network, cycle-safe and depth- and count-capped, exactly as `sitemap_tree()` and `read_sitemap()` do.

Usage

```
sitemap_tree_from_bytes(
  bytes,
  source_url,
  user_agent = default_user_agent(),
  net_limits = fetch_limits(),
  index_limits = NULL,
  policy = request_policy()
)
```

Arguments

<code>bytes</code>	The raw sitemap document: a raw vector, or a length-1 character string of the sitemap text. Transparent gzip is inflated automatically.
<code>source_url</code>	The URL the bytes came from (length-1 character). Used as the root row's <code>sitemap_url</code> , as the cycle-detection identity for expansion, and as the parent attribution for expanded children.
<code>user_agent</code>	The User-Agent header for the child fetches an index seed triggers. Defaults to the package User-Agent.
<code>net_limits</code>	Network limits for the child fetches, as from <code>fetch_limits()</code> .
<code>index_limits</code>	Sitemapindex-expansion bounds, as from <code>index_limits()</code> . Defaults to <code>index_limits()</code> .
<code>policy</code>	A request policy applied to every index-child HTTP hop. Defaults to the no-op policy.

Details

A leaf `urlset`/text seed performs no network access at all. The root row carries `provenance = "seed"`; expansion children carry `"child-of-index"`. Content that cannot be parsed yields a single rejected seed row rather than an error.

Value

A tibble with the same 8-column contract as `sitemap_tree()`: `depth`, `parent_sitemap`, `sitemap_url`, `page_count`, `gzip`, `status`, `reason`, and `provenance`.

See Also

`sitemap_tree()` for discovery from a site root or an exact URL.

Examples

```
# Parse an already-fetched leaf sitemap with no network access.
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  "<url><loc>https://example.com/</loc></url>",
  "<url><loc>https://example.com/about</loc></url>",
  "</urlset>"
)
sitemap_tree_from_bytes(xml, source_url = "https://example.com/sitemap.xml")
```

<code>validate_sitemap</code>	<i>Validate a sitemap source against the schema, protocol, and classification rules</i>
-------------------------------	---

Description

The public validation entry point. Reads one or more sitemap sources — sitemap URLs or local sitemap files — runs every finding-producer over them (the XSD schema layer, the protocol/semantic layer, the byte-level classification layer, and, for a sitemap index, the bounded index-expansion layer), and assembles the results into the stable findings contract.

Usage

```
validate_sitemap(
  x,
  mode = c("strict", "non-strict"),
  user_agent = default_user_agent(),
  limits = fetch_limits(),
  index_limits = NULL,
  policy = request_policy(),
  check_robots = FALSE,
  robots_user_agent = "*",
  inspect_pages = FALSE,
  page_sample = 50L,
  page_mode = c("sample", "full"),
  page_budget = page_inspection_budget(),
  page_user_agent = default_user_agent()
)
```

```

validate_sitemaps(
  x,
  mode = c("strict", "non-strict"),
  user_agent = default_user_agent(),
  limits = fetch_limits(),
  index_limits = NULL,
  policy = request_policy(),
  check_robots = FALSE,
  robots_user_agent = "*",
  inspect_pages = FALSE,
  page_sample = 50L,
  page_mode = c("sample", "full"),
  page_budget = page_inspection_budget(),
  page_user_agent = default_user_agent()
)

