Package 'rurl'

September 9, 2026

Type Package

Title Parse, Clean, and Normalize URLs

Version 3.0.1

Language en-US

Description A lightweight toolkit for extracting structured information from URLs.
Includes functions for parsing, normalizing protocols, extracting domains, and constructing clean URLs.
Domain and public-suffix extraction is delegated to the 'pslr' package,
which implements the Public Suffix List from <https://publicsuffix.org>.
Punycode and IDNA encoding is handled by the 'punycoder' package.

License MIT + file LICENSE

Encoding UTF-8

Collate 'rurl-package.R' 'status-constants.R' 'utils.R'
'percent-coding.R' 'parse-state.R' 'query-denylist.R'
'domain.R' 'path-query.R' 'parse-web.R' 'parse-phases.R'
'parse.R' 'verdicts.R' 'profiles.R' 'diagnostics.R'
'accessors.R' 'email-diagnostics.R' 'host-policy.R'
'scheme-policy.R' 'canonical_join.R' 'resolve.R' 'serialize.R'
'format.R' 'url-key.R' 'url-join.R' 'zzz.R'

Imports utils, stringi, punycoder (>= 1.2.1), pslr (>= 1.1.1)

URL https://bart-turczynski.gitlab.io/rurl/,
https://gitlab.com/bart-turczynski/rurl,
https://CRAN.R-project.org/package=rurl

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

X-schema.org-keywords url-parsing, url-parser, url-normalization,
url-cleaning, url-cleaner, url-checker, domain-extraction,
domain-name-detection, domain-name-checker, public-suffix, tld,
tlds, tld-extraction, tld-checker, tld-verification, idna,
punycode, uri, url, seo, web-scraping, r, rstats, r-stats,
r-package

Suggests testthat (>= 3.0.0), knitr, rmarkdown, withr, pkgdown,
oysteR, rosv, jsonlite

Config/testthat/edition 3

Depends R (>= 4.0.0)

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Bart Turczynski [aut, cre] (ORCID:
<https://orcid.org/0000-0002-8788-7980>)

Maintainer Bart Turczynski <bartek@turczynski.pl>

Repository CRAN

Date/Publication 2026-09-09 07:40:02 UTC

Contents

`canonical_join` *Canonical Join of Two URL Sets (Base R Version)*

Description

Performs a join between two data frames by canonicalizing URLs to a shared "clean" format using `safe_parse_urls` and then matching on that key. This is suitable for large crawl exports.

Usage

```
canonical_join(
  data_A,
  data_B,
  col_A = "URL",
  col_B = "URL",
  suffix_A = "_A",
  suffix_B = "_B",
  name_A = NULL,
  name_B = NULL,
  join = c("inner", "left", "right", "full"),
  collision = c("first", "all", "error"),
  on_parse_error = c("keep", "drop", "error"),
  join_parse_status = c("ok", "ok_or_warning"),
  ...
)
```

Arguments

`data_A`	A data frame containing URLs for the left side of the join.
`data_B`	A data frame containing URLs for the right side of the join.
`col_A`	Character string, the name of the column in `data_A` that contains URLs. Defaults to "URL".
`col_B`	Character string, the name of the column in `data_B` that contains URLs. Defaults to "URL".
`suffix_A`	Character string, suffix to append to `data_A` columns (excluding the URL column) in the output. Defaults to "_A".
`suffix_B`	Character string, suffix to append to `data_B` columns (excluding the URL column) in the output. Defaults to "_B".

name_A — Character string, the name of the output column holding the original data_A URLs. Defaults to NULL, in which case the name is derived from the data_A argument expression via deparse(substitute()). Supply an explicit value for stable output names when piping or passing anonymous inputs (e.g. canonical_join(df[df$x > 1,], get_b())).

name_B — Character string, the name of the output column holding the original data_B URLs. Defaults to NULL; behaves like name_A for data_B.

join — Join type: "inner", "left", "right", or "full". Defaults to "inner".

collision — How to handle duplicate canonical keys within inputs. "first" keeps the first row per key, "all" keeps all rows (many-to-many), and "error" stops on duplicates. Defaults to "first".

on_parse_error — How to handle URLs that fail canonicalization. "keep" retains them as unmatched rows (for left/right/full joins), "drop" removes them before joining, and "error" stops. Defaults to "keep".

join_parse_status — Which parse statuses yield joinable canonical keys. "ok" (default) joins only rows whose parse_status begins with "ok" ("ok", "ok-ftp", "ok-scheme-relative"). "ok_or_warning" additionally treats parseable-but-suspicious warning-* statuses ("warning-no-tld", "warning-invalid-tld", "warning-public-suffix") as joinable. Joining on warning statuses can increase false-positive matches between distinct hosts that both fail TLD derivation.

... — Additional arguments forwarded to safe_parse_urls, controlling canonicalization (e.g., protocol_handling, www_handling, trailing_slash_handling, index_page_handling, path_normalization, scheme_relative_handling, host_encoding, path_encoding, the url_standard selector, and the profile bundle). When url_standard is set, forwarding a governed low-level knob it would override (e.g. path_normalization) is an error, exactly as in safe_parse_url; the orthogonal path_encoding and host_encoding presentation knobs layer freely on any profile. A profile (e.g. "seo", "whatwg") may also be forwarded: like safe_parse_url, it bundles several knobs, expands only into knobs you did not supply, and an explicit knob always overrides it (so the url_standard conflict check is skipped on the profile path). Inspect a bundle with url_profile. See "Legacy presentation dials" below for the arguments that warn.

Value

A data frame representing the join. The output includes:

- The original URL columns (named via name_A / name_B, or after the input expressions when those are NULL).
- JoinKey: the canonicalized URL used for matching.
- All other columns from data_A and data_B with suffixes applied.

Returns an empty data frame with the expected structure if no matches are found or if inputs are invalid.

Legacy presentation dials

canonical_join() keys the join on the cleaned presentation string (clean_url), so every cleaning or display argument forwarded through ... currently changes *which rows match*. Those arguments do not participate in URL identity; they are legacy behavior retained for a deprecation window. Supplying any of protocol_handling, www_handling, source, tld_source, case_handling, trailing_slash_handling, index_page_handling, path_normalization, subdomain_levels_to_keep, host_encoding, path_encoding, port_handling, engine, profile, or any query cleaning dial (query_handling, params_keep, params_drop, params_case_sensitive, sort_params, empty_param_handling, decode_plus) emits one warning per call, of class "rurl_legacy_join_dial_warning". Results are unchanged: the warning is purely additive, so no caller is silently re-matched.

The input and interpretation arguments url_standard, scheme_acceptance, scheme_policy, and scheme_relative_handling are legitimate inputs to identity and never warn.

Because the condition is classed, it can be silenced selectively without hiding other warnings: suppressWarnings(canonical_join(A, B, www_handling = "strip"),classes = "rurl_legacy_join_dial_warning

Examples

```
A <- data.frame(
  URL = c("https://Example.com/page", "https://example.com/other"),
  ValA = 1:2, stringsAsFactors = FALSE
)
B <- data.frame(
  URL = c(
    "https://example.com/page?utm_source=nl",
    "https://example.com/missing"
  ),
  ValB = c("x", "y"), stringsAsFactors = FALSE
)

# Default canonicalization lower-cases the host and drops the query, so the
# first row of each side shares one canonical key.
canonical_join(A, B)
```

---

check_hosts *Report practical host validation verdicts for a set of URLs*

Description

Tabular companion to is_valid_host: for each URL, reports the parsed host, its get_host_type classification, a logical column for each requested policy rule, and a reasons list-column naming the host facts observed. Useful for auditing a URL set before choosing what to keep — see *why* a host is not a practical web/SEO host, not just that it is not.

Usage

```
check_hosts(
  url,
```

```
  rules = c("url", "dns", "web", "registrable", "seo"),
  url_standard = "whatwg",
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general")
)
```

Arguments

`url`	A character vector of URLs.
`rules`	A character vector of one or more rules to score, drawn from `"url"`, `"dns"`, `"web"`, `"registrable"`, `"seo"` (all five by default). See `is_valid_host` for each rule's meaning.
`url_standard`	Standard profile governing host interpretation: `"whatwg"` (default) or `"rfc3986"`.
`scheme_policy`	Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer". • "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior. • "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.
`scheme_acceptance`	Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web". • "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior. • "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority. Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`)

is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

Details

Like `is_valid_host` this is a **policy** layer, not parser conformance and not a conformance oracle: it never changes how a URL parses and the absence of a `reasons` token is not a validity guarantee (see `is_valid_host`'s "A policy layer" section).

Value

A `data.frame` with one row per input URL (input order preserved): `url`, `host` (the parsed host, or `NA`), `host_type`, one logical column per requested rule (`NA` when the URL has no host to judge; the `"url"` rule's column is named `url_valid` so it does not shadow the input `url` column), and `reasons` — a list-column whose i-th element is a character vector of the host facts observed for that URL (`character(0)` when none). The `reasons` evidence is reported for the row as a whole, independent of which `rules` were requested, and combines the host-shape diagnostics that fired (see `get_url_diagnostics`) with the policy tokens `"ip-literal"`, `"not-registrable"`, and `"underscore-label"`.

See Also

`is_valid_host`, `get_host_type`, `get_url_diagnostics`.

Examples

```
urls <- c(
  "http://example.com",         # registrable domain
  "http://_dmarc.example.com",  # valid DNS owner name, not a web hostname
  "http://a+b.example",         # valid RFC reg-name, neither web nor dns
  "http://-example.com",        # hyphen hygiene failure
  "http://a..com",              # empty label
  "http://localhost",           # web hostname, but not registrable
  "http://192.168.0.1",         # IP literal
  "http://xn--nxasmq6b.example.com" # IDN (A-label)
)
check_hosts(urls)
# Score a single rule, or a subset:
check_hosts(urls, rules = c("web", "dns"))
# RFC 3986 reg-name semantics instead of whatwg:
check_hosts("http://2130706433", url_standard = "rfc3986")
```

`check_schemes` *Report scheme facts for a set of URLs, and score them against an allowlist*

Description

Tabular companion to get_scheme and get_scheme_class, and the scheme-axis counterpart of check_hosts: for each URL, reports the parsed scheme, its special/non-special classification, whether it falls inside the built-in web-acceptance set, an optional allowed column scored against a caller-supplied allowlist, and a reasons list-column naming the scheme facts observed.

Usage

```
check_schemes(
  url,
  allowed_schemes = NULL,
  url_standard = "whatwg",
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("general", "web")
)
```

Arguments

url	A character vector of URLs.
allowed_schemes	Optional character vector of schemes the caller will act on, compared case-insensitively. When supplied, the result gains a logical allowed column (FALSE for a URL with no parsed scheme) and the token "not-in-allowlist" where it is FALSE. When NULL (default) no allowed column is emitted and no allowlist judgement is made.
url_standard	Standard profile governing scheme interpretation: "whatwg" (default) or "rfc3986". Unlike most of the package this argument has a non-NULL default, because a scheme classification is undefined without a named standard.
scheme_policy	Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from protocol_handling, which only controls how the scheme is *presented*, and from url_standard, which controls interpretation). Defaults to "infer". • "infer": (Default) Fabricate http:// for scheme-less host-shaped input (e.g. example.com parses as http://example.com), a browser-omnibox-style affordance. This is the historical behavior. • "require": Reject scheme-less input — a scheme-less host-shaped value becomes parse_status = "error" rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative //host input is governed separately by scheme_relative_handling.
scheme_acceptance	Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from scheme_policy, which governs scheme-*less* input, and from url_standard, which governs interpretation). Defaults to "web". • "web": (Default) Only the curated web-scheme allowlist (http/https/ftp/ftps/file) is admitted; a scheme-bearing input outside it is parse_status = "error". This is the historical, byte-for-byte compatible behavior.

- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

Details

Like `check_hosts` this is a **policy** layer, not parser conformance: it never changes how a URL parses. Restricting which schemes your application will act on is the caller's decision, and this helper supplies the facts to make it — it does not make it for you.

Why the reasons are descriptive: Every token names something observable about the parse. None is a risk label, and none is intended as one: whether a scheme is safe depends on what the caller does with the URL, which this package cannot observe. A `scp:` URL is unremarkable to a mirroring tool and unacceptable in an HTTP fetcher, and no property of the string distinguishes those cases.

Value

A `data.frame` with one row per input URL (input order preserved): `url`, `scheme` (the parsed scheme, lowercased, or `NA`), `scheme_class` (`"special"`, `"non-special"` or `"missing-or-error"`), `web_scheme` (is the scheme one of the five `scheme_acceptance = "web"` admits), `allowed` (only when `allowed_schemes` is supplied), and `reasons` — a list-column whose i-th element is a character vector of the scheme facts observed (`character(0)` when none). The tokens are `"no-scheme"`, `"special-scheme"`, `"non-special-scheme"`, `"outside-web-acceptance"` and `"not-in-allowlist"`.

There is deliberately no token for "carries no authority". It is the natural fact to want for `mailto:` and `javascript:`, but it cannot be computed from the public parse record: `get_host("mailto:someone@example.com")` returns `"example.com"`, reading the `@` as a userinfo delimiter, so a token derived from host presence would be wrong for exactly the schemes it is most wanted for. Reporting a fact this package cannot compute correctly would be worse than not reporting it.

Note

`scheme_acceptance` defaults to `"general"` here, not to the package-wide `"web"`. Auditing which

schemes a URL set carries is pointless under an acceptance mode that has already collapsed every non-web scheme into a parse error.

See Also

check_hosts for the host axis, get_scheme_class, get_url_diagnostics.

Examples

```
urls <- c(
  "https://example.com/a",   # special, inside web acceptance
  "ftp://example.com/a",     # special, inside web acceptance
  "scp://host/a",            # non-special, outside web acceptance
  "smb://server/share",      # non-special, outside web acceptance
  "mailto:someone@example.com", # non-special, opaque path
  "javascript:alert(1)",     # non-special, opaque path
  "notaurl"                  # no scheme
)
check_schemes(urls)
# Score against the schemes an application will actually act on:
check_schemes(urls, allowed_schemes = c("https", "http"))
```

---

format_url *Format a URL for safe human display*

Description

Renders each URL as a string a **person** can read safely: credentials are redacted, invisible and bidirectional-override code points are made visible as <U+XXXX> tokens, percent-encoded delimiters are left encoded so decoding cannot fabricate structure, and an internationalized host is shown in both its Unicode and its ASCII (punycode) spelling whenever the two differ.

Usage

```
format_url(url, engine = NULL)
```

Arguments

url	A character vector of URLs.
engine	Optional psl_engine object from pslr::psl_engine() for per-request Public Suffix List resolution. NULL (default) uses the session-global engine.

Value

A character vector the same length as url. NA_character_ for input the WHATWG parser does not accept.

The result is not a URL

format_url() output is **display only**. It is not reparsable, has no round-trip guarantee, and must never be fed back into serialize_url(), into a comparison key, or into anything that treats it as an address. Use serialize_url() for a standard-exact full string and get_clean_url() for the cleaning surface.

See Also

serialize_url() for the standard-exact full string, get_clean_url() for the cleaning surface, and safe_parse_url() for the parsed components.

Examples

```
# Credentials are redacted, never shown.
format_url("https://user:pw@example.com/a")

# Percent-encoded delimiters stay encoded: decoding them would fabricate
# structure that the URL does not have.
format_url("https://example.com/a%2Fb?x=a%26b%3Dc")

# An internationalized host is shown in both spellings when they differ.
format_url("https://xn--mnchen-3ya.de/p")

# Invisible and bidirectional-override code points are made visible.
format_url("https://example.com/p?q=x#%E2%80%AE%E2%80%8Bevil")

# A byte no valid UTF-8 sequence can contain is emitted, not decoded.
format_url("https://example.com/%FF%00")

# Input the WHATWG parser rejects is NA, not a guess.
format_url("example.com/x")
```

---

get_clean_url *Get cleaned URLs*

Description

This function returns the cleaned version of the URLs after applying protocol, www, case, and trailing slash handling rules. By default the result is a normalized canonical key composed of scheme, host, and path only; port is dropped (port_handling = "exclude"), and fragment/userinfo are always excluded (use get_port, get_fragment, or get_userinfo for those). A URL that carried credentials is silently collapsed to its credential-free spelling by default (credential_handling = "strip"); pass credential_handling = "reject" to get NA for such a row instead (RFC 3986 sections 3.2.1, 7.5 and 7.6; ADR 0017 row 12).

Usage

```
get_clean_url(
  url,
  protocol_handling = "keep",
  www_handling = "none",
  source = c("all", "private", "icann"),
  case_handling = "lower_host",
  trailing_slash_handling = "none",
  index_page_handling = "keep",
  path_normalization = "none",
  scheme_relative_handling = "keep",
  subdomain_levels_to_keep = NULL,
  host_encoding = "keep",
  path_encoding = "keep",
  query_handling = c("drop", "filter", "allow", "keep"),
  params_keep = NULL,
  params_drop = NULL,
  params_case_sensitive = FALSE,
  sort_params = FALSE,
  empty_param_handling = c("keep", "drop"),
  decode_plus = FALSE,
  port_handling = c("exclude", "keep", "strip_default", "strip_all"),
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL,
  engine = NULL,
  profile = NULL,
  credential_handling = c("strip", "reject")
)
```

Arguments

`url` A character vector containing URLs to be parsed.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.
- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).

- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

`www_handling` A character string specifying how to handle "www" and `www[number]` prefixes in the host. Defaults to "none".

- "none": (Default) Leaves the host's www prefix (or lack thereof) untouched.
- "strip": Removes any "www." or `www[number].` prefix.
- "keep": Ensures the host starts with "www.". If it has `www[number].`, it's normalized to "www.". If no www prefix, "www." is added. An empty input host remains empty.
- "if_no_subdomain": If the host is a bare registered domain (e.g., "example.com"), "www." is added. If the host already has a "www." or `www[number].` prefix, it is normalized to "www." (e.g., "www1.example.com" becomes "www.example.com"; "www1.sub.example.com" becomes "www.sub.example.com"). If a non-www subdomain exists (e.g., "sub.example.com" or the normalized "www.sub.example.com"), the host is not further altered. An empty input host remains empty.

`source` Which PSL source to use: "all", "private", or "icann". Subdomain trimming depends on which section is consulted, so pass `source = "icann"` to exclude private suffixes (e.g. github.io).

`case_handling` A character string specifying how to handle the case of the cleaned URL. Defaults to "lower_host", the RFC 3986 §6.2.2.1 normalization (scheme and host are case-insensitive and folded to lowercase; the path is case-sensitive and preserved).

- "lower_host": (Default) Lowercases scheme and host only; the path keeps its original casing.
- "keep": Preserves casing of the reconstructed URL.
- "lower": Converts the cleaned URL to lowercase.
- "upper": Converts the cleaned URL to uppercase.

`trailing_slash_handling`
A character string specifying how to handle trailing slashes in the path component of the cleaned URL. Defaults to "none".