```

Arguments

x	One or more sitemap URLs or paths to local sitemap files (.xml, .txt, .gz, or .tar.gz).
mode	"strict" (the default) or "non-strict". In non-strict, strict-only findings are dropped and schema violations are downgraded to warning; in strict, the documented info-to-warning codes are elevated.
user_agent	The User-Agent header for HTTP fetches. Defaults to the package User-Agent.
limits	Network limits for HTTP fetches, as from <code>fetch_limits()</code> .
index_limits	Sitemapindex-expansion bounds (recursion depth and per-index child-count cap), as from <code>index_limits()</code> . Defaults to <code>index_limits()</code> .
policy	A request_policy() applied to every HTTP hop (root, robots.txt, discovery, redirects, and index children) — configure custom headers, authentication, a proxy, TLS options, retry/backoff, and per-host throttling there. Defaults to the no-op policy.
check_robots	Logical; when TRUE, run the robots.txt allow/disallow check over the advertised URLs and the sitemap document itself (requires the optional <code>robotstxttr</code> package). Defaults to FALSE.
robots_user_agent	The robots.txt group to match against when <code>check_robots = TRUE</code> , e.g. "*" (the catch-all group, the default) or a specific crawler token such as "Googlebot".
inspect_pages	Logical; the master opt-in for per-URL page inspection (Layer E). When FALSE (the default) no page is fetched and the result is byte-identical to a call without it: the pinned ten-column findings surface and no <code>page_coverage</code> attribute. When TRUE, a budgeted, deduplicated, deterministically-sampled set of the advertised page URLs is fetched and each fetch's transport outcome maps to at most one page-layer finding (<code>PAGE_STATUS_ERROR</code> , <code>PAGE_STATUS_REDIRECT</code> , <code>PAGE_REDIRECT_CHAIN</code> , <code>PAGE_FETCH_FAILED</code> , <code>PAGE_SSRF_BLOCKED</code>); the run's coverage rides the <code>page_coverage</code> attribute (see Value). Network expansion is never implicit. Page inspection is batch-wide: one budget over the union of the call's deduped page URLs.

page_sample	Integer sample size for page_mode = "sample": how many of the deduplicated page URLs to inspect, chosen by a deterministic stable hash order so re-runs pick the same set. Ignored when page_mode = "full".
page_mode	"sample" (inspect page_sample deduplicated URLs, the default) or "full" (inspect every deduplicated URL, subject to the budget caps).
page_budget	A page-inspection budget list: the aggregate caps (max pages, max requests/hops, max aggregate bytes, per-page body cap, max wall time), each caller-overridable with a safe default. Applies only when inspect_pages = TRUE.
page_user_agent	The HTTP request User-Agent sent when fetching pages (recorded for the "what did the inspector see" caveat; distinct from a robots product token). Defaults to sitemapr's inspector UA.

Details

The source is read once and branched on its sniffed format: an HTML document served where a sitemap was expected yields an `UNSUPPORTED_HTML_MASQUERADE` classification finding; a plain-text sitemap is checked line-by-line; an RSS 2.0 or Atom 0.3/1.0 feed is parsed into rows and protocol-validated; an XML document is dispatched on its root element. An XML root that is neither `urlset` nor `sitemapindex` yields an `UNSUPPORTED_ROOT` finding rather than an error. A `urlset` is schema- and protocol-validated; a `sitemapindex` is schema-validated and recursively expanded (cycle-, depth-, and count-capped), with the traversal events surfaced as `INDEX_*` findings.

When `x` contains more than one source, inputs are normalized, deduplicated, and capped using the submitted-list source-record policy. Per-source failures are returned as fetch-layer findings and successful sources still contribute their findings. Scalar calls keep the stricter historical behavior: genuine transport, SSRF, or HTTP failures raise classed conditions.

When `check_robots = TRUE`, each sitemap-advertised URL is additionally tested against its governing `robots.txt` (via the optional `robotstxttr` engine), emitting robots-layer findings for URLs that are disallowed (`ROBOTS_DISALLOWED`, warning) or that cannot be decided because `robots.txt` would not fetch (`ROBOTS_INDETERMINATE`, info). Each distinct origin's `robots.txt` is fetched once under the SSRF-guarded fetch policy; matching is offline, so every advertised URL is checked with no sampling. The sitemap document's own URL is tested too: a sitemap published at a path its own `robots.txt` disallows yields `ROBOTS_SITEMAP_DISALLOWED` (warning, scoped to the source). When `robotstxttr` is not installed, a classed warning naming the install command is signaled and the check is skipped; every other layer is unaffected.