- "none": (Default) No specific handling is applied. Path remains as is after initial parsing.
- "keep": Ensures a trailing slash. If a path exists and doesn't end with one, it's added. If path is just "/", it's kept.
- "strip": Removes a trailing slash if present, unless the path is solely "/".

`index_page_handling`
A character string specifying how to handle index/default pages. Defaults to "keep".

- "keep": (Default) Leave index/default page segments untouched.
- "strip": Remove a trailing index.* or default.* segment (case-insensitive).

`path_normalization`
How to normalize path structure. Defaults to "none". rurl owns dot-segment resolution: the path is read from the input verbatim (never from a pre-normalized

path), so "none" preserves . / .. segments (/a/../b stays /a/../b) and only the settings below change them. Resolution follows RFC 3986 section 5.2.4 and acts on *literal* ./.. segments only — a percent-encoded %2e is a normal path byte, never a dot segment, so it is never treated as traversal.

- "none": (Default) No normalization; dot and slash structure is preserved exactly as written.
- "collapse_slashes": Collapse duplicate slashes in the path.
- "dot_segments": Resolve . and .. segments per RFC 3986.
- "both": Apply both collapse_slashes and dot_segments.

scheme_relative_handling

How to handle URLs starting with "//". Defaults to "keep".

- "keep": Parse using http but return scheme as NA and set status to "ok-scheme-relative".
- "http": Assume http for parsing and output.
- "https": Assume https for parsing and output.
- "error": Treat scheme-relative URLs as invalid.

subdomain_levels_to_keep

An integer or NULL. Determines how many levels of subdomains are kept, in addition to any 'www.' prefix handled by www_handling.

- NULL: (Default) No specific subdomain stripping is performed beyond www_handling.
- 0: All subdomains are stripped. If www_handling preserved or added 'www.', it remains (e.g., 'www.sub.example.com' becomes 'www.example.com'; 'sub.example.com' becomes 'example.com').
- N > 0: Keeps up to N levels of subdomains, counted from right-to-left (closest to the registered domain), in addition to any 'www.' prefix. E.g., if N=1, 'three.two.one.example.com' becomes 'one.example.com'; 'www.three.two.one.example.com' (post www_handling) becomes 'www.one.example.com'.

host_encoding How to present the host in clean_url. Defaults to "keep".

- "keep": Leave the host as parsed (may preserve original case).
- "idna": Convert Unicode host labels to Punycode (IDNA) for the cleaned URL.
- "unicode": Decode Punycode labels to Unicode for the cleaned URL.

Under url_standard = "whatwg" every value renders the UTS-46-mapped host, because mapping is part of WHATWG host parsing rather than a feature of the idna dial (BÜCHER.example presents as bücher.example; RUL-002). There "keep" preserves only whether the input was written as an A-label (xn--...), so get_host() and get_domain() agree on the same row. "rfc3986" and NULL are unaffected.

path_encoding How to present the path percent-encoding in clean_url — the readable-vs-browser rendering choice (the path analog of host_encoding). Defaults to "keep". This is an orthogonal presentation knob: it is independent of url_standard and layers on top of any profile (e.g. url_standard = "whatwg", path_encoding = "encode" emits the WHATWG-parsed path in browser form), exactly like host_encoding. Only "keep" preserves a profile's canonical identity path verbatim; "encode" and "decode" are presentation forms that may re-encode or decode reserved octets (so %2F may fold to a path-separating /), independent of whether a profile is set.

- "keep": Leave the path percent-encoding untouched (the path is preserved as written in the URL, so `%2F` stays `%2F` rather than decoding into a path-separating `/`). With no `url_standard`, rurl keeps its historical RFC-style percent-hex case canonicalization, so `%2f` becomes `%2F`. Under `url_standard = "rfc3986"`, the profile's RFC 3986 §6.2.2.2 normalization applies: a triplet encoding an unreserved byte is decoded, every other triplet stays encoded with uppercased hex, so `%7E` becomes `~` while `%2F` stays `%2F`. Under `url_standard = "whatwg"`, existing percent-triplet spelling is preserved byte-for-byte. Use "encode" to additionally normalize which bytes are encoded.
- "encode": The browser/percent-encoded rendering. Decodes the path first, then percent-encodes each segment (slashes preserved), so a readable non-ASCII path is emitted in its percent-encoded UTF-8 form.
- "decode": The readable rendering. Percent-decodes UTF-8 sequences in the path, so a percent-encoded segment is shown as readable text.

`query_handling` A character string controlling whether (and how) the query string is included in `clean_url`. Defaults to "drop", which preserves the historical query-free `clean_url`. The raw `query` result field is never affected by this option — it always reports the faithful original query.

- "drop": (Default) `clean_url` carries no query, exactly as before.
- "filter": Keep contentful params, dropping known trackers via a built-in denylist (e.g. `utm_*`, `fbclid`, `gclid`). `params_drop` extends the denylist; `params_keep` rescues names (winning over both the denylist and empty-dropping).
- "allow": Keep only params whose names match `params_keep`; all others are dropped. Here `params_keep` is an inclusion criterion only, not an empty-rescue.
- "keep": Keep every param, re-encoded into canonical form (not the verbatim original — that stays on the `query` field).

In every non-"drop" mode the surviving query is re-encoded canonically (uppercase percent-hex, spaces as `%20`) and appended after the path. The query is intentionally EXEMPT from `case_handling` (query values are case-sensitive — tokens, IDs, signatures), so under `case_handling = "lower"` or `"upper"` the `clean_url` is no longer uniformly cased: scheme/host/path fold but the query keeps its original case. Because `clean_url` is the `canonical_join` key, any non-"drop" mode also brings the query into that join key (so `?id=1` and `?id=2` stop collapsing, while `utm`-only differences still collapse under "filter").

`params_keep` Character vector of parameter-name globs (only `*` is special), or `NULL` (default). In "filter" mode this is the rescue list; in "allow" mode it is the allowlist. Ignored in "drop"/"keep".

`params_drop` Character vector of parameter-name globs to add to the built-in denylist in "filter" mode, or `NULL` (default). Ignored in "drop"/"allow"/"keep".

`params_case_sensitive` Logical (default `FALSE`). Controls whether the denylist and `params_keep`/`params_drop` matching is case-sensitive.

`sort_params` Logical (default `FALSE`). When `TRUE`, surviving params are stably sorted by decoded key. Active in "filter"/"allow"/"keep".

`empty_param_handling`
: One of "keep" (default) or "drop". "drop" removes empty-valued params (e.g. `?ref=`), except those rescued by `params_keep` in "filter" mode.

`decode_plus`
: Logical (default `FALSE`). When `TRUE`, `+` in query values is treated as a space (HTML-form decoding) before percent-decoding. `FALSE` keeps `+` literal (RFC 3986 generic behavior).

`port_handling`
: A character string controlling whether the port appears in `clean_url`. Defaults to "exclude", today's only historical behavior. This knob is standalone and standard-independent (editorial, like `www_handling`) – `url_standard` never governs whether it may be set.

  - "exclude": (Default) The port never appears in `clean_url`.
  - "strip_all": Explicit alias of "exclude".
  - "keep": Include the syntactic port when present, including a default port under `url_standard = "whatwg"`. This is an explicit non-parity override for callers that need the input's port spelling.
  - "strip_default": Keep only non-default ports (using the same scheme-default table), independent of `url_standard`. Default-ness is judged on the scheme the input was **parsed** with, never on the scheme `protocol_handling` renders: `http://example.com:443/a` under `protocol_handling = "https"` keeps `:443`, and `http://example.com:80/a` drops `:80` (RFC 3986 §6.2.3; WHATWG URL Standard port state; RUL-016). This is the value `profile = "seo"` pins.

`scheme_policy`
: Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

  - "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
  - "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`
: Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

  - "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
  - "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme

with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under `"whatwg"` the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (`http`/`https`/`ftp`) and nulls default ports in parse output; use `port_handling = "strip_default"` for spec-style clean URL port rendering. See `resolve_url` for `url_standard`-governed reference resolution. The selector does **not** govern whether `port_handling` may be set (it is a standalone editorial knob), nor does it govern `path_encoding` (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

`engine` Optional **pslr** engine controlling which Public Suffix List backs domain / TLD / subdomain extraction: `NULL` (default) resolves against **pslr**'s session-global default list — exactly the historical behavior — while a `pslr::psl_engine()` snapshot resolves against that specific list, per request, without mutating any global state (never call `pslr::psl_use()` for this). Use it to pin a particular list version or to load an alternate list via `pslr::psl_engine(source = "path", path = ...)`. **Process-local:** an engine holds a C++ external pointer that does **not** serialize across R sessions or parallel workers — build it in the process that uses it; never cache it to disk or send it to a worker (rebuild one per process instead). Only the domain-derived outputs (`domain`, `tld`, and the subdomain-trimmed host / `clean_url`) depend on it.

`profile` Optional named profile bundling several knobs at once: `NULL` (default; behaves exactly as the individual arguments select, fully backward compatible), `"browser"`, `"whatwg"`, `"rfc-syntax"`, `"seo"`, or the `"seo"` alias `"canonical"`. A profile is **separate** from `url_standard` (it bundles acceptance, interpretation, leniency, and canonicalization together) and expands only into arguments you

did not supply explicitly — an explicit argument always overrides the profile. "browser" is a browser-*like* fix-up posture (http-prepending; not Chrome-faithful); "whatwg" is the absolute-URL no-base posture that *rejects* schemeless input (unlike a bare url_standard = "whatwg"); "rfc-syntax" is RFC 3986 generic syntax as parsing, not normalization (case and dot-segments are preserved); "seo"/"canonical" is rurl's origin-cleaning intent — a **lossy policy projection of a WHATWG-parsed URL** (ADR 0017), which claims no resource equivalence: url_standard = "whatwg" underneath (which also resolves ./.. folder segments), https, a Unicode host regardless of the input spelling, strip www / trailing slash / index page, drop the whole query, and drop a **default** port only (port_handling = "strip_default": a non-default port names a different origin and survives). Inspect the resolved bundle with url_profile. Also accepted by canonical_join (forwarded through its ...).

credential_handling

How clean_url treats a URL whose parsed authority carried a userinfo delimiter (user@, user:password@, a bare @, or a repeated @). Defaults to "strip". A policy dial on the clean surface (ADR 0017, mutation-table row 12; RUL-001), not a standards axis: it composes with every url_standard, including NULL, and never touches the user / password columns, parse_status, the diagnostics, serialize_url or get_url_key.

- "strip": (Default) The userinfo is dropped and the rest of the URL is emitted, exactly as before this argument existed.
- "reject": clean_url is NA for such a row. RFC 3986 section 3.2.1 deprecates the user:password form and lets an application reject it; sections 7.5 and 7.6 describe the credential leak and the https://example.com@evil.example/ semantic attack a silently collapsed clean URL would hide. Use this when a cleaned URL that *looks* like the credential-free original would be misleading.

There is no "keep": serialize_url already preserves credentials under both standards, and format_url redacts them for display.

Details

The query string is dropped by default (query_handling = "drop"), so the historical scheme/host/path output is byte-identical. Pass query_handling = "keep", "filter", or "allow" (with the companion params_* / sort_params / empty_param_handling / decode_plus arguments) to retain a shaped query on the cleaned URL; the engine is the same one safe_parse_url and get_query use, so get_clean_url(u, query_handling = "filter") equals safe_parse_url(u, query_handling = "filter")$clean_url.

The port is included only when port_handling != "exclude"; see safe_parse_url for the full port_handling semantics.

Value

A character vector of cleaned URLs.

Cleaning is lossy by design

A cleaned URL is an SEO and deduplication product, not an identity function and not a standard's serialization. It is deliberately *lossy*: the fragment and userinfo are always dropped, the query is dropped by default, and the cleaning policies (`www_handling`, `trailing_slash_handling`, `index_page_handling`, subdomain trimming, `port_handling`) exist precisely to collapse URLs that differ. Distinct URLs therefore map to the same cleaned string, so this is never an identity function and never round-trips back to its input.

Two kinds of knob do the collapsing, and `profile = "seo"` carries both (RUL-017). *Normalization* knobs apply what a standard says yields the same resource: the WHATWG parse, host case and UTS #46 rendering, and removal of a default port. *Editorial* knobs assert a fact about the site that no standard settles: https, no `www.`, no trailing slash, no index page, no query. Only the editorial knobs can change the addressed resource; see `url_profile` for the per-knob classification.

It is also **not** the surface rurl's standards-conformance claims are measured on. When identity or conformance is the goal, use a different surface:

- `serialize_url` — the selected standard's own full-string serialization, credentials and fragment included. This is the substrate the conformance claims are measured on.
- `get_url_key` (with `url_key_policy`) and the `url_join` family — resource identity for deduplicating, grouping, matching and joining. No cleaning option can reach a key byte.

Path percent-encoding under a standard

`path_encoding` is a presentation knob and does not select a standard's path identity; `url_standard` does. On the path axis the selector means:

- `url_standard = "rfc3986"` applies RFC 3986 section 6.2.2.2 percent-encoding normalization: a triplet encoding an *unreserved* byte (`A-Z`, `a-z`, `0-9`, `-`, `.`, `_`, `~`) is decoded to that byte, and every other triplet — reserved bytes such as `%2F`, `%3F`, `%23`, and non-ASCII bytes such as `%C3` — is left encoded with its hex digits uppercased. It is applied *before* dot-segment removal, so an encoded dot segment (`%2E` / `%2E%2E`) resolves like a literal one.
- `url_standard = "whatwg"` preserves existing percent-triplets byte-for-byte, hex case included, and resolves encoded dot segments without any decode.
- `url_standard = NULL` decodes nothing and canonicalizes triplet hex case to uppercase.

That normalization is part of the path a URL denotes, and only `path_encoding = "keep"` (the default) preserves it verbatim — `"encode"` and `"decode"` layer a presentation form on top and may fold a reserved octet such as `%2F` into a path-separating `/`. See `vignette("url-standard")`.

See Also

`serialize_url` for a standard's full-string serialization (the conformance-bearing surface), `get_url_key` and `url_join` for resource identity and joining, and `safe_parse_url` for the parsed components.

Examples

```
get_clean_url("Example.COM/Path") # Default lower_host: host folds, path kept
get_clean_url(
  "Example.COM/Path",
```

```
  case_handling = "keep",
  trailing_slash_handling = "keep"
)
get_clean_url(
  "Example.COM/Path/",
  case_handling = "upper",
  trailing_slash_handling = "strip"
)
get_clean_url("http://example.com", www_handling = "strip")
get_clean_url(
  "http://deep.sub.domain.example.com/path",
  subdomain_levels_to_keep = 0
)
# -> "http://example.com/path"
get_clean_url(
  "http://www.deep.sub.domain.example.com/path",
  subdomain_levels_to_keep = 1,
  www_handling = "strip"
)
# -> "http://domain.example.com/path"
get_clean_url(
  "http://www.deep.sub.domain.example.com/path",
  subdomain_levels_to_keep = 1,
  www_handling = "keep"
)
# -> "http://www.domain.example.com/path"
# Query dropped by default (byte-identical to earlier releases):
get_clean_url("http://example.com/p?utm_source=nl&id=42")
# -> "http://example.com/p"
# Strip trackers, keep contentful params:
get_clean_url(
  "http://example.com/p?utm_source=nl&id=42",
  query_handling = "filter"
)
# -> "http://example.com/p?id=42"
# Lossy by design: different URLs clean to the same string, and the cleaned
# string is not what a standard would serialize.
u <- c("http://u:pw@example.com/a#frag", "http://example.com/a?q=1")
get_clean_url(u)
# -> both "http://example.com/a"
# Ask for NA instead of a silently collapsed credential-bearing URL:
get_clean_url(u, credential_handling = "reject")
# -> NA, "http://example.com/a"
serialize_url(u[1])
# The identity surface keeps them apart: the query is identity, the
# fragment and userinfo are not.
k <- get_url_key(u)
k[1] == k[2]
# -> FALSE
# RFC 3986 section 6.2.2.2 on the path: unreserved triplets decode,
# reserved ones stay encoded.
get_clean_url("http://example.com/a%7Eb%2Fc", url_standard = "rfc3986")
# -> "http://example.com/a~b%2Fc"
```

```
get_clean_url("http://example.com/a%7Eb%2Fc", url_standard = "whatwg")
# -> "http://example.com/a%7Eb%2Fc"
```

`get_domain` *Get domain names*

Description

Extracts the registered domain name from a URL (e.g., "example.com"). Relies on the Public Suffix List.

Usage

```
get_domain(
  url,
  protocol_handling = "keep",
  www_handling = "none",
  subdomain_levels_to_keep = NULL,
  source = c("all", "private", "icann"),
  host_encoding = c("keep", "idna", "unicode"),
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL,
  engine = NULL
)
```

Arguments

`url` A character vector of URLs.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.
- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

`www_handling` A character string specifying how to handle "www" and `www[number]` prefixes in the host. Defaults to "none".

- "none": (Default) Leaves the host's www prefix (or lack thereof) untouched.
- "strip": Removes any "www." or `www[number].` prefix.
- "keep": Ensures the host starts with "www.". If it has `www[number].`, it's normalized to "www.". If no www prefix, "www." is added. An empty input host remains empty.
- "if_no_subdomain": If the host is a bare registered domain (e.g., "example.com"), "www." is added. If the host already has a "www." or `www[number].` prefix, it is normalized to "www." (e.g., "www1.example.com" becomes "www.example.com"; "www1.sub.example.com" becomes "www.sub.example.com"). If a non-www subdomain exists (e.g., "sub.example.com" or the normalized "www.sub.example.com"), the host is not further altered. An empty input host remains empty.

`subdomain_levels_to_keep`

An integer or NULL. Determines how many levels of subdomains are kept, in addition to any 'www.' prefix handled by `www_handling`.

- `NULL`: (Default) No specific subdomain stripping is performed beyond `www_handling`.
- `0`: All subdomains are stripped. If `www_handling` preserved or added 'www.', it remains (e.g., 'www.sub.example.com' becomes 'www.example.com'; 'sub.example.com' becomes 'example.com').
- `N > 0`: Keeps up to N levels of subdomains, counted from right-to-left (closest to the registered domain), in addition to any 'www.' prefix. E.g., if N=1, 'three.two.one.example.com' becomes 'one.example.com'; 'www.three.two.one.example.com' (post www_handling) becomes 'www.one.example.com'.

`source` Which PSL source to use: "all", "private", or "icann".

`host_encoding` How to present the host in `clean_url`. Defaults to "keep".

- "keep": Leave the host as parsed (may preserve original case).
- "idna": Convert Unicode host labels to Punycode (IDNA) for the cleaned URL.
- "unicode": Decode Punycode labels to Unicode for the cleaned URL.

Under `url_standard = "whatwg"` every value renders the UTS-46-mapped host, because mapping is part of WHATWG host parsing rather than a feature of the `idna` dial (`BÜCHER.example` presents as `bücher.example`; RUL-002). There `"keep"` preserves only whether the input was written as an A-label (`xn--...`), so `get_host()` and `get_domain()` agree on the same row. `"rfc3986"` and `NULL` are unaffected.