Value

The findings tibble described in `docs/findings-contract.md`: the columns `code`, `severity`, `layer`, `subject_type`, `subject_ref`, `message`, `evidence`, `mode`, `is_strict_only`, and `remediation_hint`, in the contract's stable order. The same source and mode yield a row-for-row identical tibble across calls. A genuine transport, SSRF, or HTTP failure raises a classed error condition. When `inspect_pages = TRUE`, the tibble additionally carries a `page_coverage` attribute (`attr(x, "page_coverage")`) — a versioned, batch-wide named list reporting what the run covered (eligible, deduplicated, selected, attempted, completed, partial, and which caps bit) so a sampled or capped run is never misread as clean; it is absent when `inspect_pages = FALSE`. The tibble may also carry a `layers_run` attribute (`attr(x, "layers_run")`) — a character vector naming validation layers this run exercised whose execution the result cannot otherwise evidence, so that `report_sitemap()`

can tell "ran and found nothing" from "never ran". It records "robots" for `check_robots = TRUE` (a clean robots run emits no findings) and "schema" when XSD validation ran on the parsed document. It is a run manifest and not part of the row contract: treat it as advisory, present only when there was something to record.

Examples

```
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  '<url><loc>https://example.com/</loc>',
  '<priority>2.0</priority></url>',
  '</urlset>'
)
path <- tempfile(fileext = ".xml")
writeLines(xml, path)
validate_sitemap(path, mode = "non-strict")

# Validate directly from a sitemap URL.
# validate_sitemap("https://example.com/sitemap.xml")
```

validate_sitemap_robots

Validate a sitemap under an explicit robots context (sitemap-spec §13.0)

Description

The robots-aware entry point parallel to `validate_sitemap()`. It runs the identical validation pipeline with the robots allow/disallow layer **on**, evaluated under a caller-supplied `robots_context()` rather than the bare matcher user-agent `validate_sitemap()` accepts, and appends the context to the result as a `robots_context` list-column.

Usage

```
validate_sitemap_robots(
  x,
  context = robots_context(),
  mode = c("strict", "non-strict"),
  user_agent = default_user_agent(),
  limits = fetch_limits(),
  index_limits = NULL,
  policy = request_policy(),
  inspect_pages = FALSE,
  page_sample = 50L,
  page_mode = c("sample", "full"),
  page_budget = page_inspection_budget(),
  page_user_agent = default_user_agent()
)
```

```

validate_sitemaps_robots(
  x,
  context = robots_context(),
  mode = c("strict", "non-strict"),
  user_agent = default_user_agent(),
  limits = fetch_limits(),
  index_limits = NULL,
  policy = request_policy(),
  inspect_pages = FALSE,
  page_sample = 50L,
  page_mode = c("sample", "full"),
  page_budget = page_inspection_budget(),
  page_user_agent = default_user_agent()
)

```

Arguments

x	One or more sitemap URLs or paths to local sitemap files (.xml, .txt, .gz, or .tar.gz).
context	A robots evaluation context from robots_context() or robots_context_preset() (the three ADR-009 §1 robots axes). Defaults to robots_context() , the Google-default widening <code>validate_sitemap()</code> has always applied. Carried into the <code>robots_context</code> list-column of the result.
mode	"strict" (the default) or "non-strict". In non-strict, strict-only findings are dropped and schema violations are downgraded to warning; in strict, the documented info-to-warning codes are elevated.
user_agent	The User-Agent header for HTTP fetches. Defaults to the package User-Agent.
limits	Network limits for HTTP fetches, as from fetch_limits() .
index_limits	Sitemapindex-expansion bounds (recursion depth and per-index child-count cap), as from index_limits() . Defaults to index_limits() .
policy	A request_policy() applied to every HTTP hop (root, robots.txt, discovery, redirects, and index children) — configure custom headers, authentication, a proxy, TLS options, retry/backoff, and per-host throttling there. Defaults to the no-op policy.
inspect_pages	Logical; the master opt-in for per-URL page inspection (Layer E). When FALSE (the default) no page is fetched and the result is byte-identical to a call without it: the pinned ten-column findings surface and no <code>page_coverage</code> attribute. When TRUE, a budgeted, deduplicated, deterministically-sampled set of the advertised page URLs is fetched and each fetch's transport outcome maps to at most one page-layer finding (<code>PAGE_STATUS_ERROR</code> , <code>PAGE_STATUS_REDIRECT</code> , <code>PAGE_REDIRECT_CHAIN</code> , <code>PAGE_FETCH_FAILED</code> , <code>PAGE_SSRF_BLOCKED</code>); the run's coverage rides the <code>page_coverage</code> attribute (see Value). Network expansion is never implicit. Page inspection is batch-wide: one budget over the union of the call's deduped page URLs.