`scheme_policy` Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.

- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`

Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under `"whatwg"` the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (`http`/`https`/`ftp`) and nulls default ports in parse output; use `port_handling = "strip_default"` for spec-style clean URL port rendering. See `resolve_url` for `url_standard`-governed reference resolution. The selector does **not** govern whether `port_handling` may be set (it is a standalone editorial knob), nor does

it govern `path_encoding` (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

`engine` Optional **pslr** engine controlling which Public Suffix List backs domain / TLD / subdomain extraction: `NULL` (default) resolves against **pslr**'s session-global default list — exactly the historical behavior — while a `pslr::psl_engine()` snapshot resolves against that specific list, per request, without mutating any global state (never call `pslr::psl_use()` for this). Use it to pin a particular list version or to load an alternate list via `pslr::psl_engine(source = "path", path = ...)`. **Process-local:** an engine holds a C++ external pointer that does **not** serialize across R sessions or parallel workers — build it in the process that uses it; never cache it to disk or send it to a worker (rebuild one per process instead). Only the domain-derived outputs (`domain`, `tld`, and the subdomain-trimmed host / `clean_url`) depend on it.

Value

A character vector of domain names.

Examples

```
get_domain("http://www.example.co.uk/path")
```

`get_fragment` *Get URL fragments*

Description

Extracts the fragment component of a URL. The value is never percent-decoded. Under `url_standard = "whatwg"` it carries the standard's percent-encoded spelling (the fragment percent-encode set is applied, so a double-quote inside the fragment becomes `%22`); under `url_standard = "rfc3986"` or no selector it is the raw source spelling, exactly as written in the URL.

Usage

```
get_fragment(
  url,
  protocol_handling = "keep",
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL
)
```

Arguments

`url` A character vector of URLs.

protocol_handling
: A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

  - "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.
  - "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
  - "strip": Any existing scheme is removed (scheme component will be NA).
  - "http": The scheme is forced to be "http".
  - "https": The scheme is forced to be "https".

scheme_policy
: Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

  - "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
  - "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

scheme_acceptance
: Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

  - "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
  - "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR

0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under `"whatwg"` the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (`http`/`https`/`ftp`) and nulls default ports in parse output; use `port_handling = "strip_default"` for spec-style clean URL port rendering. See `resolve_url` for `url_standard`-governed reference resolution. The selector does **not** govern whether `port_handling` may be set (it is a standalone editorial knob), nor does it govern `path_encoding` (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

Value

A character vector of fragments.

Examples

```
get_fragment("http://example.com/path#section")
get_fragment("http://example.com/p#a\"b", url_standard = "whatwg")
```

`get_host` *Get URL hosts*

Description

Extracts the host component of a URL.

Usage

```
get_host(
  url,
  protocol_handling = "keep",
  www_handling = "none",
  source = c("all", "private", "icann"),
  subdomain_levels_to_keep = NULL,
  case_handling = c("lower", "keep", "upper", "lower_host"),
  host_encoding = c("keep", "idna", "unicode"),
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL,
  engine = NULL
)
```

Arguments

`url` A character vector of URLs.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.
- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

`www_handling` A character string specifying how to handle "www" and `www[number]` prefixes in the host. Defaults to "none".

- "none": (Default) Leaves the host's www prefix (or lack thereof) untouched.
- "strip": Removes any "www." or `www[number].` prefix.
- "keep": Ensures the host starts with "www.". If it has `www[number].`, it's normalized to "www.". If no www prefix, "www." is added. An empty input host remains empty.
- "if_no_subdomain": If the host is a bare registered domain (e.g., "example.com"), "www." is added. If the host already has a "www." or `www[number].` prefix, it is normalized to "www." (e.g., "www1.example.com" becomes "www.example.com"; "www1.sub.example.com" becomes "www.sub.example.com").

If a non-www subdomain exists (e.g., "sub.example.com" or the normalized "www.sub.example.com"), the host is not further altered. An empty input host remains empty.

`source` Which PSL source to use: "all", "private", or "icann". Subdomain trimming depends on which section is consulted, so pass `source = "icann"` to exclude private suffixes (e.g. github.io).

`subdomain_levels_to_keep`
An integer or NULL. Determines how many levels of subdomains are kept, in addition to any 'www.' prefix handled by `www_handling`.

- `NULL`: (Default) No specific subdomain stripping is performed beyond `www_handling`.
- `0`: All subdomains are stripped. If `www_handling` preserved or added 'www.', it remains (e.g., 'www.sub.example.com' becomes 'www.example.com'; 'sub.example.com' becomes 'example.com').
- `N > 0`: Keeps up to N levels of subdomains, counted from right-to-left (closest to the registered domain), in addition to any 'www.' prefix. E.g., if N=1, 'three.two.one.example.com' becomes 'one.example.com'; 'www.three.two.one.example.com' (post www_handling) becomes 'www.one.example.com'.

`case_handling` How to handle casing of the returned host. Defaults to "lower".

`host_encoding` How to present the host in `clean_url`. Defaults to "keep".

- "keep": Leave the host as parsed (may preserve original case).
- "idna": Convert Unicode host labels to Punycode (IDNA) for the cleaned URL.
- "unicode": Decode Punycode labels to Unicode for the cleaned URL.

Under `url_standard = "whatwg"` every value renders the UTS-46-mapped host, because mapping is part of WHATWG host parsing rather than a feature of the `idna` dial (`BÜCHER.example` presents as `bücher.example`; RUL-002). There `"keep"` preserves only whether the input was written as an A-label (`xn--...`), so `get_host()` and `get_domain()` agree on the same row. `"rfc3986"` and `NULL` are unaffected.

`scheme_policy` Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`
Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under `"whatwg"` the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (`http`/`https`/`ftp`) and nulls default ports in parse output; use `port_handling = "strip_default"` for spec-style clean URL port rendering. See `resolve_url` for `url_standard`-governed reference resolution. The selector does **not** govern whether `port_handling` may be set (it is a standalone editorial knob), nor does it govern `path_encoding` (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

`engine` Optional **pslr** engine controlling which Public Suffix List backs domain / TLD / subdomain extraction: `NULL` (default) resolves against **pslr**'s session-global default list — exactly the historical behavior — while a `pslr::psl_engine()` snapshot resolves against that specific list, per request, without mutating any global state (never call `pslr::psl_use()` for this). Use it to pin a particular list version or to load an alternate list via `pslr::psl_engine(source = "path", path = ...)`. **Process-local:** an engine holds a C++ external pointer that does

not serialize across R sessions or parallel workers — build it in the process that uses it; never cache it to disk or send it to a worker (rebuild one per process instead). Only the domain-derived outputs (`domain`, `tld`, and the subdomain-trimmed host / `clean_url`) depend on it.

Details

Under `scheme_acceptance = "general"` a `mailto:` URL's first recipient domain is returned, decomposed through the same PSL seam a web host uses, so `get_domain` / `get_tld` / `get_subdomain` work on it too (ADR 0012 D7). This deliberately diverges from `safe_parse_url`, whose `host` column is `NA` for a `mailto:` URL: a `mailto:` is a WHATWG opaque path and has no authority, so the recipient domain is surfaced here as extraction metadata rather than presented as a parsed authority. Under the default `"web"` acceptance a `mailto:` URL is not parsed and this returns `NA`.

Value

A character vector of URL hosts.

Examples

```
get_host("http://sub.example.com:8080")
get_host(
  "http://www.two.one.example.com",
  subdomain_levels_to_keep = 1
) # Result: "www.one.example.com"
get_host(
  "http://www.two.one.example.com",
  www_handling = "strip",
  subdomain_levels_to_keep = 1
) # Result: "one.example.com"
get_host(
  "http://www.two.one.example.com",
  www_handling = "keep",
  subdomain_levels_to_keep = 1
) # Result: "www.one.example.com"
get_host(
  "http://three.two.one.example.com",
  subdomain_levels_to_keep = 0
) # Result: "example.com"
get_host(
  "http://www.three.two.one.example.com",
  subdomain_levels_to_keep = 0
) # Result: "www.example.com"
```

`get_host_type` *Classify the host of each URL under a standard profile*

Description

Companion helper for the `url_standard` selector: reports the host *type* of each URL as exactly one of `"domain"`, `"ipv4"`, `"ipv6"`, `"reg-name"`, or `"missing"`. Unlike a raw host string, `host_type` is a function of *both* the host and the selected standard: the numeric host `2130706433` is a `"reg-name"` under `"rfc3986"` but an `"ipv4"` address under `"whatwg"`. Callers reading the result must therefore know which selector produced it.

Usage

```
get_host_type(
  url,
  url_standard,
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general")
)
```

Arguments

`url`	A character vector of URLs.
`url_standard`	Standard profile governing host interpretation: either `"rfc3986"` or `"whatwg"`. **Required, with no default** (ADR 0015). The requirement is semantic, not stylistic: whether a host is an IPv4 literal or a registered name is a question only a standard answers, so there is no profile-neutral classification a default could stand for. `get_parse_verdicts` is deliberately *not* gated this way — its layers describe the parse that actually ran, which is defined with or without a selector.
`scheme_policy`	Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer". • "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior. • "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.
`scheme_acceptance`	Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web". • "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior. • "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires

an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

Details

The metadata is intentionally exposed through this helper rather than as a column on `safe_parse_urls` or a field on `safe_parse_url`, keeping those functions' output shapes fixed (ADR 0006).

Value

A character vector the same length as `url`, each element one of the `host_type` tokens above, or `NA` for a row that cannot be classified under the selected standard.

`NA` **means exactly one thing:** this row is unclassifiable under the standard you named. It can no longer also mean "no selector was passed", because omitting `url_standard` is an error rather than a mode (ADR 0015). An all-`NA` result is therefore evidence about the input, not about the call.

See Also

`get_url_diagnostics`, `get_parse_verdicts`, `safe_parse_url`

Examples

```
get_host_type("http://example.com/", url_standard = "rfc3986")
get_host_type("http://2130706433/", url_standard = "whatwg")
```

---

get_mailto_recipients *Per-recipient email diagnostics for the mailto: positional recipient list*

Description

Companion helper (ADR 0006) that reports structural, per-recipient *facts* about the recipients in the positional `to` of a `mailto:` URL — the comma-separated `addr-spec` list before the `?` (RFC 6068 section 2). Recipients carried in `to`/`cc`/`bcc` *hfields* are RFC 5322 address-lists and are deliberately **out of scope**; only the positional list is analysed.

Usage

```
get_mailto_recipients(
  url,
  url_standard = "rfc3986",
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("general", "web"),
  smtp_wire = FALSE
)
```

Arguments

`url`	A character vector of URLs. Non-`mailto:` URLs (and, under `scheme_acceptance = "web"`, all `mailto:` URLs, which the web allowlist does not accept) contribute no rows.
`url_standard`	Standard profile passed to the general parser. A `mailto:` path is opaque, so this does not affect the classification; it defaults to `"rfc3986"` because the general parser requires a selector.
`scheme_policy`	Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer". • "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior. • "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.
`scheme_acceptance`	Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web". • "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior. • "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR

0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`smtp_wire` Logical; when `TRUE`, compute the opt-in SMTP wire-projection columns (see the corresponding section). Defaults to `FALSE`.

Details

Each fact **names the grammar it was judged against**. The left of the `addr-spec` is classified as an RFC 6068 `local-part` and, independently, the right is classified both as RFC 6068 `mailto` domain vocabulary and as an SMTP (RFC 5321) mailbox right-hand side — these are distinct grammars, so no single column spans them. Public-suffix knowledge is reported separately and is explicitly **non-validating**: a known suffix is not mailbox validity.

Value

A `data.frame` (always, including for length-1 or all-empty input) with one row per positional-`to` recipient and columns:

`url` the source URL the recipient came from.

`recipient_index` 1-based index of the recipient within that URL's positional list.

`mailto_local_part_form` RFC 6068 local-part form: `"dot-atom-text"`, `"quoted-string"`, `"invalid"`, or `"indeterminate"`.

`mailto_domain_form` RFC 6068 domain form: `"ascii-dot-atom-text"`, `"idna2008-domain"`, `"bracketed-domain"`, `"invalid"`, or `"indeterminate"`.

`smtp_mailbox_rhs_syntax_form` RFC 5321 mailbox RHS form, independent of the mailto grammar and of DNS: `"domain"`, `"address-literal"`, `"invalid"`, or `"indeterminate"`.

`public_suffix_known` `TRUE`/`FALSE` whether the domain's public suffix is known to the PSL (non-validating); `NA` when the RHS is not a domain form.

`smtp_domain_wire_form` (opt-in) `"ascii-domain"`, `"a-label-domain"`, `"u-label-domain"`, `"address-literal"`, or `"unavailable"`.

`smtp_envelope_wire_mode` (opt-in) `"ascii"`, `"smtputf8"`, or `"unavailable"`.

`smtp_envelope_address_requires_smtputf8` (opt-in) logical; `NA` when no wire projection could be made.

`smtp_local_part_length_ok` (opt-in) logical, serialized local-part at most 64 octets; `NA` when unavailable.

`smtp_direct_forward_path_fits` (opt-in) logical, the octet length of `"<"` + `Mailbox` + `">"` is at most 256; `NA` when unavailable. The familiar 254 is the RFC 3696 EID 1690 derivation of this path limit, not a standalone production.

Facts, not a gate

These are **selected structural facts**, not a conformance oracle and never a validator. A recipient's classification never turns a parse into an error, and the absence of an `invalid` value does not imply the address is deliverable or fully RFC-conformant. No DNS resolution or deliverability check is performed.

SMTP wire-projection facts (opt-in)

Set `smtp_wire = TRUE` to additionally compute the SMTP transport facts that require an actual serialized wire projection of the address (octet-length limits per RFC 5321 section 4.5.3.1, and the SMTPUTF8 envelope mode per RFC 6531/6530). These are octet facts on the UTF-8 wire bytes, distinct from the syntax classifications above and from DNS. When `smtp_wire = FALSE` (the default) the five `smtp_*` wire columns are still present but carry the `"unavailable"`/`NA` sentinel; the same sentinel is used for a recipient whose address cannot be projected (an invalid mailbox).

Provenance-preserving parse

The positional list is tokenized on the **raw** (still percent-encoded) source before decoding, so an encoded comma (`%2C`) is never a recipient separator and an encoded quote or bracket (`%22`, `%5B`/`%5D`) still protects a raw comma; each field is then percent-decoded exactly once and classified.

See Also

`get_url_diagnostics`, `safe_parse_url`

Examples

```
get_mailto_recipients("mailto:jane@example.com",
  scheme_acceptance = "general")
get_mailto_recipients(
  "mailto:a@example.com,\"b,c\"@example.org",
  scheme_acceptance = "general"
)
# opt-in SMTP wire-projection facts
get_mailto_recipients("mailto:a@xn--mnchen-3ya.de",
  scheme_acceptance = "general", smtp_wire = TRUE)
```

`get_parse_status` *Get the parse status of URLs*

Description

The status is a single value collapsed from three independent facts — URL syntax, admission policy, and the Public Suffix List annotation — so it is a *lossy* view of them. The guaranteed loss: a structural syntax failure and a policy rejection both report `"error"`. Call `get_parse_verdicts` when you need to tell those apart, or to read the PSL result as a typed annotation state rather than as a warning. Nothing here is deprecated — the layered accessor is purely additive.

Usage

```
get_parse_status(
  url,
  protocol_handling = "keep",
  www_handling = "none",
  subdomain_levels_to_keep = NULL,
  source = c("all", "private", "icann"),
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL
)
```

Arguments

`url` A character vector of URLs to be parsed.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.
- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

`www_handling` A character string specifying how to handle "www" and `www[number]` prefixes in the host. Defaults to "none".

- "none": (Default) Leaves the host's www prefix (or lack thereof) untouched.
- "strip": Removes any "www." or `www[number].` prefix.
- "keep": Ensures the host starts with "www.". If it has `www[number].`, it's normalized to "www.". If no www prefix, "www." is added. An empty input host remains empty.
- "if_no_subdomain": If the host is a bare registered domain (e.g., "example.com"), "www." is added. If the host already has a "www." or `www[number].` prefix, it is normalized to "www." (e.g., "www1.example.com" becomes "www.example.com"; "www1.sub.example.com" becomes "www.sub.example.com"). If a non-www subdomain exists (e.g., "sub.example.com" or the normalized "www.sub.example.com"), the host is not further altered. An empty input host remains empty.

subdomain_levels_to_keep
: An integer or NULL. Determines how many levels of subdomains are kept, in addition to any 'www.' prefix handled by www_handling.

  - NULL: (Default) No specific subdomain stripping is performed beyond www_handling.
  - 0: All subdomains are stripped. If www_handling preserved or added 'www.', it remains (e.g., 'www.sub.example.com' becomes 'www.example.com'; 'sub.example.com' becomes 'example.com').
  - N > 0: Keeps up to N levels of subdomains, counted from right-to-left (closest to the registered domain), in addition to any 'www.' prefix. E.g., if N=1, 'three.two.one.example.com' becomes 'one.example.com'; 'www.three.two.one.example.com' (post www_handling) becomes 'www.one.example.com'.

source
: Which PSL source to use: "all", "private", or "icann". Warning statuses such as warning-no-tld, warning-invalid-tld, and warning-public-suffix depend on which PSL section is consulted, so pass source = "icann" to use only ICANN-managed TLDs.

scheme_policy
: Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from protocol_handling, which only controls how the scheme is *presented*, and from url_standard, which controls interpretation). Defaults to "infer".

  - "infer": (Default) Fabricate http:// for scheme-less host-shaped input (e.g. example.com parses as http://example.com), a browser-omnibox-style affordance. This is the historical behavior.
  - "require": Reject scheme-less input — a scheme-less host-shaped value becomes parse_status = "error" rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative //host input is governed separately by scheme_relative_handling.

scheme_acceptance
: Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from scheme_policy, which governs scheme-*less* input, and from url_standard, which governs interpretation). Defaults to "web".

  - "web": (Default) Only the curated web-scheme allowlist (http/https/ftp/ftps/file) is admitted; a scheme-bearing input outside it is parse_status = "error". This is the historical, byte-for-byte compatible behavior.
  - "general": Admit any syntactically valid scheme token and parse opaque (mailto:x), non-special (foo://host), and RFC-generic URLs. Requires an explicit url_standard ("rfc3986" or "whatwg"), which decides the interpretation; general with url_standard = NULL is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no // is an *opaque path*: it has no authority, so host, user, port and the domain/tld columns are all NA and the entire remainder is the path (query/fragment are still split off). This includes mailto: — the recipient's @ never re-triggers authority parsing. To decompose a mailto: recipient, use the accessors (get_host() / get_domain() / get_user(), ADR 0012 D7) or get_mailto_recipients(); those deliberately return a recipient's parts where this table presents NA, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under `"whatwg"` the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (`http`/`https`/`ftp`) and nulls default ports in parse output; use `port_handling = "strip_default"` for spec-style clean URL port rendering. See `resolve_url` for `url_standard`-governed reference resolution. The selector does **not** govern whether `port_handling` may be set (it is a standalone editorial knob), nor does it govern `path_encoding` (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

Value

A character vector with the parse status of each URL: one of `"ok"`, `"ok-ftp"`, `"ok-scheme-relative"`, `"warning-no-tld"`, `"warning-invalid-tld"`, `"warning-public-suffix"`, `"warning-userinfo"` (a scheme-less input carrying userinfo, e.g. `"user@example.com"`), or `"error"`. See `safe_parse_url` for the full semantics.