page_sample	Integer sample size for page_mode = "sample": how many of the deduplicated page URLs to inspect, chosen by a deterministic stable hash order so re-runs pick the same set. Ignored when page_mode = "full".
page_mode	"sample" (inspect page_sample deduplicated URLs, the default) or "full" (inspect every deduplicated URL, subject to the budget caps).
page_budget	A page-inspection budget list: the aggregate caps (max pages, max requests/hops, max aggregate bytes, per-page body cap, max wall time), each caller-overridable with a safe default. Applies only when inspect_pages = TRUE.
page_user_agent	The HTTP request User-Agent sent when fetching pages (recorded for the "what did the inspector see" caveat; distinct from a robots product token). Defaults to sitemapr's inspector UA.

Details

Use it when the robots question is "would *this engine* fetch these URLs". `validate_sitemap()` carries only `robots_user_agent`, a matcher user-agent string that widens onto the Google policy and matcher; this entry point carries all three axes, so the `ROBOTS_DISALLOWED` / `ROBOTS_INDETERMINATE` / `ROBOTS_SITEMAP_DISALLOWED` findings are decided by the selected engine's own status policy and matcher. Under the default `robots_context()` the result is exactly `validate_sitemap(check_robots = TRUE)`, plus the added column.

The robots axes are independent of the `sitemap_ruleset` axis (ADR-009 §1): this entry point selects an engine's **robots** semantics and does not select a sitemap ruleset. It returns the baseline schema-v1 result, so the additive per-engine ruleset columns of `validate_sitemap_ruleset()` are not present. To select both axes, call `validate_sitemap_ruleset()` with its `robots_context` argument; independence means the axes are chosen separately, not that they cannot be chosen together. This entry point remains the shorthand for the robots axis alone, and is exactly `validate_sitemap_ruleset(x, "sitemaps.org", robots_context = context)`.

A backend the installed `robotstxttr` reports as `capability_unavailable` decides nothing rather than guessing: every advertised URL comes back as `ROBOTS_INDETERMINATE`. The same is true of a product token a bounded matcher backend does not accept — prefer `robots_context_preset()`, whose tokens are known-good for their backend.

Value

The findings tibble of `validate_sitemap()` — the pinned ten columns — plus a `robots_context` list-column of the context as a named list, appended last. The same source, mode and context yield a row-for-row identical tibble across calls.

See Also

`robots_context()` and `robots_context_preset()` for the carrier, `validate_sitemap()` for the baseline entry point, and `validate_sitemap_ruleset()` for the independent sitemap-ruleset axis.

Examples

```
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
```

```
'<url><loc>https://example.com/</loc></url>',
'</urlset>'
)
path <- tempfile(fileext = ".xml")
writeLines(xml, path)

# A local file advertises a remote URL, so this reaches the network only
# when `robotstxttr` is installed; without it the layer warns and skips.
# validate_sitemap_robots(path, robots_context_preset("yandex"))
```

validate_sitemap_ruleset

Validate a sitemap under an engine-aware ruleset (ADR-009)

Description

The versioned, engine-aware entry point parallel to `validate_sitemap()`. It runs the identical validation pipeline (the XSD schema, protocol/semantic, byte-level classification, and bounded index-expansion layers), then, when an engine overlay is selected, augments the findings tibble with the additive schema-v2 columns (docs/decisions/ADR-009-per-engine-validation-profiles.md §5/§6, docs/findings-contract.md "Per-engine ruleset extension").