See Also

`get_parse_verdicts` for the unprojected layers

Examples

```
get_parse_status(
  c("http://example.com", "ftp://example.com", "mailto:user@example.com")
)
get_parse_status(c("http://example.com", "not-a-url"))
get_parse_status("http://example.com", source = "icann")
```

`get_parse_verdicts` *Report the layered validation verdicts for each URL*

Description

Companion helper that separates the three independent questions `get_parse_status` collapses into one value: did the input present well-formed URL syntax (layer 1), was the parsed object admitted under the active policy (layer 2), and what did the Public Suffix List annotation find (layer 3).

Usage

```
get_parse_verdicts(
  url,
  url_standard = NULL,
  protocol_handling = c("keep", "none", "strip", "http", "https"),
  scheme_relative_handling = c("keep", "http", "https", "error"),
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  tld_source = c("all", "private", "icann")
)
```

Arguments

`url` A character vector of URLs.

`url_standard` Standard profile governing interpretation: `NULL` (default), `"rfc3986"`, or `"whatwg"`. Unlike the other companion helpers this argument does not gate the result.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.
- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

`scheme_relative_handling`
How to handle URLs starting with "//". Defaults to "keep".

- "keep": Parse using http but return scheme as NA and set status to "ok-scheme-relative".
- "http": Assume http for parsing and output.
- "https": Assume https for parsing and output.
- "error": Treat scheme-relative URLs as invalid.

`scheme_policy` Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`

Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`tld_source` Which TLD source to use for TLD extraction: "all", "icann", or "private". Defaults to "all".

Details

The single `parse_status` value is a *lossy* projection of these three. The guaranteed loss is that a structural syntax failure and a policy rejection both surface as `"error"`: under the default `scheme_acceptance = "web"`, `"mailto:jane@example.com"` and `"http://"` are both `"error"`, but the first is `layer2_policy_verdict = "rejected-scheme"` (rurl declined to accept the scheme) while the second is `layer1_syntax_verdict = "fail"` (it did not parse). Recovering that distinction is what this helper is for.

Like the other companion helpers this never widens the `safe_parse_url` frame — the parse table keeps its 18 columns and the verdicts live here. Unlike `get_host_type` and `get_scheme_class`, which require a selector, it is fully defined at `url_standard = NULL`: layers 1 and 2 describe the parse that actually occurred, which happens with or without a standard selector.

Value

A data frame with one row per element of `url` and three character columns: `layer1_syntax_verdict`, `layer2_policy_verdict`, and `layer3_annotation_state`.

Why this helper is not gated on `url_standard`

`get_host_type` and `get_scheme_class` *require* a selector and error without one (ADR 0015). That requirement is not a house style — it is a consequence of what those helpers report. Their content is *standard-relative*: whether a host is an IPv4 address, or a scheme is special, is a question only a standard can answer, so without a selector there is no fact to return.

Verdict layers are not standard-relative in that way. A syntax failure (layer 1) and an admission rejection (layer 2) are facts about the parse *that this call actually performed*, under whatever options were supplied. Those facts exist at `url_standard = NULL` exactly as they do under a selector, so there is nothing to withhold.

Gating anyway would also break the helper's central guarantee. Because `parse_status` is the projection of these three layers, projecting the reported layers must reproduce the reported status. Refusing to answer at `url_standard = NULL` while `parse_status` still reports a real value would make the companion contradict the column it exists to explain — on the most common call, and precisely when a caller is asking why a default-options parse failed.

Layers

Layer 1 — syntax (`"pass"` / `"fail"`). Whether a syntax failure was *observed*. `"pass"` is the absence of an observed failure, not a proof of conformance — a row the admission gate rejected before any parse ran reports its rejection in layer 2 and is not additionally reported as a syntax failure.

Layer 2 — policy (`"admitted"`, `"admitted-scheme-relative"`, `"admitted-ftp"`, `"rejected-scheme"`, `"warn-userinfo"`). Whether the object is admitted under the active `scheme_acceptance` / `scheme_relative_handling` / userinfo policy, and with what note. `"rejected-scheme"` rejects; `"warn-userinfo"` accepts with a note; the `admitted*` values accept.

Layer 3 — annotation (`"not-applicable"`, `"known"`, `"unknown"`, and the currently unproduced `"not-requested"`, `"invalid-input"`, `"dependency-error"`). The PSL registrability fact as a typed state rather than a bare `NA`, and **never fatal**: a host with no public suffix is a `"unknown"` annotation, not a parse failure. A host form with no registrable-domain concept at all — an IP literal, a `file:` host, an opaque authority — is `"not-applicable"`, never `"unknown"`.

See Also

get_parse_status, get_url_diagnostics, safe_parse_url

Examples

```
# Both are parse_status "error" -- for entirely different reasons.
get_parse_verdicts(c("mailto:jane@example.com", "http://"))

# A PSL miss is an annotation state, never a fatal verdict.
get_parse_verdicts("http://example.invalidtld/")
```

---

get_password *Get URL passwords*

Description

Extracts the password component of a URL. The value is never percent-decoded. Under `url_standard = "whatwg"` it carries the standard's percent-encoded spelling (the userinfo percent-encode set is applied, so a ":" inside the password becomes `%3A`); under `url_standard = "rfc3986"` or no selector it is the raw source spelling, exactly as written in the URL. This is the same contract as get_user.

Usage

```
get_password(
  url,
  protocol_handling = "keep",
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL
)
```

Arguments

`url` A character vector of URLs.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.

- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

`scheme_policy` Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`

Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the

host IPv4/reg-name model, and case_handling — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (path_normalization or case_handling) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only case_handling = "lower_host" is accepted under a selector — "keep", "lower", and "upper" all conflict, since "lower" also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under "whatwg" the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (http/https/ftp) and nulls default ports in parse output; use port_handling = "strip_default" for spec-style clean URL port rendering. See resolve_url for url_standard-governed reference resolution. The selector does **not** govern whether port_handling may be set (it is a standalone editorial knob), nor does it govern path_encoding (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

Value

A character vector of passwords.

See Also

get_user, get_userinfo.

Examples

```
get_password("ftp://user:password@ftp.example.com/file.txt")
get_password("http://u:p:q@example.com/", url_standard = "whatwg")
```

---

get_path *Get URL paths*

Description

Extracts the path component of a URL.

Usage

```
get_path(
  url,
  protocol_handling = "keep",
  case_handling = c("lower_host", "keep", "lower", "upper"),
  trailing_slash_handling = c("none", "keep", "strip"),
  index_page_handling = c("keep", "strip"),
  path_normalization = c("none", "collapse_slashes", "dot_segments", "both"),
  path_encoding = c("keep", "encode", "decode"),
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
```

```
  url_standard = NULL
)
```

Arguments

`url`	A character vector of URLs.
`protocol_handling`	A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it. • "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`. • "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA). • "strip": Any existing scheme is removed (scheme component will be NA). • "http": The scheme is forced to be "http". • "https": The scheme is forced to be "https".
`case_handling`	How to handle casing of the returned path. Defaults to "lower_host", which preserves the path's original casing (paths are case-sensitive per RFC 3986 §6.2.2.1). Use "lower"/"upper" to force a case.
`trailing_slash_handling`	A character string specifying how to handle trailing slashes in the path component of the cleaned URL. Defaults to "none". • "none": (Default) No specific handling is applied. Path remains as is after initial parsing. • "keep": Ensures a trailing slash. If a path exists and doesn't end with one, it's added. If path is just "/", it's kept. • "strip": Removes a trailing slash if present, unless the path is solely "/".
`index_page_handling`	A character string specifying how to handle index/default pages. Defaults to "keep". • "keep": (Default) Leave index/default page segments untouched. • "strip": Remove a trailing index.* or default.* segment (case-insensitive).
`path_normalization`	How to normalize path structure. Defaults to "none". rurl owns dot-segment resolution: the path is read from the input verbatim (never from a pre-normalized path), so `"none"` preserves `.` / `..` segments (`/a/../b` stays `/a/../b`) and only the settings below change them. Resolution follows RFC 3986 section 5.2.4 and acts on *literal* `./..` segments only — a percent-encoded `%2e` is a normal path byte, never a dot segment, so it is never treated as traversal.

- "none": (Default) No normalization; dot and slash structure is preserved exactly as written.
- "collapse_slashes": Collapse duplicate slashes in the path.
- "dot_segments": Resolve . and .. segments per RFC 3986.
- "both": Apply both collapse_slashes and dot_segments.

`path_encoding` How to present the path percent-encoding in `clean_url` — the readable-vs-browser rendering choice (the path analog of `host_encoding`). Defaults to "keep". This is an orthogonal presentation knob: it is independent of `url_standard` and layers on top of any profile (e.g. `url_standard = "whatwg", path_encoding = "encode"` emits the WHATWG-parsed path in browser form), exactly like `host_encoding`. Only "keep" preserves a profile's canonical identity path verbatim; "encode" and "decode" are presentation forms that may re-encode or decode reserved octets (so `%2F` may fold to a path-separating `/`), independent of whether a profile is set.

- "keep": Leave the path percent-encoding untouched (the path is preserved as written in the URL, so `%2F` stays `%2F` rather than decoding into a path-separating `/`). With no `url_standard`, rurl keeps its historical RFC-style percent-hex case canonicalization, so `%2f` becomes `%2F`. Under `url_standard = "rfc3986"`, the profile's RFC 3986 §6.2.2.2 normalization applies: a triplet encoding an unreserved byte is decoded, every other triplet stays encoded with uppercased hex, so `%7E` becomes `~` while `%2F` stays `%2F`. Under `url_standard = "whatwg"`, existing percent-triplet spelling is preserved byte-for-byte. Use "encode" to additionally normalize which bytes are encoded.
- "encode": The browser/percent-encoded rendering. Decodes the path first, then percent-encodes each segment (slashes preserved), so a readable non-ASCII path is emitted in its percent-encoded UTF-8 form.
- "decode": The readable rendering. Percent-decodes UTF-8 sequences in the path, so a percent-encoded segment is shown as readable text.

`scheme_policy` Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`
Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.

- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under `"whatwg"` the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (`http`/`https`/`ftp`) and nulls default ports in parse output; use `port_handling = "strip_default"` for spec-style clean URL port rendering. See `resolve_url` for `url_standard`-governed reference resolution. The selector does **not** govern whether `port_handling` may be set (it is a standalone editorial knob), nor does it govern `path_encoding` (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

Value

A character vector of URL paths.

Examples

```
get_path("http://example.com/some/path?query=1")
```

`get_port` *Get URL ports*

Description

Extracts the port component of a URL.

Usage

```
get_port(
  url,
  protocol_handling = "keep",
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL
)
```

Arguments

`url` A character vector of URLs.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.
- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

`scheme_policy` Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.

- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`

Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under `"whatwg"` the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (`http`/`https`/`ftp`) and nulls default ports in parse output; use `port_handling = "strip_default"` for spec-style clean URL port rendering. See `resolve_url` for `url_standard`-governed reference resolution. The selector does **not** govern whether `port_handling` may be set (it is a standalone editorial knob), nor does

it govern path_encoding (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

Details

Under url_standard = "whatwg" a port equal to the scheme's default is *not part of the parsed URL* (the standard discards it during parsing), so "http://example.com:80/" reports NA rather than 80. Under url_standard = "rfc3986" or no selector the written port is reported as-is. This is distinct from port_handling, a presentation dial that governs whether a port is rendered into clean_url; the two are independent.

Value

An integer vector of ports.

Examples

```
get_port("http://example.com:8080/path")
get_port("http://example.com:80/path", url_standard = "whatwg")
```

---

get_query *Get URL query strings*

Description

Extracts the query component of a URL, optionally parsing it into a list.

Usage

```
get_query(
  url,
  protocol_handling = "keep",
  format = c("string", "list"),
  decode = TRUE,
  query_handling = c("keep", "drop", "filter", "allow"),
  params_keep = NULL,
  params_drop = NULL,
  params_case_sensitive = FALSE,
  sort_params = FALSE,
  empty_param_handling = c("keep", "drop"),
  decode_plus = FALSE,
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL
)
```

Arguments

url	A character vector of URLs.
protocol_handling	A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. mailto:, tel:, ws:) yields parse_status = "error". Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. "asdfghjkl", "12345", "/path") or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like "192.168.010.1") is rejected as "error" rather than having a scheme fabricated for it. • "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields "error". • "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA). • "strip": Any existing scheme is removed (scheme component will be NA). • "http": The scheme is forced to be "http". • "https": The scheme is forced to be "https".
format	Return format: "string" (default) or "list" for parsed elements.
decode	Logical; if TRUE (default), percent-decodes the query (the whole string for format="string", keys/values for format="list"). Set FALSE to obtain the query as written: the raw query for the default query_handling = "keep", or the canonical re-encoded form (uppercase hex, %20, %26/%3D) once any filtering is requested.
query_handling	A character string controlling whether (and how) the query string is included in clean_url. Defaults to "drop", which preserves the historical query-free clean_url. The raw query result field is never affected by this option — it always reports the faithful original query. • "drop": (Default) clean_url carries no query, exactly as before. • "filter": Keep contentful params, dropping known trackers via a built-in denylist (e.g. utm_*, fbclid, gclid). params_drop extends the denylist; params_keep rescues names (winning over both the denylist and empty-dropping). • "allow": Keep only params whose names match params_keep; all others are dropped. Here params_keep is an inclusion criterion only, not an empty-rescue. • "keep": Keep every param, re-encoded into canonical form (not the verbatim original — that stays on the query field). In every non-"drop" mode the surviving query is re-encoded canonically (uppercase percent-hex, spaces as %20) and appended after the path. The query is intentionally EXEMPT from case_handling (query values are case-sensitive — tokens, IDs, signatures), so under case_handling = "lower" or "upper" the clean_url is no longer uniformly cased: scheme/host/path fold but the query keeps its original case. Because clean_url is the canonical_join key, any

non-"drop" mode also brings the query into that join key (so `?id=1` and `?id=2` stop collapsing, while `utm`-only differences still collapse under "filter").

`params_keep` Character vector of parameter-name globs (only `*` is special), or `NULL` (default). In "filter" mode this is the rescue list; in "allow" mode it is the allowlist. Ignored in "drop"/"keep".

`params_drop` Character vector of parameter-name globs to add to the built-in denylist in "filter" mode, or `NULL` (default). Ignored in "drop"/"allow"/"keep".

`params_case_sensitive`
Logical (default `FALSE`). Controls whether the denylist and `params_keep`/`params_drop` matching is case-sensitive.

`sort_params` Logical (default `FALSE`). When `TRUE`, surviving params are stably sorted by decoded key. Active in "filter"/"allow"/"keep".

`empty_param_handling`
One of "keep" (default) or "drop". "drop" removes empty-valued params (e.g. `?ref=`), except those rescued by `params_keep` in "filter" mode.

`decode_plus` Logical (default `FALSE`). When `TRUE`, `+` in query values is treated as a space (HTML-form decoding) before percent-decoding. `FALSE` keeps `+` literal (RFC 3986 generic behavior).

`scheme_policy` Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`
Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path`

(query/fragment are still split off). This includes mailto: — the recipient's @ never re-triggers authority parsing. To decompose a mailto: recipient, use the accessors (get_host() / get_domain() / get_user(), ADR 0012 D7) or get_mailto_recipients(); those deliberately return a recipient's parts where this table presents NA, because a recipient domain is extraction metadata, not the URL's authority.

Note that file: is admitted under *both* values, including the default. A file: URL denotes local-filesystem access, and one with a non-empty host (file://server/share/x) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses file: URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see SECURITY.md.

url_standard Optional top-level standard profile: NULL (default), "rfc3986", or "whatwg". With NULL the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and case_handling — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (path_normalization or case_handling) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only case_handling = "lower_host" is accepted under a selector — "keep", "lower", and "upper" all conflict, since "lower" also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under "whatwg" the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (http/https/ftp) and nulls default ports in parse output; use port_handling = "strip_default" for spec-style clean URL port rendering. See resolve_url for url_standard-governed reference resolution. The selector does **not** govern whether port_handling may be set (it is a standalone editorial knob), nor does it govern path_encoding (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

Details

The underlying parse preserves the raw query string byte-for-byte (a bare key such as ?flag stays flag, not flag=). By default this accessor still percent-decodes for readability (decode = TRUE); pass decode = FALSE to obtain the raw query exactly as written in the URL.

Under url_standard = "whatwg" the underlying query carries the standard's percent-encoded spelling (the query percent-encode set is applied, so a literal space becomes %20); under url_standard = "rfc3986" or no selector it is the raw source spelling. That distinction is only visible with decode = FALSE, since decoding collapses both spellings.

The filtering arguments (query_handling, params_keep, params_drop, params_case_sensitive, sort_params, empty_param_handling, decode_plus) share the engine used by get_clean_url, but default to query_handling = "keep" here: an accessor returns the query as found unless you ask it to filter. When no filtering or reordering is requested (the default profile), the output is byte-for-byte identical to earlier releases; once you opt in, the surviving params are selected first and only then rendered per format/decode.

Value

A character vector (format="string") or list (format="list").

Security note

Because `decode = TRUE` is the **default**, this accessor can return characters the URL itself never contained literally — including control characters such as CR and LF, which `%0D`/`%0A` decode to. Treat the result as untrusted input: do not interpolate it into a header, a log line, a shell command, or a SQL statement without escaping it for that sink, and do not assume it is single-line. Pass `decode = FALSE` when you want the query exactly as written, with no decoding step at all.

Examples

```
get_query("http://example.com/path?a=1&b=2")
get_query("http://example.com/path?a=1&b=2", format = "list")
# Drop trackers, keep contentful params:
get_query(
  "http://example.com/?utm_source=nl&id=42",
  query_handling = "filter"
)
# Canonical (re-encoded) form:
get_query(
  "http://example.com/?a=1%262",
  query_handling = "keep", decode = FALSE
)
```

`get_scheme` *Get URL schemes*

Description

Extracts the scheme (protocol) of a URL.

Usage

```
get_scheme(
  url,
  protocol_handling = "keep",
  scheme_relative_handling = "keep",
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL
)
```

Arguments

`url` A character vector of URLs.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.
- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

`scheme_relative_handling`
How to handle URLs starting with "//". Defaults to "keep".

- "keep": Parse using http but return scheme as NA and set status to "ok-scheme-relative".
- "http": Assume http for parsing and output.
- "https": Assume https for parsing and output.
- "error": Treat scheme-relative URLs as invalid.

`scheme_policy` Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`
Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.

- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under `"whatwg"` the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (`http`/`https`/`ftp`) and nulls default ports in parse output; use `port_handling = "strip_default"` for spec-style clean URL port rendering. See `resolve_url` for `url_standard`-governed reference resolution. The selector does **not** govern whether `port_handling` may be set (it is a standalone editorial knob), nor does it govern `path_encoding` (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

Details

By default (`scheme_acceptance = "web"`) only rurl's web-scheme allowlist parses, so an opaque scheme such as `mailto:` or `tel:` yields `NA`. Pass `scheme_acceptance = "general"` (which requires an explicit `url_standard`) to run the general parser, under which those schemes resolve and their scheme string is returned.

Value

A character vector of URL schemes.

Examples

```
get_scheme("https://example.com")
get_scheme(
  "mailto:jane@example.com",
  url_standard = "rfc3986", scheme_acceptance = "general"
)
```

`get_scheme_class` *Classify the scheme of each URL as WHATWG special or not*

Description

Companion helper for the `url_standard` selector: reports whether each URL's resolved scheme is a WHATWG "special scheme" (`"special"`), one rurl supports but WHATWG does not treat specially (`"non-special"`), or absent/unparseable (`"missing-or-error"` – an unsupported scheme, a scheme-relative URL under the default `scheme_relative_handling = "keep"`, or an input that failed to parse at all).

Usage

```
get_scheme_class(
  url,
  url_standard,
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general")
)
```

Arguments

`url`	A character vector of URLs.
`url_standard`	Standard profile under which each URL is parsed before its scheme is classified: either `"rfc3986"` or `"whatwg"`. **Required, with no default** (ADR 0015). The three tokens below do not differ between the profiles, but the parse that resolves the scheme does, so the profile has to be named. `get_parse_verdicts` is deliberately *not* gated this way — its layers describe the parse that actually ran, which is defined with or without a selector.
`scheme_policy`	Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer". • "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior. • "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

scheme_acceptance

Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from scheme_policy, which governs scheme-*less* input, and from url_standard, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (http/https/ftp/ftps/file) is admitted; a scheme-bearing input outside it is parse_status = "error". This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (mailto:x), non-special (foo://host), and RFC-generic URLs. Requires an explicit url_standard ("rfc3986" or "whatwg"), which decides the interpretation; general with url_standard = NULL is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no // is an *opaque path*: it has no authority, so host, user, port and the domain/tld columns are all NA and the entire remainder is the path (query/fragment are still split off). This includes mailto: — the recipient's @ never re-triggers authority parsing. To decompose a mailto: recipient, use the accessors (get_host() / get_domain() / get_user(), ADR 0012 D7) or get_mailto_recipients(); those deliberately return a recipient's parts where this table presents NA, because a recipient domain is extraction metadata, not the URL's authority.

Note that file: is admitted under *both* values, including the default. A file: URL denotes local-filesystem access, and one with a non-empty host (file://server/share/x) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses file: URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see SECURITY.md.

Details

Unlike get_host_type, the classification itself does not vary between "rfc3986" and "whatwg" – "special scheme" is a WHATWG concept describing a fixed property of the scheme string, not something RFC 3986 redefines. url_standard is nonetheless required, mirroring get_host_type()'s contract: the resolved scheme this classification reads is itself produced by a profile-dependent parse, so the profile has to be named (ADR 0015).

Within rurl's allowlist (http/https/ftp/ftps/ file), http, https, ftp, and file are WHATWG special schemes; ftps (FTP-over-TLS, rurl's own addition) is not. This is metadata only – it does not add ws/wss to rurl's allowed schemes and does not change what safe_parse_url accepts.

Under the default scheme_acceptance = "web" an opaque scheme such as mailto: is outside rurl's web allowlist and classifies as "missing-or-error". Pass scheme_acceptance = "general" to run the general parser, under which such a scheme resolves and classifies as "non-special" (it is not a WHATWG special scheme).

Value

A character vector the same length as url, each element one of "special", "non-special", or "missing-or-error".

Never NA**.** Every element receives one of the three tokens above — input that is unparseable, scheme-less, empty or NA classifies as "missing-or-error" rather than falling through to NA. The one arm that used to return NA was the selector-less call, which is now an error (ADR 0015).

See Also

get_host_type, get_scheme

Examples

```
get_scheme_class("http://example.com/", url_standard = "whatwg")
get_scheme_class("ftps://example.com/", url_standard = "whatwg")
get_scheme_class("//example.com/path", url_standard = "whatwg")
get_scheme_class(
  "mailto:jane@example.com",
  url_standard = "rfc3986", scheme_acceptance = "general"
)
```

---

get_subdomain *Get URL subdomains*

Description

Extracts the subdomain component of a URL.

Usage

```
get_subdomain(
  url,
  protocol_handling = "keep",
  www_handling = "none",
  source = c("all", "private", "icann"),
  include_www = FALSE,
  format = c("string", "labels"),
  host_encoding = c("keep", "idna", "unicode"),
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL,
  engine = NULL
)
```

Arguments

url	A character vector of URLs.
protocol_handling	A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme

(e.g. mailto:, tel:, ws:) yields parse_status = "error". Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. "asdfghjkl", "12345", "/path") or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like "192.168.010.1") is rejected as "error" rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields "error".
- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

www_handling A character string specifying how to handle "www" and www[number] prefixes in the host. Defaults to "none".

- "none": (Default) Leaves the host's www prefix (or lack thereof) untouched.
- "strip": Removes any "www." or www[number]. prefix.
- "keep": Ensures the host starts with "www.". If it has www[number]., it's normalized to "www.". If no www prefix, "www." is added. An empty input host remains empty.
- "if_no_subdomain": If the host is a bare registered domain (e.g., "example.com"), "www." is added. If the host already has a "www." or www[number]. prefix, it is normalized to "www." (e.g., "www1.example.com" becomes "www.example.com"; "www1.sub.example.com" becomes "www.sub.example.com"). If a non-www subdomain exists (e.g., "sub.example.com" or the normalized "www.sub.example.com"), the host is not further altered. An empty input host remains empty.

source Which PSL source to use: "all", "private", or "icann".

include_www Logical; if FALSE (default), removes a leading www/www[0-9]* label only when it is the sole subdomain label.

format Return format: "string" (default) or "labels" for a character vector of labels.

host_encoding How to present the host in clean_url. Defaults to "keep".

- "keep": Leave the host as parsed (may preserve original case).
- "idna": Convert Unicode host labels to Punycode (IDNA) for the cleaned URL.
- "unicode": Decode Punycode labels to Unicode for the cleaned URL.

Under url_standard = "whatwg" every value renders the UTS-46-mapped host, because mapping is part of WHATWG host parsing rather than a feature of the idna dial (BÜCHER.example presents as bücher.example; RUL-002). There "keep" preserves only whether the input was written as an A-label (xn--...), so get_host() and get_domain() agree on the same row. "rfc3986" and NULL are unaffected.

scheme_policy Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from protocol_handling, which only controls how the scheme is

presented, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`

Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their

meaning; always pass it by name. Under "whatwg" the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (http/https/ftp) and nulls default ports in parse output; use port_handling = "strip_default" for spec-style clean URL port rendering. See resolve_url for url_standard-governed reference resolution. The selector does **not** govern whether port_handling may be set (it is a standalone editorial knob), nor does it govern path_encoding (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

engine — Optional **pslr** engine controlling which Public Suffix List backs domain / TLD / subdomain extraction: NULL (default) resolves against **pslr**'s session-global default list — exactly the historical behavior — while a pslr::psl_engine() snapshot resolves against that specific list, per request, without mutating any global state (never call pslr::psl_use() for this). Use it to pin a particular list version or to load an alternate list via pslr::psl_engine(source = "path", path = ...). **Process-local:** an engine holds a C++ external pointer that does **not** serialize across R sessions or parallel workers — build it in the process that uses it; never cache it to disk or send it to a worker (rebuild one per process instead). Only the domain-derived outputs (domain, tld, and the subdomain-trimmed host / clean_url) depend on it.

Value

A character vector (format="string") or list of label vectors (format="labels").

Examples

```
get_subdomain("http://www.blog.example.co.uk")
get_subdomain("http://www.blog.example.co.uk", format = "labels")
```

---

get_tld *Extract the top-level domain (TLD) from a URL*

Description

Uses safe_parse_url internally to extract the TLD, benefiting from all memoization layers for improved performance.

Usage

```
get_tld(
  url,
  source = c("all", "private", "icann"),
  host_encoding = c("keep", "idna", "unicode"),
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL,
  engine = NULL
)
```

Arguments

url	A character vector of URLs.
source	Which TLD source to use: "all", "icann", or "private".
host_encoding	How to present the host in clean_url. Defaults to "keep". • "keep": Leave the host as parsed (may preserve original case). • "idna": Convert Unicode host labels to Punycode (IDNA) for the cleaned URL. • "unicode": Decode Punycode labels to Unicode for the cleaned URL. Under url_standard = "whatwg" every value renders the UTS-46-mapped host, because mapping is part of WHATWG host parsing rather than a feature of the idna dial (BÜCHER.example presents as bücher.example; RUL-002). There "keep" preserves only whether the input was written as an A-label (xn--...), so get_host() and get_domain() agree on the same row. "rfc3986" and NULL are unaffected.
scheme_policy	Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from protocol_handling, which only controls how the scheme is *presented*, and from url_standard, which controls interpretation). Defaults to "infer". • "infer": (Default) Fabricate http:// for scheme-less host-shaped input (e.g. example.com parses as http://example.com), a browser-omnibox-style affordance. This is the historical behavior. • "require": Reject scheme-less input — a scheme-less host-shaped value becomes parse_status = "error" rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative //host input is governed separately by scheme_relative_handling.
scheme_acceptance	Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from scheme_policy, which governs scheme-*less* input, and from url_standard, which governs interpretation). Defaults to "web". • "web": (Default) Only the curated web-scheme allowlist (http/https/ftp/ftps/file) is admitted; a scheme-bearing input outside it is parse_status = "error". This is the historical, byte-for-byte compatible behavior. • "general": Admit any syntactically valid scheme token and parse opaque (mailto:x), non-special (foo://host), and RFC-generic URLs. Requires an explicit url_standard ("rfc3986" or "whatwg"), which decides the interpretation; general with url_standard = NULL is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no // is an *opaque path*: it has no authority, so host, user, port and the domain/tld columns are all NA and the entire remainder is the path (query/fragment are still split off). This includes mailto: — the recipient's @ never re-triggers authority parsing. To decompose a mailto: recipient, use the accessors (get_host() / get_domain() / get_user(), ADR 0012 D7) or get_mailto_recipients(); those deliberately return a recipient's parts where this table presents NA, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under `"whatwg"` the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (`http`/`https`/`ftp`) and nulls default ports in parse output; use `port_handling = "strip_default"` for spec-style clean URL port rendering. See `resolve_url` for `url_standard`-governed reference resolution. The selector does **not** govern whether `port_handling` may be set (it is a standalone editorial knob), nor does it govern `path_encoding` (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

`engine` Optional **pslr** engine controlling which Public Suffix List backs domain / TLD / subdomain extraction: `NULL` (default) resolves against **pslr**'s session-global default list — exactly the historical behavior — while a `pslr::psl_engine()` snapshot resolves against that specific list, per request, without mutating any global state (never call `pslr::psl_use()` for this). Use it to pin a particular list version or to load an alternate list via `pslr::psl_engine(source = "path", path = ...)`. **Process-local:** an engine holds a C++ external pointer that does **not** serialize across R sessions or parallel workers — build it in the process that uses it; never cache it to disk or send it to a worker (rebuild one per process instead). Only the domain-derived outputs (`domain`, `tld`, and the subdomain-trimmed host / `clean_url`) depend on it.

Value

A character vector of TLDs.

Examples

```
get_tld("example.com")
```

`get_url_diagnostics` *Report non-fatal diagnostics for each URL under a standard profile*

Description

Companion helper for the `url_standard` selector: reports the non-fatal validation/safety *facts* rurl observed while parsing each URL (for example an IPv4 host written in a numeric or non-decimal shorthand, or a path segment carrying an encoded reserved byte). Diagnostics are facts, not policy: they are emitted keyed to host/path *shape* in both standard modes so a security-sensitive consumer can reject a footgun URL regardless of which selector it chose, while a link-graph consumer can ignore them. The complete token vocabulary is enumerated below under *Diagnostic vocabulary (canonical)*.

Usage

```
get_url_diagnostics(
  url,
  url_standard,
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general")
)
```

Arguments

`url`	A character vector of URLs.
`url_standard`	Standard profile governing interpretation: either `"rfc3986"` or `"whatwg"`. **Required, with no default** (ADR 0015): the vocabulary is profile-dependent — several tokens fire under one standard only — so there is no profile-neutral set of findings a default could stand for.
`scheme_policy`	Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer". • "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior. • "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.
`scheme_acceptance`	Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

Details

A single URL can carry several diagnostics, so the return shape is not a plain scalar-per-URL vector (see *Value*). `parse_status` stays coarse; diagnostics are never encoded into it.

Value

For a length-1 `url`, a character vector of zero or more diagnostic tokens for that URL. For a length-n `url` (including `n == 0`), a list of length n whose i-th element is the character vector of that URL's tokens (`character(0)` when it has none).

An empty result means exactly one thing: that URL raised no diagnostics under the standard you named. It can no longer also mean that no selector was passed, because omitting `url_standard` is an error rather than a mode (ADR 0015). `character(0)` is therefore evidence that the URL is clean under that profile.

Selected facts, not a conformance oracle

The diagnostics are deliberately a **selected** set of facts, **not** a complete validator. The *absence* of a diagnostic never implies the URL conforms to its scheme's specification or to WHATWG/RFC 3986. Full per-standard conformance validation is out of scope (ADR 0012 D5).

Two WHATWG-generic facts gate on the *interpreting standard*, not the acceptance axis, so they are reported whenever `url_standard = "whatwg"` — including the default `"web"` acceptance path (they are route-independent, string-level facts; RURL-sgjzbqzk):

- `invalid-URL-unit` / `invalid-credentials` — WHATWG validation errors (WHATWG-verbatim names): a malformed %-escape or a non-URL code point, and any credentials (userinfo) present. Bounded detection.

The default combination (`"web"` acceptance with `url_standard = NULL`) emits no diagnostics at all, so it is unaffected.

With `scheme_acceptance = "general"` (the general-parser posture) a further set of selected facts is reported. These fire *only* under `"general"`; the default `"web"` acceptance never emits them:

- `unicode-outside-rfc3986-uri` — under `"rfc3986"`, a directly-written non-ASCII scalar value accepted by the sole RFC 3986 generic-grammar tolerance (not RFC 3987/IRI conformance).
- `transform-skipped-ineligible-scheme` — the scheme is non-HTTP(S) and so ineligible for the SEO/semantic Stage-B transforms.
- scheme-specific facts: `ws-fragment-forbidden` / `ws-userinfo-forbidden` (RFC 6455), `mailto-fragment-discouraged` (RFC 6068), `tel-missing-phone-context` (RFC 3966), `data-missing-comma` (RFC 2397), and, under `"rfc3986"`, `file-non-absolute-path`, `file-userinfo-extension` (userinfo, permitted by RFC 8089 Appendix E.1's non-normative extended grammar), and `file-component-outside-rfc8089` (a port, query or fragment, which RFC 8089's grammar does not mention and which are therefore inherited generic RFC 3986 components).

Diagnostic vocabulary (canonical)

This section is the **single authoritative enumeration** of the diagnostics vocabulary. It is held to the runtime registry (`.URL_DIAGNOSTICS`) in both directions by `tools/diagnostics-doc-consistency.R`, a CI gate: a token cannot be added, renamed, or removed without this list moving with it. Earlier design documents (including the v1 selector PRD's section 7 table) are historical records of what the vocabulary was when they were accepted — they are not registries and do not track it.

Every token below is emitted only when `url_standard` is not `NULL`. Tokens marked *general* additionally require `scheme_acceptance = "general"`; the rest fire under both acceptance postures and, unless noted, under both `"rfc3986"` and `"whatwg"`.

Host — IPv4 shape. Facts about a host written as, or coerced to, an IPv4 address; security filters typically reject all of them.

- `ipv4-number-form` — numeric IPv4 shorthand instead of dotted decimal.
- `ipv4-non-dotted` — a whole-host number parsed/coerced to IPv4 in WHATWG mode.
- `ipv4-short-form` — fewer than four dotted parts.
- `ipv4-non-decimal` — hex or octal notation participated in IPv4 parsing.
- `ipv4-octal` — octal interpretation changed the apparent address value.
- `ipv4-leading-zero` — a dotted decimal-looking part had a leading zero.
- `ipv4-out-of-range` — a dotted part exceeds 255 (fatal under `"whatwg"`; flags a numeric-looking `reg-name` under `"rfc3986"`, e.g. `256.1.1.1`).

Host — DNS length, UTS-46 and charset. Probed against the resolved host; IP literals are excluded.

- `domain-label-too-long` — a label exceeds the DNS 63-byte limit.

- `domain-name-too-long` — the whole name exceeds the DNS 253-byte limit.
- `domain-empty-label` — the host contains an empty label (a `".."` run, or a leading dot).
- `domain-hyphen-violation` — a label breaks the UTS-46 hyphen rules (leading/trailing hyphen, or `"--"` in positions 3–4 of a non-`xn--` label).
- `domain-std3-violation` — a label carries a code point outside the STD3 LDH set.
- `host-charset-shimmed` — the host carries one of the 15 code points WHATWG keeps but the historical parser rejected, accepted by the shim (ADR 0009: `! $ & ( ) * + , ; =`, plus the ASCII quotation mark, apostrophe, grave accent, and the two braces). `"whatwg"` only.

Path.

- `encoded-dot-segment` — an encoded-dot segment (`%2e` / `%2e%2e`, any hex case) that the profile's dot handling acted on.
- `encoded-reserved-path-byte` — the preserved path still carries an encoded reserved byte (`%2F`, `%3F`, `%23`) held as data rather than as a separator.

Port. Facts about the raw port versus the resolved scheme's WHATWG default, independent of the `port_handling` knob.

- `explicit-default-port` — the port was written out and equals the scheme's default.
- `non-default-port` — a port is present and is not the scheme's default (including any port on a scheme with no defined default).

Input shape — WHATWG cleanup. All three are `"whatwg"` only; `"rfc3986"` has no strip or rewrite step.

- `invalid-reverse-solidus` — a literal \ was reinterpreted as `/` (special schemes only).
- `control-char-stripped` — an ASCII tab/LF/CR was removed from the interior of the input (step 1, second half).
- `leading-trailing-stripped` — a leading and/or trailing run of C0-control-or-SPACE was removed (step 1, first half).

Layer 5 — selected per-standard and per-scheme facts (ADR 0012 D5). Described in full under *Selected facts, not a conformance oracle* above.