Usage

```
validate_sitemap_ruleset(
  x,
  sitemap_ruleset = sitemap_rulesets(),
  context = ruleset_context(),
  mode = c("strict", "non-strict"),
  user_agent = default_user_agent(),
  limits = fetch_limits(),
  index_limits = NULL,
  policy = request_policy(),
  check_robots = FALSE,
  robots_user_agent = "*",
  robots_context = NULL,
  inspect_pages = FALSE,
  page_sample = 50L,
  page_mode = c("sample", "full"),
  page_budget = page_inspection_budget(),
  page_user_agent = default_user_agent()
)

validate_sitemaps_ruleset(
  x,
  sitemap_ruleset = sitemap_rulesets(),
  context = ruleset_context(),
  mode = c("strict", "non-strict"),
```

```

    user_agent = default_user_agent(),
    limits = fetch_limits(),
    index_limits = NULL,
    policy = request_policy(),
    check_robots = FALSE,
    robots_user_agent = "*",
    robots_context = NULL,
    inspect_pages = FALSE,
    page_sample = 50L,
    page_mode = c("sample", "full"),
    page_budget = page_inspection_budget(),
    page_user_agent = default_user_agent()
)

```

Arguments

x	One or more sitemap URLs or paths to local sitemap files (.xml, .txt, .gz, or .tar.gz).
sitemap_ruleset	The engine ruleset to validate under; one of <code>sitemap_rulesets()</code> (baseline "sitemaps.org" first, the default). The baseline emits the schema-v1 result; an engine overlay adds the additive columns.
context	A per-source validation context from <code>ruleset_context()</code> (the four independent ADR-009 §1 axes). Carried into the context list-column of the additive result. Ignored on the baseline path (which emits no additive columns). This is the RULESET context; the robots context is the separate <code>robots_context</code> argument, and passing one where the other belongs is rejected rather than silently accepted.
mode	"strict" (the default) or "non-strict". In non-strict, strict-only findings are dropped and schema violations are downgraded to warning; in strict, the documented info-to-warning codes are elevated.
user_agent	The User-Agent header for HTTP fetches. Defaults to the package User-Agent.
limits	Network limits for HTTP fetches, as from <code>fetch_limits()</code> .
index_limits	Sitemapindex-expansion bounds (recursion depth and per-index child-count cap), as from <code>index_limits()</code> . Defaults to <code>index_limits()</code> .
policy	A <code>request_policy()</code> applied to every HTTP hop (root, robots.txt, discovery, redirects, and index children) — configure custom headers, authentication, a proxy, TLS options, retry/backoff, and per-host throttling there. Defaults to the no-op policy.
check_robots	Logical; when TRUE, run the robots.txt allow/disallow check over the advertised URLs and the sitemap document itself (requires the optional <code>robotstxttr</code> package). Defaults to FALSE.
robots_user_agent	The robots.txt group to match against when <code>check_robots = TRUE</code> , e.g. "*" (the catch-all group, the default) or a specific crawler token such as "Googlebot".

robots_context	A robots evaluation context from <code>robots_context()</code> or <code>robots_context_preset()</code> , or NULL (the default) to leave the robots axis as <code>validate_sitemap()</code> treats it. Supplying one runs the robots layer by construction — as in <code>validate_sitemap_robots()</code> , the context IS the request, so <code>check_robots</code> need not also be set — and appends the <code>robots_context</code> list-column. It cannot be combined with a non-default <code>robots_user_agent</code> : that argument is the string shorthand for the same axis, and honoring both would make it ambiguous which one decided the findings.
inspect_pages	Logical; the master opt-in for per-URL page inspection (Layer E). When FALSE (the default) no page is fetched and the result is byte-identical to a call without it: the pinned ten-column findings surface and no <code>page_coverage</code> attribute. When TRUE, a budgeted, deduplicated, deterministically-sampled set of the advertised page URLs is fetched and each fetch's transport outcome maps to at most one page-layer finding (<code>PAGE_STATUS_ERROR</code> , <code>PAGE_STATUS_REDIRECT</code> , <code>PAGE_REDIRECT_CHAIN</code> , <code>PAGE_FETCH_FAILED</code> , <code>PAGE_SSRF_BLOCKED</code>); the run's coverage rides the <code>page_coverage</code> attribute (see Value). Network expansion is never implicit. Page inspection is batch-wide: one budget over the union of the call's deduped page URLs.
page_sample	Integer sample size for <code>page_mode = "sample"</code> : how many of the deduplicated page URLs to inspect, chosen by a deterministic stable hash order so re-runs pick the same set. Ignored when <code>page_mode = "full"</code> .
page_mode	"sample" (inspect <code>page_sample</code> deduplicated URLs, the default) or "full" (inspect every deduplicated URL, subject to the budget caps).
page_budget	A page-inspection budget list: the aggregate caps (max pages, max requests/hops, max aggregate bytes, per-page body cap, max wall time), each caller-overridable with a safe default. Applies only when <code>inspect_pages = TRUE</code> .
page_user_agent	The HTTP request User-Agent sent when fetching pages (recorded for the "what did the inspector see" caveat; distinct from a robots product token). Defaults to <code>sitemapr's</code> inspector UA.

Details

Backwards compatibility is preserved by construction (ADR-009 §5): a baseline call (`sitemap_ruleset = "sitemaps.org"`, the default) returns exactly the pinned ten-column schema-v1 result, byte-identical to `validate_sitemap()`. The four additive columns appear **only** for an explicit engine overlay ("google" / "bing" / "yandex"); there is deliberately no `profile=` argument on `validate_sitemap()` and no silent default switch to an engine.

This slice fixes the engine-aware carrier and the additive schema; no per-engine evaluators exist yet, so every finding produced under an overlay is a reused baseline code and carries provenance = "inherited_protocol" (an ADR-009 §0 executable class). Later slices override the provenance per code.

Per-URL page inspection (`inspect_pages = TRUE`) works exactly as in `validate_sitemap()`, except the transport / canonical / hreflang page findings are assembled under the selected `sitemap_ruleset` too: under an engine overlay they carry the same additive schema-v2 columns as the base findings, so the per-engine provenance / context (ADR-009 §5.2/§5.3) engages over the page layer rather than emitting as generic baseline diagnostics. As in `validate_sitemap()`, `inspect_pages = FALSE` is byte-identical to a call without the argument.

This is the engine-aware entry point, and it carries **both** engine-aware axes: pass `robots_context =` to select an engine's robots semantics in the same call (`SITE-otfmeqyx`). The two axes stay independent as ADR-009 §1 requires — neither is derived from the other, so a Bing sitemap rule-set with a Yandex robots context is a legal and honored pair — but they are no longer mutually exclusive, and a combined call runs the pipeline once rather than fetching everything twice. Each axis governs its own columns: the additive ruleset columns appear only under an engine overlay, and `robots_context` appears only when an explicit robots context is supplied. Use `validate_sitemap_robots()` when you want the robots axis alone.

Value

The findings tibble of `validate_sitemap()`. Under the baseline `sitemap_ruleset` it is exactly the pinned ten columns; under an engine overlay it additionally carries `ruleset` (character), `ruleset_revision` (character), `context` (a list-column of the context object as a named list), and `provenance` (character, per finding), appended in that order after the ten pinned columns. The same source, mode, rule-set, and context yield a row-for-row identical tibble across calls. An engine overlay also stamps a `ruleset_run` attribute (`attr(x, "ruleset_run")`) naming the selected ruleset, so `report_sitemap()` can tell an engine-gated check that ran and passed from one this call could never reach — the `ruleset` column answers that per row, and a clean run has no rows. Like `layers_run` it is a run manifest and not part of the row contract; the baseline path stamps nothing. Supplying `robots_context` appends one further list-column, `robots_context`, last — after the additive rule-set columns when both axes are selected. Because each axis governs its own columns, a baseline call carrying a robots context returns exactly `validate_sitemap_robots()`'s result for the same context.

See Also

`validate_sitemap()` for the baseline entry point, `validate_sitemap_robots()` for the robots axis alone, `sitemap_rulesets()` for the ruleset value set, `ruleset_context()` for the per-source context axes, and `robots_context()` for the robots axes.