- `invalid-URL-unit` — WHATWG validation error: a malformed `%`-escape or a non-URL code point. `"whatwg"` only.
- `invalid-credentials` — WHATWG validation error: credentials (userinfo) are present. `"whatwg"` only.
- `unicode-outside-rfc3986-uri` — *general*; a directly-written non-ASCII scalar value under `"rfc3986"`.
- `transform-skipped-ineligible-scheme` — *general*; a non-HTTP(S) scheme, ineligible for the Stage-B transforms.
- `ws-fragment-forbidden` — *general*; a fragment on a `ws:`/`wss:` URL (RFC 6455).
- `ws-userinfo-forbidden` — *general*; userinfo on a `ws:`/`wss:` URL (RFC 6455).
- `mailto-fragment-discouraged` — *general*; a fragment on a `mailto:` URL (RFC 6068).

- `tel-missing-phone-context` — *general*; a local `tel:` number with no `phone-context` (RFC 3966).
- `data-missing-comma` — *general*; a `data:` URL with no `","` separator (RFC 2397).
- `file-non-absolute-path` — *general*; a non-absolute `file:` path under `"rfc3986"`.
- `file-userinfo-extension` — *general*; userinfo on a `file:` URL, permitted by RFC 8089 Appendix E.1's non-normative extended grammar.
- `file-component-outside-rfc8089` — *general*; a port, query or fragment on a `file:` URL, inherited from generic RFC 3986.

See Also

`get_host_type`, `safe_parse_url`

Examples

```
get_url_diagnostics("http://example.com/", url_standard = "rfc3986")
get_url_diagnostics(
  c("http://example.com/", "http://2130706433/"),
  url_standard = "whatwg"
)
```

`get_url_key` *URL comparison key*

Description

Projects each URL onto a versioned, non-URL comparison key: the value rurl uses to decide whether two URLs identify the same web resource. Use it to deduplicate, group, or match URLs without relying on a cleaned display string.

Usage

```
get_url_key(url, policy = url_key_policy())
```

Arguments

`url`	A character vector of URLs. Factors are coerced.
`policy`	A `rurl_url_key_policy` object from `url_key_policy()`, which is also the default. The policy is scalar and is never recycled.

Value

A classed character vector (`rurl_url_key`) the same length as `url`, preserving its names. `NA` for a non-keyable element. The policy version, schema version, standard, scheme-equality mode and the per-element `keyability` reasons ride along as attributes.

The key is not a URL

The returned object is a classed character vector whose contents are injectively framed component bytes, not a URL. Never parse it, never render it to users, and never reconstruct a URL from it. `print()` deliberately shows a truncated diagnostic form for that reason. What it *is* good for is comparison: `==`, `match()`, `duplicated()`, `%in%` and the url_join family all work on it directly.

Framing is length-prefixed, so component boundaries cannot be forged. A host or path containing separators, control bytes or colons can never make two different URLs collide.

Non-keyable input

A URL the selected standard cannot parse has no identity, so its key is `NA` and never matches anything – not even another `NA`. The reason is kept alongside rather than collapsed into the `NA`, and is readable with `attr(key, "keyability")`: one of `"ok"`, `"missing-input"`, `"empty-input"` or `"invalid-parse"`. Missing input is never conflated with an invalid parse.

See Also

`url_key_policy()` for the dials, url_join for joining on the key, and `serialize_url()` for a standard's full-string serialization (which *is* a URL, unlike this).

Examples

```
# Presentation differences that are not identity differences.
get_url_key(c("http://example.com:80/a", "http://example.com/a"))

# The fragment and userinfo are excluded from web-resource identity.
k <- get_url_key(c("http://u:pw@example.com/a#top", "http://example.com/a"))
k[1] == k[2]

# Query order and duplicates are significant.
k <- get_url_key(c("http://example.com/?a=1&b=2",
                   "http://example.com/?b=2&a=1"))
k[1] == k[2]

# Deduplicate by identity rather than by string.
u <- c("HTTP://Example.com/a", "http://example.com/a",
       "http://example.com/b")
u[!duplicated(get_url_key(u))]

# Non-keyable input carries a typed reason.
k <- get_url_key(c("http://example.com/", NA, "", ":::"))
attr(k, "keyability")
```

`get_user` *Get URL user names*

Description

Extracts the user component of a URL. The value is never percent-decoded. Under `url_standard = "whatwg"` it carries the standard's percent-encoded spelling (WHATWG stores the username buffer encoded with the userinfo percent-encode set, so `"http://a^b@host/"` yields `"a%5Eb"`); under `url_standard = "rfc3986"` or no selector it is the raw source spelling, exactly as written in the URL.

Usage

```
get_user(
  url,
  protocol_handling = "keep",
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL
)
```

Arguments

`url` A character vector of URLs.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.
- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

`scheme_policy` Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`

Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under `"whatwg"` the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (`http`/`https`/`ftp`) and nulls default ports in parse output; use `port_handling =`

"strip_default" for spec-style clean URL port rendering. See resolve_url for url_standard-governed reference resolution. The selector does **not** govern whether port_handling may be set (it is a standalone editorial knob), nor does it govern path_encoding (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

Details

Under scheme_acceptance = "general" the user of a mailto: URL's first recipient (its addr-spec local-part) is returned, mirroring how get_host / get_domain extract that recipient's domain (ADR 0012 D7). As with get_host, this deliberately diverges from safe_parse_url, whose user column is NA for a mailto: URL (an opaque path carries no authority). Under the default "web" acceptance a mailto: URL is not parsed and this returns NA.

Value

A character vector of user names.

See Also

get_mailto_recipients for the full per-recipient list.

Examples

```
get_user("ftp://user:password@ftp.example.com/file.txt")
get_user("mailto:jane@example.com",
  scheme_acceptance = "general", url_standard = "rfc3986")
```

---

get_userinfo *Get URL userinfo*

Description

Extracts the userinfo component of a URL (user or user:password).

Usage

```
get_userinfo(
  url,
  protocol_handling = "keep",
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL
)
```

Arguments

`url` A character vector of URLs.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.
- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

`scheme_policy` Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`
Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path`

(query/fragment are still split off). This includes mailto: — the recipient's @ never re-triggers authority parsing. To decompose a mailto: recipient, use the accessors (get_host() / get_domain() / get_user(), ADR 0012 D7) or get_mailto_recipients(); those deliberately return a recipient's parts where this table presents NA, because a recipient domain is extraction metadata, not the URL's authority.

Note that file: is admitted under *both* values, including the default. A file: URL denotes local-filesystem access, and one with a non-empty host (file://server/share/x) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses file: URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see SECURITY.md.

url_standard Optional top-level standard profile: NULL (default), "rfc3986", or "whatwg". With NULL the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and case_handling — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (path_normalization or case_handling) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only case_handling = "lower_host" is accepted under a selector — "keep", "lower", and "upper" all conflict, since "lower" also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under "whatwg" the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (http/https/ftp) and nulls default ports in parse output; use port_handling = "strip_default" for spec-style clean URL port rendering. See resolve_url for url_standard-governed reference resolution. The selector does **not** govern whether port_handling may be set (it is a standalone editorial knob), nor does it govern path_encoding (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

Value

A character vector of userinfo values.

Examples

```
get_userinfo("ftp://user:password@ftp.example.com/file.txt")
get_userinfo("ftp://user@ftp.example.com/file.txt")
```

---

is_valid_host *Test whether each URL's host satisfies a practical validation rule*

Description

A policy predicate layered on top of standards-correct parsing. rurl's `url_standard` profiles deliberately match the URL *standards* rather than impose practical web/SEO hygiene, so hosts such as `a+b.example` (a valid RFC 3986 reg-name), `_dmarc.example.com` (a valid DNS owner name), or `-example.com` all parse successfully. `is_valid_host()` answers the separate, product-level question of whether such a host is usable as a practical web hostname, DNS owner name, registrable site host, or SEO-safe host.

Usage

```
is_valid_host(
  url,
  rule = c("web", "dns", "registrable", "seo", "url"),
  url_standard = "whatwg",
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general")
)
```

Arguments

`url`	A character vector of URLs.
`rule`	A single rule to test. One of:
	`"web"` (default) usable as an HTTP-authority host: an IP literal, or a strict letter-digit-hyphen (LDH) name host (labels 1–63 octets, no leading/trailing hyphen, no underscore, name $\leq$ 253).
	`"dns"` a legal DNS owner-name shape — the `"web"` rule but the underscore is permitted anywhere (as `_dmarc`, `_dkim`, SRV owner names require).
	`"registrable"` the host has a Public Suffix List registrable domain (equivalently `get_host_type` is `"domain"`); IP literals and single-label / unknown-suffix hosts are `FALSE`.
	`"seo"` SEO-safe: `"web"` and `"registrable"` and carrying no host-shape footgun diagnostic (e.g. an IPv4 written in a numeric or non-decimal shorthand).
	`"url"` the loosest rule: the selected standard admitted a host at all.
`url_standard`	Standard profile governing host interpretation: `"whatwg"` (default; the living web standard) or `"rfc3986"`. The verdict can depend on the standard — `2130706433` is an `"ipv4"` host (`web = TRUE`) under `"whatwg"` but a reg-name under `"rfc3986"`.
`scheme_policy`	Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".
	• "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
	• "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme.

Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`

Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

Value

A logical vector the same length as `url`. `NA` for a URL with no host to judge (a host-less scheme, or an input that did not parse).

A policy layer, not parser conformance

This never changes how a URL parses: it does not turn a hostname-policy failure into a parse error, does not affect `get_parse_status`, and adds no columns to `safe_parse_url`. It is also **not** a conformance oracle (see `get_url_diagnostics` and ADR 0012): a `TRUE` verdict means only that rurl found no practical footgun it checks for, never that the host is provably valid per every specification.

See Also

`check_hosts` for a tabular report over several rules at once, `get_host_type`, `get_url_diagnostics`.

Examples

```
is_valid_host(c("http://example.com", "http://_dmarc.example.com"))
# web: FALSE for the underscore host; dns: TRUE for it
is_valid_host("http://_dmarc.example.com", rule = "dns")
is_valid_host(
  c("http://a+b.example", "http://-example.com", "http://a..com"),
  rule = "web"
)
is_valid_host(
  c("http://example.com", "http://localhost", "http://192.168.0.1"),
  rule = "registrable"
)
```

---

query_param_summary *Summarize query parameters across a set of URLs*

Description

Tabulates which query parameters appear across a vector of URLs and what values they take, with a `would_drop` column previewing what `query_handling = "filter"` would remove. Useful for auditing a URL set before choosing a cleaning policy: see the trackers before you strip them.

Usage

```
query_param_summary(
  urls,
  level = c("param", "value"),
  params_keep = NULL,
  params_drop = NULL,
  params_case_sensitive = FALSE,
  empty_param_handling = c("keep", "drop"),
  decode_plus = FALSE
)
```

Arguments

`urls`	A character vector of URLs.
`level`	One of "param" (default) for one row per distinct parameter name, or "value" for one row per distinct (parameter, value) pair.
`params_keep`	Character vector of parameter-name globs (only `*` is special), or `NULL` (default). In "filter" mode this is the rescue list; in "allow" mode it is the allowlist. Ignored in "drop"/"keep".
`params_drop`	Character vector of parameter-name globs to add to the built-in denylist in "filter" mode, or `NULL` (default). Ignored in "drop"/"allow"/"keep".
`params_case_sensitive`	Logical (default `FALSE`). Controls whether the denylist and `params_keep`/`params_drop` matching is case-sensitive.

empty_param_handling
: One of "keep" (default) or "drop". "drop" removes empty-valued params (e.g. ?ref=), except those rescued by params_keep in "filter" mode.

decode_plus
: Logical (default FALSE). When TRUE, + in query values is treated as a space (HTML-form decoding) before percent-decoding. FALSE keeps + literal (RFC 3986 generic behavior).

Details

Parameter names are grouped *faithfully* (case-sensitively and by their decoded spelling), so utm_source and UTM_SOURCE are reported as separate rows. The would_drop preview, by contrast, honours params_case_sensitive: with the default params_case_sensitive = FALSE, UTM_SOURCE matches the built-in denylist and shows would_drop = TRUE; set it to TRUE and the upper-case spelling no longer matches. The raw query field is only read, never mutated.

Value

A flat (long) data.frame. For level = "param": param, n (total occurrences), n_urls (distinct URLs containing the param), example_value, example_url, would_drop. For level = "value": param, value, n, n_urls, example_url, would_drop. The example_* columns and the param-level would_drop reflect the first-seen occurrence (deterministic given input order). Returns a zero-row data.frame with the level's columns when no URL carries a query.

Examples

```
urls <- c(
  "http://example.com/?utm_source=nl&id=42",
  "http://example.com/watch?v=abc&utm_source=x",
  "http://example.com/?id=99"
)
query_param_summary(urls)
query_param_summary(urls, level = "value")
# Preview a custom policy:
query_param_summary(urls, params_drop = "id")
```

---

resolve_url *Resolve a URL reference against a base URL*

Description

Resolves a relative or absolute URL reference against a base URL following the RFC 3986 section 5 reference-resolution algorithm, then renders the resolved absolute URL on the output surface output selects. Under the default output = "clean" the result is canonicalized with the same machinery as safe_parse_url; under output = "serialized" it is handed to serialize_url instead. url_standard and any ... options flow straight through to the parse, so the host IPv4/reg-name model, path percent/dot-segment handling, default-port elision, WHATWG backslash-as-slash recognition, and diagnostics are exactly those of a direct safe_parse_url() call on the resolved URL.

Usage

```
resolve_url(
  relative_or_absolute,
  base_url,
  url_standard = NULL,
  output = c("clean", "serialized"),
  form = c("source", "normalized"),
  ...
)
```

Arguments

relative_or_absolute	A character vector of URL references to resolve. Each may be relative ("../b", "?q=1", "#frag", "//host/p") or already absolute ("https://host/p"); an absolute reference ignores base_url.
base_url	A character vector of base URLs, recycled against relative_or_absolute. Each base must itself be an absolute URL (carry a scheme); a relative reference resolved against a scheme-less or NA base yields NA.
url_standard	Optional standard profile forwarded to the parse: NULL (default), "rfc3986", or "whatwg". See safe_parse_url for the axes it governs, and *Reference resolution is standard-aware* for the reference-parsing rules "whatwg" adds ahead of the merge. Required (non-NULL) when output = "serialized".
output	Which output surface to return: "clean" (default, today's canonical clean_url bytes) or "serialized" (the selected standard's full-string serialization of the resolved absolute URL, via serialize_url). See *Which output surface you want*.
form	For output = "serialized" with url_standard = "rfc3986" only, the RFC posture forwarded to serialize_url: "source" (default, source-preserving) or "normalized". Ignored for "whatwg", whose serializer has a single spec-defined form, and ignored under output = "clean", whose rendering is driven by the cleaning dials instead – the same argument-is-inert-where-it-does-not-apply contract serialize_url itself holds for form.
...	Additional arguments forwarded to safe_parse_urls (e.g. port_handling, query_handling, host_encoding). Passing a governed low-level knob that conflicts with url_standard errors, exactly as it does for safe_parse_url. These are presentation dials consumed by the "clean" path only; supplying any of them together with output = "serialized" is an error, because serialize_url takes no presentation arguments and the dial could not be honored.

Value

A character vector the same length as the recycled inputs, unnamed (names are not data). Under output = "clean" each element is the canonical clean_url of the resolved reference; under output = "serialized" it is the standard's full-string serialization of the resolved absolute URL. NA where resolution cannot produce an absolute URL, or where the resolved URL is not accepted by the parser ("clean") or by the selected standard's parser ("serialized").

Reference resolution is standard-aware

The merge itself (empty reference, fragment-only, query-only, scheme-relative `//host` reference, absolute-path reference, and relative-path merge) is RFC 3986 section 5.2–5.3 under `url_standard = "rfc3986"` and under the default `NULL` selector, with one exception under `"rfc3986"` noted last below. Under `url_standard = "whatwg"` the WHATWG URL Standard's reference-parsing rules are applied first, because they are rules the two standards genuinely disagree on rather than composition of the axes `url_standard` already governs (decision P2.7 D-B, `design/work/url-v3/decisions/P2.7-display-and-`

- **A reference carrying the base's own special scheme is relative, not absolute.** WHATWG's "special relative or authority state" consumes a scheme equal to the base's when that scheme is special (`http`, `https`, `ws`, `wss`, `ftp`, `file`) and keeps parsing against the base, so `resolve_url("http:foo.com", "http://example.org/foo/bar", url_standard = "whatwg")` is `"http://example.org/foo/foo.com"`, while RFC 3986 treats any scheme as making the reference absolute and gives `"http://foo.com/"`. A *different* scheme stays absolute under both, even when it is also special.
- **Under a special base, a leading \ in the reference is a /, and a run of either introduces an authority.** WHATWG's "relative slash state" reads `\` exactly as `/`, so `resolve_url("\x", "http://example.org/foo/bar", url_standard = "whatwg", output = "serialized")` is `"http://example.org/x"`; a second slash-or-backslash enters the authority, and "special authority ignore slashes" then skips the whole run for the five non-`file` special schemes, so `"///example.org/x"` and `"/\/\//example.org/x"` both resolve with the host `example.org`. `file` has its own state machine and consumes exactly two. RFC 3986 has neither rule: `\` is an ordinary path byte and `//` is the entire authority production.
- **The reference is stripped before it is read.** The WHATWG basic URL parser's step 1 removes a leading and trailing run of C0 control or SPACE (U+0000–U+0020) and then every ASCII tab, LF and CR, so `" foo.com "` resolves as `"foo.com"` does. A reference that strips to the empty string is the empty reference, which resolves to the base minus its fragment. RFC 3986 has no strip step – such bytes are required to be percent-encoded – so under `"rfc3986"` and `NULL` they stay in the reference.
- **Against a `file:` base, Windows drive letters follow the WHATWG `file:` state machine.** A reference that *begins* with a drive letter empties the base path instead of shortening it, so `resolve_url("C|/foo", "file:///tmp/mock/path", url_standard = "whatwg", output = "serialized")` is `"file:///C:/foo"`; a rooted reference inherits the base's drive letter (`"/"` against `"file:///C:/a/b"` is `"file:///C:/"`); `..` never removes a lone drive letter (`".."` against `"file:///C:/"` is `"file:///C:/"`); and `C|` in the first segment is normalized to `C:`. A drive letter in the authority position (`"//d:"`) is an empty host plus a path segment, `"file:///d:"`. RFC 3986 has no drive-letter concept, so under `"rfc3986"` and `NULL` the plain section 5.2 merge applies.

The `NULL` selector is frozen and unaffected (ADR 0007; P2.7 D-C): every rule above is reachable only through `url_standard = "whatwg"`.

Two further rules apply under **both** named profiles, because the two standards agree on them. First, a resolved path whose first segment is empty is recomposed with the `/.` guard: RFC 3986 section 3.3 forbids a path beginning with `//` after no authority, and the WHATWG URL serializer emits the same guard, so `resolve_url("/..//path","non-spec:/p", url_standard = "whatwg", output = "serialized")` is `"non-spec:/.//path"` rather than a string that re-reads as the authority `path`. The `NULL` selector recomposes the unguarded string, as it always did. Second, a scheme is `ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )` – RFC 3986 section 3.1's own grammar,

and WHATWG's – so a relative path whose first segment merely *contains* a colon is a path, not an absolute reference: `resolve_url("[61:24:74]:98", "http://example.org/foo/bar",url_standard = "whatwg", output = "serialized")` is `"http://example.org/foo/[61:24:74]:98"`, and `"rfc3986"` merges the same way. The `NULL` selector instead keeps RFC 3986 Appendix B's explicitly *non-validating* `[^:/?#]+`, which reads `"10.0.0.7"` as a scheme and discards the base; that is frozen behavior (ADR 0007), not a recommendation.

Which output surface you want

`output` selects between two different products, not two settings of one (decision P2.7 D-A, `design/work/url-v3/decisions`

- `output = "clean"` (the default) returns the *canonical* `clean_url` of the resolved reference, not a verbatim RFC 3986 recomposition: as everywhere else in rurl, the fragment and userinfo are excluded from `clean_url`, the query is included only when `query_handling != "drop"` (the default drops it), and the port only when `port_handling != "exclude"`. This surface is **intentionally lossy** – it is a cleaning/SEO product driven by presentation policy, and it therefore **cannot carry a conformance claim**. This differs from a generic resolver such as `xml2::url_absolute()` or Python's `urljoin`, which preserve every component verbatim; `resolve_url()` resolves *and* canonicalizes.
- `output = "serialized"` returns `serialize_url(<resolved absolute URL>, standard = url_standard, form = form)`: the standard's own full-string serialization, with the fragment preserved and credentials reconstructed. This is the standards surface – the one a conformance claim may be measured on – and it is where RFC 3986 section 5.4's own expectations are reproduced exactly (`resolve_url("?y", "http://a/b/c/d;p?q", url_standard = "rfc3986", output = "serialized")` is `"http://a/b/c/d;p?y"`).

`output = "serialized"` **requires** an explicit `url_standard`: `NULL` selects no standard, so there is nothing to serialize *to*, and the combination is an error rather than a silent choice of one. Because `serialize_url` accepts no presentation options at all, `output = "serialized"` also rejects any `...` argument: honoring, say, `port_handling = "exclude"` is impossible on that surface, and accepting-then-discarding it would misreport what was returned.

To inspect individual resolved components (including the fragment), resolve first and pass the result to `safe_parse_url`.

See Also

`safe_parse_url`, `get_clean_url`, `serialize_url`

Examples

```
resolve_url("../g", "http://a/b/c/d;p?q") # -> "http://a/b/g"
resolve_url("g", "http://a/b/c/d;p?q") # -> "http://a/b/c/g"
resolve_url("//example.org/p", "http://a/b/c") # -> "http://example.org/p"
resolve_url("https://x.com/y", "http://a/b/c") # absolute ref, base ignored
resolve_url(c("g", "../h"), "http://a/b/c/") # vectorized

# The standards surface keeps the query and the fragment RFC 3986 section
# 5.4 requires; the (lossy) clean surface drops both by design.
resolve_url("?y", "http://a/b/c/d;p?q",
            url_standard = "rfc3986", output = "serialized")
```

```
resolve_url("#s", "http://a/b/c/d;p?q",
            url_standard = "whatwg", output = "serialized")
resolve_url("#s", "http://a/b/c/d;p?q")
```

`rurl_cache_config` *Configure the rurl memoization caches*

Description

Enables or disables individual caches and sets an optional bound on the `full_parse` cache. Called with no arguments, it leaves the configuration unchanged and returns the current state.

Usage

```
rurl_cache_config(
  full_parse = NULL,
  puny_encode = NULL,
  puny_decode = NULL,
  max_full_parse = NULL
)
```

Arguments

`full_parse`	Logical; enable/disable the full URL parse cache.
`puny_encode`	Logical; enable/disable the IDNA/Punycode encode cache.
`puny_decode`	Logical; enable/disable the Punycode decode cache.
`max_full_parse`	A single number (≥ 1) or `Inf` bounding the `full_parse` cache.

Details

Disabling a cache stops new writes to it (existing entries are left in place until `rurl_clear_caches` is called). When `full_parse` reaches `max_full_parse` entries, the *entire* cache is cleared before the next new entry is stored, so its peak size never exceeds the bound. This is a hard reset-watermark, not an LRU or FIFO eviction policy: `max_full_parse` caps peak memory, but is *not* a working-set size — once the bound is hit the cache empties completely and rebuilds from scratch. The default bound is `100000` unique url × core-option combinations (the cache stores the option-independent parse core, keyed by url, protocol/scheme handling, `www_handling`, and `tld_source`; set `max_full_parse = Inf` for the historical unbounded behavior). The `puny_encode` and `puny_decode` caches are unbounded by design (each stays small — bounded by the number of unique hosts/labels seen, not URL+option combinations).

Value

Invisibly, the updated `rurl_cache_info` data.frame.

See Also

`rurl_cache_info`, `rurl_clear_caches`

Examples

```
rurl_cache_config(max_full_parse = 10000)
rurl_cache_config(puny_encode = FALSE)
rurl_cache_config() # inspect current configuration
```

---

rurl_cache_info *Inspect the rurl memoization caches*

Description

Reports the number of entries currently held in each memoization cache, along with whether the cache is enabled and any configured entry bound.

Usage

```
rurl_cache_info()
```

Value

A data.frame with one row per cache (`full_parse`, `puny_encode`, `puny_decode`) and columns `entries`, `enabled`, and `max_entries`.

See Also

`rurl_cache_config`, `rurl_clear_caches`

Examples

```
get_domain("https://www.example.com")
rurl_cache_info()
```

---

rurl_clear_caches *Clear all rurl caches*

Description

Clears the memoization caches used by rurl functions. This is useful if you need to free memory.

Usage

```
rurl_clear_caches()
```

Value

Invisibly returns NULL.

Examples

```
rurl_clear_caches()
```

safe_parse_urls *Parse multiple URLs and return a data.frame of components*

Description

Vectorized wrapper around `safe_parse_url` that returns a data.frame with one row per input URL.

Usage

```
safe_parse_urls(
  url,
  protocol_handling = c("keep", "none", "strip", "http", "https"),
  www_handling = c("none", "strip", "keep", "if_no_subdomain"),
  tld_source = c("all", "private", "icann"),
  case_handling = c("lower_host", "keep", "lower", "upper"),
  trailing_slash_handling = c("none", "keep", "strip"),
  index_page_handling = c("keep", "strip"),
  path_normalization = c("none", "collapse_slashes", "dot_segments", "both"),
  scheme_relative_handling = c("keep", "http", "https", "error"),
  subdomain_levels_to_keep = NULL,
  host_encoding = c("keep", "idna", "unicode"),
  path_encoding = c("keep", "encode", "decode"),
  query_handling = c("drop", "filter", "allow", "keep"),
  params_keep = NULL,
  params_drop = NULL,
  sort_params = FALSE,
  empty_param_handling = c("keep", "drop"),
  params_case_sensitive = FALSE,
  decode_plus = FALSE,
  port_handling = c("exclude", "keep", "strip_default", "strip_all"),
  scheme_policy = c("infer", "require"),
  scheme_acceptance = c("web", "general"),
  url_standard = NULL,
  engine = NULL,
  profile = NULL,
  credential_handling = c("strip", "reject")
)
```

Arguments

`url`	A character vector of URLs to be parsed.
`protocol_handling`	A character string specifying how to handle protocols. Defaults to "keep". Regardless of this option, rurl only processes authority-based URLs whose scheme is one of http, https, ftp, or ftps; a scheme-bearing input with any other scheme (e.g. `mailto:`, `tel:`, `ws:`) yields `parse_status = "error"`. Scheme inference (below) also requires the input to be host-shaped: a scheme-less string that is not

a host (e.g. `"asdfghjkl"`, `"12345"`, `"/path"`) or is a non-canonical IP literal (integer/hex/octal/short forms, or leading-zero octets like `"192.168.010.1"`) is rejected as `"error"` rather than having a scheme fabricated for it.

- "keep": If a supported scheme exists (http, https, ftp, ftps), it's used. If no scheme and the input is host-shaped, "http://" is added; otherwise the input is not a URL and yields `"error"`.
- "none": If a supported scheme exists, it's used. If no scheme, then no scheme is used (scheme component will be NA).
- "strip": Any existing scheme is removed (scheme component will be NA).
- "http": The scheme is forced to be "http".
- "https": The scheme is forced to be "https".

`www_handling` A character string specifying how to handle "www" and `www[number]` prefixes in the host. Defaults to "none".

- "none": (Default) Leaves the host's www prefix (or lack thereof) untouched.
- "strip": Removes any "www." or `www[number].` prefix.
- "keep": Ensures the host starts with "www.". If it has `www[number].`, it's normalized to "www.". If no www prefix, "www." is added. An empty input host remains empty.
- "if_no_subdomain": If the host is a bare registered domain (e.g., "example.com"), "www." is added. If the host already has a "www." or `www[number].` prefix, it is normalized to "www." (e.g., "www1.example.com" becomes "www.example.com"; "www1.sub.example.com" becomes "www.sub.example.com"). If a non-www subdomain exists (e.g., "sub.example.com" or the normalized "www.sub.example.com"), the host is not further altered. An empty input host remains empty.

`tld_source` Which TLD source to use for TLD extraction: "all", "icann", or "private". Defaults to "all".

`case_handling` A character string specifying how to handle the case of the cleaned URL. Defaults to "lower_host", the RFC 3986 §6.2.2.1 normalization (scheme and host are case-insensitive and folded to lowercase; the path is case-sensitive and preserved).

- "lower_host": (Default) Lowercases scheme and host only; the path keeps its original casing.
- "keep": Preserves casing of the reconstructed URL.
- "lower": Converts the cleaned URL to lowercase.
- "upper": Converts the cleaned URL to uppercase.

`trailing_slash_handling`
A character string specifying how to handle trailing slashes in the path component of the cleaned URL. Defaults to "none".

- "none": (Default) No specific handling is applied. Path remains as is after initial parsing.
- "keep": Ensures a trailing slash. If a path exists and doesn't end with one, it's added. If path is just "/", it's kept.
- "strip": Removes a trailing slash if present, unless the path is solely "/".

index_page_handling
: A character string specifying how to handle index/default pages. Defaults to "keep".
  - "keep": (Default) Leave index/default page segments untouched.
  - "strip": Remove a trailing index.* or default.* segment (case-insensitive).

path_normalization
: How to normalize path structure. Defaults to "none". rurl owns dot-segment resolution: the path is read from the input verbatim (never from a pre-normalized path), so `"none"` preserves `.` / `..` segments (`/a/../b` stays `/a/../b`) and only the settings below change them. Resolution follows RFC 3986 section 5.2.4 and acts on *literal* `./..` segments only — a percent-encoded `%2e` is a normal path byte, never a dot segment, so it is never treated as traversal.
  - "none": (Default) No normalization; dot and slash structure is preserved exactly as written.
  - "collapse_slashes": Collapse duplicate slashes in the path.
  - "dot_segments": Resolve . and .. segments per RFC 3986.
  - "both": Apply both collapse_slashes and dot_segments.

scheme_relative_handling
: How to handle URLs starting with "//". Defaults to "keep".
  - "keep": Parse using http but return scheme as NA and set status to "ok-scheme-relative".
  - "http": Assume http for parsing and output.
  - "https": Assume https for parsing and output.
  - "error": Treat scheme-relative URLs as invalid.

subdomain_levels_to_keep
: An integer or NULL. Determines how many levels of subdomains are kept, in addition to any 'www.' prefix handled by `www_handling`.
  - `NULL`: (Default) No specific subdomain stripping is performed beyond `www_handling`.
  - `0`: All subdomains are stripped. If `www_handling` preserved or added 'www.', it remains (e.g., 'www.sub.example.com' becomes 'www.example.com'; 'sub.example.com' becomes 'example.com').
  - `N > 0`: Keeps up to N levels of subdomains, counted from right-to-left (closest to the registered domain), in addition to any 'www.' prefix. E.g., if N=1, 'three.two.one.example.com' becomes 'one.example.com'; 'www.three.two.one.example.com' (post www_handling) becomes 'www.one.example.com'.

host_encoding
: How to present the host in `clean_url`. Defaults to "keep".
  - "keep": Leave the host as parsed (may preserve original case).
  - "idna": Convert Unicode host labels to Punycode (IDNA) for the cleaned URL.
  - "unicode": Decode Punycode labels to Unicode for the cleaned URL.

  Under `url_standard = "whatwg"` every value renders the UTS-46-mapped host, because mapping is part of WHATWG host parsing rather than a feature of the `idna` dial (`BÜCHER.example` presents as `bücher.example`; RUL-002). There `"keep"` preserves only whether the input was written as an A-label (`xn--...`), so `get_host()` and `get_domain()` agree on the same row. `"rfc3986"` and `NULL` are unaffected.

path_encoding How to present the path percent-encoding in clean_url — the readable-vs-browser rendering choice (the path analog of host_encoding). Defaults to "keep". This is an orthogonal presentation knob: it is independent of url_standard and layers on top of any profile (e.g. url_standard = "whatwg", path_encoding = "encode" emits the WHATWG-parsed path in browser form), exactly like host_encoding. Only "keep" preserves a profile's canonical identity path verbatim; "encode" and "decode" are presentation forms that may re-encode or decode reserved octets (so %2F may fold to a path-separating /), independent of whether a profile is set.

- "keep": Leave the path percent-encoding untouched (the path is preserved as written in the URL, so %2F stays %2F rather than decoding into a path-separating /). With no url_standard, rurl keeps its historical RFC-style percent-hex case canonicalization, so %2f becomes %2F. Under url_standard = "rfc3986", the profile's RFC 3986 §6.2.2.2 normalization applies: a triplet encoding an unreserved byte is decoded, every other triplet stays encoded with uppercased hex, so %7E becomes ~ while %2F stays %2F. Under url_standard = "whatwg", existing percent-triplet spelling is preserved byte-for-byte. Use "encode" to additionally normalize which bytes are encoded.
- "encode": The browser/percent-encoded rendering. Decodes the path first, then percent-encodes each segment (slashes preserved), so a readable non-ASCII path is emitted in its percent-encoded UTF-8 form.
- "decode": The readable rendering. Percent-decodes UTF-8 sequences in the path, so a percent-encoded segment is shown as readable text.

query_handling A character string controlling whether (and how) the query string is included in clean_url. Defaults to "drop", which preserves the historical query-free clean_url. The raw query result field is never affected by this option — it always reports the faithful original query.

- "drop": (Default) clean_url carries no query, exactly as before.
- "filter": Keep contentful params, dropping known trackers via a built-in denylist (e.g. utm_*, fbclid, gclid). params_drop extends the denylist; params_keep rescues names (winning over both the denylist and empty-dropping).
- "allow": Keep only params whose names match params_keep; all others are dropped. Here params_keep is an inclusion criterion only, not an empty-rescue.
- "keep": Keep every param, re-encoded into canonical form (not the verbatim original — that stays on the query field).

In every non-"drop" mode the surviving query is re-encoded canonically (uppercase percent-hex, spaces as %20) and appended after the path. The query is intentionally EXEMPT from case_handling (query values are case-sensitive — tokens, IDs, signatures), so under case_handling = "lower" or "upper" the clean_url is no longer uniformly cased: scheme/host/path fold but the query keeps its original case. Because clean_url is the canonical_join key, any non-"drop" mode also brings the query into that join key (so ?id=1 and ?id=2 stop collapsing, while utm-only differences still collapse under "filter").

params_keep Character vector of parameter-name globs (only * is special), or NULL (default).

In "filter" mode this is the rescue list; in "allow" mode it is the allowlist. Ignored in "drop"/"keep".

`params_drop` Character vector of parameter-name globs to add to the built-in denylist in "filter" mode, or `NULL` (default). Ignored in "drop"/"allow"/"keep".

`sort_params` Logical (default `FALSE`). When `TRUE`, surviving params are stably sorted by decoded key. Active in "filter"/"allow"/"keep".

`empty_param_handling`
One of "keep" (default) or "drop". "drop" removes empty-valued params (e.g. `?ref=`), except those rescued by `params_keep` in "filter" mode.

`params_case_sensitive`
Logical (default `FALSE`). Controls whether the denylist and `params_keep`/`params_drop` matching is case-sensitive.

`decode_plus` Logical (default `FALSE`). When `TRUE`, `+` in query values is treated as a space (HTML-form decoding) before percent-decoding. `FALSE` keeps `+` literal (RFC 3986 generic behavior).

`port_handling` A character string controlling whether the port appears in `clean_url`. Defaults to "exclude", today's only historical behavior. This knob is standalone and standard-independent (editorial, like `www_handling`) – `url_standard` never governs whether it may be set.

- "exclude": (Default) The port never appears in `clean_url`.
- "strip_all": Explicit alias of "exclude".
- "keep": Include the syntactic port when present, including a default port under `url_standard = "whatwg"`. This is an explicit non-parity override for callers that need the input's port spelling.
- "strip_default": Keep only non-default ports (using the same scheme-default table), independent of `url_standard`. Default-ness is judged on the scheme the input was **parsed** with, never on the scheme `protocol_handling` renders: `http://example.com:443/a` under `protocol_handling = "https"` keeps `:443`, and `http://example.com:80/a` drops `:80` (RFC 3986 §6.2.3; WHATWG URL Standard port state; RUL-016). This is the value `profile = "seo"` pins.

`scheme_policy` Controls whether scheme-less, host-shaped input is *accepted* (an input-acceptance axis, distinct from `protocol_handling`, which only controls how the scheme is *presented*, and from `url_standard`, which controls interpretation). Defaults to "infer".

- "infer": (Default) Fabricate `http://` for scheme-less host-shaped input (e.g. `example.com` parses as `http://example.com`), a browser-omnibox-style affordance. This is the historical behavior.
- "require": Reject scheme-less input — a scheme-less host-shaped value becomes `parse_status = "error"` rather than gaining a fabricated scheme. Use this for a strict, pure-parser posture. Note this governs only bare host input; scheme-relative `//host` input is governed separately by `scheme_relative_handling`.

`scheme_acceptance`
Which scheme *tokens* may enter parsing (a scheme-acceptance axis, distinct from `scheme_policy`, which governs scheme-*less* input, and from `url_standard`, which governs interpretation). Defaults to "web".

- "web": (Default) Only the curated web-scheme allowlist (`http`/`https`/`ftp`/`ftps`/`file`) is admitted; a scheme-bearing input outside it is `parse_status = "error"`. This is the historical, byte-for-byte compatible behavior.
- "general": Admit any syntactically valid scheme token and parse opaque (`mailto:x`), non-special (`foo://host`), and RFC-generic URLs. Requires an explicit `url_standard` (`"rfc3986"` or `"whatwg"`), which decides the interpretation; `general` with `url_standard = NULL` is an error. Non-special / opaque hosts receive no www-stripping, no domain/TLD derivation, and are never run through the IDNA/punycode helpers. A non-special scheme with no `//` is an *opaque path*: it has no authority, so `host`, `user`, `port` and the `domain`/`tld` columns are all `NA` and the entire remainder is the `path` (`query`/`fragment` are still split off). This includes `mailto:` — the recipient's `@` never re-triggers authority parsing. To decompose a `mailto:` recipient, use the accessors (`get_host()` / `get_domain()` / `get_user()`, ADR 0012 D7) or `get_mailto_recipients()`; those deliberately return a recipient's parts where this table presents `NA`, because a recipient domain is extraction metadata, not the URL's authority.