Examples

```
xml <- paste0(
  '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">',
  '<url><loc>https://example.com/</loc></url>',
  '</urlset>'
)
path <- tempfile(fileext = ".xml")
writeLines(xml, path)

# Baseline: identical to validate_sitemap().
validate_sitemap_ruleset(path, "sitemaps.org")

# Engine overlay: adds the additive schema-v2 columns.
validate_sitemap_ruleset(path, "google")

# Both engine-aware axes in one call, independently chosen.
# validate_sitemap_ruleset(
#   path, "bing", robots_context = robots_context_preset("yandex")
# )
```

Index

- * **request-policy**
 - request_auth_basic, 18
 - request_auth_bearer, 19
 - request_policy, 19
 - request_proxy, 21
 - request_retry, 22
 - request_throttle, 23
- audit_accessors, 3
- audit_findings (audit_accessors), 3
- audit_findings(), 5, 32
- audit_problems (audit_accessors), 3
- audit_problems(), 5, 32, 34
- audit_sitemap, 4
- audit_sitemap(), 16
- audit_sources (audit_accessors), 3
- audit_sources(), 5, 32, 34
- audit_tree (audit_accessors), 3
- audit_tree(), 5, 32
- audit_unchanged, 6
- audit_unchanged(), 7, 8
- audit_urls (audit_accessors), 3
- audit_urls(), 5, 32
- compare_sitemap_audits, 7
- compare_sitemap_audits(), 6
- discovery_limits, 8
- discovery_limits(), 10, 12
- fetch_limits, 9
- fetch_limits(), 9, 12
- gsc_submission, 10
- htmltools::HTML, 16, 17
- index_limits, 11
- index_limits(), 9, 10
- print.sitemap_audit_diff
 - (compare_sitemap_audits), 7
- probe_url, 12
- problems (sitemap_companions), 33
- read_sitemap, 14
- read_sitemap(), 3–5, 10, 12, 13, 16, 17, 31–34, 38
- read_sitemaps (read_sitemap), 14
- report_sitemap, 15
- report_sitemap(), 5, 41, 48
- request_auth_basic, 18
- request_auth_basic(), 19–24
- request_auth_bearer, 19
- request_auth_bearer(), 18, 20–24
- request_policy, 19
- request_policy(), 4, 14, 17–19, 22–24, 37, 40, 43, 46
- request_proxy, 21
- request_proxy(), 18–21, 23, 24
- request_retry, 22
- request_retry(), 18–22, 24
- request_throttle, 23
- request_throttle(), 18–23
- robots_context, 24
- robots_context(), 26, 27, 42–44, 47, 48
- robots_context_preset, 26
- robots_context_preset(), 25, 27, 43, 44, 47
- robots_context_presets, 27
- robots_context_presets(), 26
- robots_cross_submission, 27
- ruleset_context, 28
- ruleset_context(), 10, 24, 25, 27, 28, 30, 35, 46, 48
- ruleset_context_for_child, 29
- ruleset_context_for_child(), 29
- ruleset_revision, 31
- ruleset_revision(), 35
- sitemap_audit, 31
- sitemap_audit(), 3–5, 7, 8, 33, 34

- sitemap_companions, 33
- sitemap_contract, 34
- sitemap_rulesets, 35
- sitemap_rulesets(), 29, 31, 46, 48
- sitemap_tree, 36
- sitemap_tree(), 3, 9, 13, 31, 32, 38, 39
- sitemap_tree_from_bytes, 38
- sitemap_tree_from_bytes(), 37
- sources(sitemap_companions), 33
- summary.sitemap_audit_diff
 - (compare_sitemap_audits), 7

- validate_sitemap, 39
- validate_sitemap(), 3–5, 16, 17, 31, 32, 42, 44, 45, 47, 48
- validate_sitemap_robots, 42
- validate_sitemap_robots(), 24–26, 47, 48
- validate_sitemap_ruleset, 45
- validate_sitemap_ruleset(), 17, 44
- validate_sitemaps(validate_sitemap), 39
- validate_sitemaps_robots
 - (validate_sitemap_robots), 42
- validate_sitemaps_ruleset
 - (validate_sitemap_ruleset), 45