Note that `file:` is admitted under *both* values, including the default. A `file:` URL denotes local-filesystem access, and one with a non-empty host (`file://server/share/x`) is a UNC path on Windows, so dereferencing it reaches a remote SMB share. rurl parses `file:` URLs; it never opens them. Restricting schemes before anything dereferences them is the caller's job — see `SECURITY.md`.

`url_standard` Optional top-level standard profile: `NULL` (default), `"rfc3986"`, or `"whatwg"`. With `NULL` the behavior is exactly what the individual low-level options select (fully backward compatible). When set, it selects a coherent set of standard-conformant behaviors for the axes it governs — path percent/dot handling, the host IPv4/reg-name model, and `case_handling` — so callers do not have to hand-assemble the low-level knobs. Passing a governed low-level knob (`path_normalization` or `case_handling`) with a value the selected profile would not choose is an error; passing the value the profile would pick is accepted (only `case_handling = "lower_host"` is accepted under a selector — `"keep"`, `"lower"`, and `"upper"` all conflict, since `"lower"` also lowercases the path, which neither standard sanctions). Added as the last argument so existing positional calls keep their meaning; always pass it by name. Under `"whatwg"` the selector additionally recognizes a literal backslash as a path separator for WHATWG-special schemes (`http`/`https`/`ftp`) and nulls default ports in parse output; use `port_handling = "strip_default"` for spec-style clean URL port rendering. See `resolve_url` for `url_standard`-governed reference resolution. The selector does **not** govern whether `port_handling` may be set (it is a standalone editorial knob), nor does it govern `path_encoding` (an orthogonal path-presentation knob that layers on any profile), IDNA rendering, or query handling.

`engine` Optional **pslr** engine controlling which Public Suffix List backs domain / TLD / subdomain extraction: `NULL` (default) resolves against **pslr**'s session-global default list — exactly the historical behavior — while a `pslr::psl_engine()` snapshot resolves against that specific list, per request, without mutating any global state (never call `pslr::psl_use()` for this). Use it to pin a particular list version or to load an alternate list via `pslr::psl_engine(source = "path", path = ...)`. **Process-local:** an engine holds a C++ external pointer that does

not serialize across R sessions or parallel workers — build it in the process that uses it; never cache it to disk or send it to a worker (rebuild one per process instead). Only the domain-derived outputs (`domain`, `tld`, and the subdomain-trimmed host / `clean_url`) depend on it.

`profile` Optional named profile bundling several knobs at once: `NULL` (default; behaves exactly as the individual arguments select, fully backward compatible), `"browser"`, `"whatwg"`, `"rfc-syntax"`, `"seo"`, or the `"seo"` alias `"canonical"`. A profile is **separate** from `url_standard` (it bundles acceptance, interpretation, leniency, and canonicalization together) and expands only into arguments you did not supply explicitly — an explicit argument always overrides the profile. `"browser"` is a browser-*like* fix-up posture (http-prepending; not Chrome-faithful); `"whatwg"` is the absolute-URL no-base posture that *rejects* schemeless input (unlike a bare `url_standard = "whatwg"`); `"rfc-syntax"` is RFC 3986 generic syntax as parsing, not normalization (case and dot-segments are preserved); `"seo"`/`"canonical"` is rurl's origin-cleaning intent — a **lossy policy projection of a WHATWG-parsed URL** (ADR 0017), which claims no resource equivalence: `url_standard = "whatwg"` underneath (which also resolves `./..` folder segments), https, a Unicode host regardless of the input spelling, strip www / trailing slash / index page, drop the whole query, and drop a **default** port only (`port_handling = "strip_default"`: a non-default port names a different origin and survives). Inspect the resolved bundle with `url_profile`. Also accepted by `canonical_join` (forwarded through its `...`).

`credential_handling`

How `clean_url` treats a URL whose parsed authority carried a userinfo delimiter (`user@`, `user:password@`, a bare `@`, or a repeated `@`). Defaults to "strip". A policy dial on the clean surface (ADR 0017, mutation-table row 12; RUL-001), not a standards axis: it composes with every `url_standard`, including `NULL`, and never touches the `user` / `password` columns, `parse_status`, the diagnostics, `serialize_url` or `get_url_key`.

- "strip": (Default) The userinfo is dropped and the rest of the URL is emitted, exactly as before this argument existed.
- "reject": `clean_url` is `NA` for such a row. RFC 3986 section 3.2.1 deprecates the `user:password` form and lets an application reject it; sections 7.5 and 7.6 describe the credential leak and the `https://example.com@evil.example/` semantic attack a silently collapsed clean URL would hide. Use this when a cleaned URL that *looks* like the credential-free original would be misleading.

There is no "keep": `serialize_url` already preserves credentials under both standards, and `format_url` redacts them for display.

Value

A data.frame with one row per URL and the same fields returned by `safe_parse_url`. Invalid inputs return NA fields with `parse_status = "error"`. Names on `url` are not carried into the result: the frame always has ordinary sequential row names, matching the accessors (`get_host` and friends), which return unnamed vectors.

Examples

```
safe_parse_urls(c("example.com", "https://www.example.com/path"))
```

`serialize_url` *Serialize URLs to a standard's own full-string form*

Description

Renders each URL as the selected standard would serialize it: the **full string**, credentials and fragment included, with a present-but-empty ? or # delimiter preserved. This is rurl's *standard serialization* surface, and it is deliberately **not** `get_clean_url()`.

Usage

```
serialize_url(
  url,
  standard = c("whatwg", "rfc3986"),
  form = c("source", "normalized"),
  engine = NULL
)
```

Arguments

`url`	A character vector of URLs.
`standard`	The standard to serialize to: `"whatwg"` (default) or `"rfc3986"`. Unlike the parse surface, `NULL` is not accepted: the parse surface's `url_standard = NULL` is a frozen legacy profile that names no standard (ADR 0007), so there is nothing to serialize *as*. Passing `NULL` is an error rather than a silent `"whatwg"`.
`form`	For `standard = "rfc3986"` only, the RFC posture: `"source"` (default, source-preserving) or `"normalized"`. Ignored for `"whatwg"`, whose serializer has a single spec-defined form.
`engine`	Optional `psl_engine` object from `pslr::psl_engine()` for per-request Public Suffix List resolution. `NULL` (default) uses the session-global engine.

Value

A character vector the same length as `url`. `NA_character_` for input the selected standard's parser does not accept.

Which surface you want

`serialize_url()` answers *"what does this URL look like under the URL Standard / RFC 3986?"*. `get_clean_url()` answers *"what is the canonical, tidied form of this URL for SEO or deduplication?"*. They are different products, not two settings of one:

- `serialize_url()` preserves userinfo, the fragment, and empty delimiters; takes **no** presentation options; and is the substrate rurl's conformance claims are measured on.

- `get_clean_url()` drops userinfo and the fragment by design, and is driven by cleaning policy (`www_handling`, `trailing_slash_handling`, `index_page_handling`, query filtering, port handling, and so on).

Because a standard serialization is an *identity*, `serialize_url()` accepts no presentation arguments at all. There is no `port_handling`, `trailing_slash_handling` or `path_encoding` to pass; asking a serializer to strip a trailing slash would be a category error.

Parse posture

Each standard is parsed under its own spec posture – the `"whatwg"` and `"rfc-syntax"` profiles (see `url_profile()`). Both accept any scheme, and both **require** a scheme: neither standard defines a base-URL-free parse of `example.com/x`, so scheme-less input returns `NA` rather than being silently upgraded to `https://`. Input that the standard's parser rejects also returns `NA`.

Standards and forms

`standard = "whatwg"` The WHATWG URL Standard's URL serializer (`#concept-url-serializer`). Spec-exact, which makes credentials lossy in one direction: WHATWG appends credentials only when the username or password is non-empty, so `http://@h/` serializes as `http://h/` and `http://u:@h/` as `http://u@h/`. Both are pinned by the Web Platform Tests. `form` is ignored.

`standard = "rfc3986", form = "source"` RFC 3986 section 5.3 component recomposition with **no** normalization: source bytes are preserved and the undivided `userinfo` slice is emitted verbatim (RFC 3986 has no username/password split), so every credential spelling `u@`, `u:@`, `:p@`, `@` – survives.

`standard = "rfc3986", form = "normalized"` Adds RFC 3986 section 6.2.2 syntax-based normalization (scheme and host case, percent-encoding triplet case and unreserved-octet decoding, dot-segment removal) and the section 6.2.3 default-port elision.

Both RFC forms are exposed because choosing one would forfeit either the round-trip oracle (`source`) or the normalized comparison substrate (`normalized`).

See Also

`get_clean_url()` for the cleaning surface, `safe_parse_url()` for the parsed components, and `url_profile()` for the parse postures used here.

Examples

```
# The fragment and credentials survive; clean_url drops both by design.
serialize_url("http://user:pw@Example.COM:80/a/../b?q=1#frag")
get_clean_url("http://user:pw@Example.COM:80/a/../b?q=1#frag")

# A present-but-empty delimiter carries information and is preserved.
serialize_url(c("http://example.com/", "http://example.com/#",
                "http://example.com/?"))

# RFC 3986: source-preserving versus normalized.
serialize_url("HTTP://Example.COM:80/a/%7Euser/../x", standard = "rfc3986")
serialize_url("HTTP://Example.COM:80/a/%7Euser/../x", standard = "rfc3986",
```

```
              form = "normalized")

# Any scheme is accepted; no scheme is not.
serialize_url(c("urn:ietf:rfc:2648", "mailto:a@b.com", "foo://h/x"))
serialize_url("example.com/x")
```

`url_join` *Identity-keyed URL joins*

Description

Six joins that match rows on URL *identity* rather than on string equality. Each side names one URL column; both sides are keyed with one immutable `url_key_policy()`, and rows pair up when their comparison keys are equal.

Usage

```
url_inner_join(
  x,
  y,
  by,
  policy = url_key_policy(),
  suffix = c(".x", ".y"),
  key_name = NULL,
  relationship = "none",
  multiple = "all",
  invalid = "keep",
  warnings = "allow",
  engine = NULL
)

url_left_join(
  x,
  y,
  by,
  policy = url_key_policy(),
  suffix = c(".x", ".y"),
  key_name = NULL,
  relationship = "none",
  multiple = "all",
  invalid = "keep",
  warnings = "allow",
  engine = NULL
)

url_right_join(
```

```
  x,
  y,
  by,
  policy = url_key_policy(),
  suffix = c(".x", ".y"),
  key_name = NULL,
  relationship = "none",
  multiple = "all",
  invalid = "keep",
  warnings = "allow",
  engine = NULL
)

url_full_join(
  x,
  y,
  by,
  policy = url_key_policy(),
  suffix = c(".x", ".y"),
  key_name = NULL,
  relationship = "none",
  multiple = "all",
  invalid = "keep",
  warnings = "allow",
  engine = NULL
)

url_semi_join(
  x,
  y,
  by,
  policy = url_key_policy(),
  key_name = NULL,
  relationship = "none",
  invalid = "keep",
  warnings = "allow",
  engine = NULL
)

url_anti_join(
  x,
  y,
  by,
  policy = url_key_policy(),
  key_name = NULL,
  relationship = "none",
  invalid = "keep",
  warnings = "allow",
```

```
  engine = NULL
)
```

Arguments

x, y	Data frames to join.
by	The URL columns to key on: either one column name present on both sides ("URL"), or the named form c(x_col = "y_col") when they differ.
policy	A rurl_url_key_policy from url_key_policy(), applied symmetrically to both sides. Side-specific rules are prohibited: equality has to stay symmetric and transitive.
suffix	Length-2 character vector disambiguating column names present on both sides. Default c(".x", ".y"). If the result would still contain a duplicate name, the join errors rather than repairing it silently.
key_name	Optional column name under which to expose the comparison key. NULL (default) hides it. The exposed value is the classed key from get_url_key(), never a URL-looking string, and a name that collides with an output column is an error.
relationship	Cardinality you assert about matching keys, checked *before* the result is materialized: "none" (default, no check), "one-to-one", "one-to-many", "many-to-one", or "many-to-many" (no constraint, declared explicitly).
multiple	How many y rows a matching x row may take: "all" (default, lossless) or the separately named lossy narrowings "first" / "last", which take the first or last match in y row order.
invalid	What to do with rows that cannot be keyed: "keep" (default), "drop" or "error".
warnings	What to do with rows that parsed with a warning: "allow" (default), "reject" (ineligible to match, but retained) or "error".
engine	Optional psl_engine object from pslr::psl_engine() for per-request Public Suffix List resolution. NULL (default) uses the session-global engine. It cannot affect the comparison key – identity frames no public-suffix component – but it can affect which rows count as warning rows under warnings = "reject".

Value

A data frame built by row-slicing x, so x's column types and subclass survive. url_inner_join(), url_left_join(), url_right_join() and url_full_join() return x's columns followed by y's, disambiguated by suffix; url_semi_join() and url_anti_join() return x's columns only. A zero-row result is built by the same path, so it carries the complete typed schema.

Why not a plain join

Joining data frames on raw URL strings misses http://example.com:80/a against http://example.com/a. Joining them on a *cleaned* string overmatches instead, because cleaning is a display policy: it can strip a trailing slash, a query parameter or a www. that genuinely distinguished two resources. These joins use get_url_key(), so what matches is what rurl considers the same resource – and no cleaning or display option can change that.

Row order

Order is part of the contract, not an artifact of the implementation:

`url_inner_join` matching pairs in `x` order, `y` matches in `y` order within each `x` row.

`url_left_join` every `x` row in `x` order; unmatched `x` rows carry a missing `y` payload typed from `y`'s own columns.

`url_right_join` the exact mirror: every `y` row in `y` order, `x` matches in `x` order.

`url_full_join` the left-join result, then the `y` rows it never consumed, in `y` order.

`url_semi_join` each `x` row with at least one match, once, `x` columns only.

`url_anti_join` each `x` row with no match, once, `x` columns only.

Duplicate keys expand as a Cartesian product. Rows are never silently discarded to "resolve" a duplicate – multiplicity is a fact you declare with `relationship` or narrow with `multiple`.

Rows that cannot be keyed

A URL the standard cannot parse, or a missing or empty one, has no identity and never matches – not even another unparseable URL. `invalid` decides what happens to those rows: `"keep"` (default) leaves them in, unmatched, so a left join still returns them; `"drop"` removes them before matching; `"error"` refuses the join and reports the offending row positions.

`warnings` is a separate axis for rows that *did* parse but carry a note – userinfo on a scheme-less input, or a host whose public-suffix annotation did not resolve. `"allow"` (default) matches them normally, `"reject"` makes them ineligible to match without removing them, and `"error"` refuses the join.

`url_anti_join()` keeps non-keyable `x` rows, because a row that cannot match anything is exactly what an anti join asks for.

Conditions

Failures raise typed conditions – `rurl_url_join_input_error`, `rurl_url_join_policy_error`, `rurl_url_join_suffix_error`, `rurl_url_join_key_name_error`, `rurl_url_join_relationship_error`, `rurl_url_join_invalid_error` and `rurl_url_join_warning_error`, all inheriting from `rurl_url_join_error` – so they can be caught precisely. Messages report row *positions* and truncated keys, never URL content, so a credential in the input cannot leak into an error message.

See Also

`get_url_key()` and `url_key_policy()` for the identity model, and `canonical_join()` for the legacy join that matches on cleaned strings.

Examples

```
pages <- data.frame(
  URL = c("http://example.com:80/a", "https://example.com/b",
          "http://example.com/c?", "not a url"),
  clicks = c(10, 20, 30, 40),
  stringsAsFactors = FALSE
```

```
)
meta <- data.frame(
  URL = c("http://example.com/a", "http://example.com/b",
          "http://example.com/c"),
  title = c("A", "B", "C"),
  stringsAsFactors = FALSE
)

# Only row 1 matches: `:80` is redundant under http, but http is not https,
# and a present-but-empty query is not the same resource as no query.
url_inner_join(pages, meta, by = "URL")

# Every left row survives, unmatched ones with a typed missing payload.
url_left_join(pages, meta, by = "URL")

# Rows that could not be parsed at all.
url_anti_join(pages, meta, by = "URL")

# Expose the key you matched on.
url_inner_join(pages, meta, by = "URL", key_name = "key")

# Relaxing scheme equality brings row 2 in.
url_inner_join(pages, meta, by = "URL",
               policy = url_key_policy(scheme_equality = "http_https"))
```

`url_key_policy` *Comparison-key policy*

Description

Builds the immutable, versioned policy object that governs URL *identity* for `get_url_key()` and the url_join family. One policy is applied symmetrically to both sides of every comparison, because equality has to stay symmetric and transitive.

Usage

```
url_key_policy(
  standard = c("whatwg", "rfc3986"),
  scheme_equality = c("exact", "http_https", "http_https_missing")
)
```

Arguments

`standard`	The standard whose identity semantics apply: `"whatwg"` (default) or `"rfc3986"`. Unlike the parse surface, `NULL` is not accepted – an unnamed standard cannot freeze key bytes.

`scheme_equality`
: How strictly schemes compare. `"exact"` (default) compares the normalized scheme identity. `"http_https"` additionally collapses `http` and `https` into one class, so `http://h/` and `https://h/` compare equal; every other scheme stays exact, including the `ws`/`wss` pair. `"http_https_missing"` is accepted by the vocabulary but not implemented, and errors – see Details.

Details

`scheme_equality = "http_https_missing"` would additionally collapse "no scheme written" into the `http`/`https` class. It errors rather than guessing, because the pair it would have to equate also differs on whether an authority delimiter (`//`) was present, which rurl frames as independent identity. Collapsing that too is a contract change, not an implementation detail, so the mode refuses instead of silently picking a side.

Value

An object of class `rurl_url_key_policy`.

Identity is not presentation

A comparison key is derived from the URL's canonical identity state – after the selected standard has interpreted it, and *before* any cleaning or display transform. No cleaning option can reach it. `www_handling`, `case_handling`, `trailing_slash_handling`, `index_page_handling`, `path_encoding`, `host_encoding`, `port_handling`, query cleaning and every `url_profile()` bundle are structurally incapable of changing a key byte. That is the point: two URLs that a cleaning profile happens to render alike are not thereby the same resource.

What the key does and does not distinguish

Framed as identity: the scheme (and, separately, whether one was written at all), the authority delimiter, the host and its kind, the port, the path and its kind, and the query – order and duplicates significant.

Excluded by contract: the fragment and any userinfo. Neither identifies a web resource, so `http://u:pw@h/p#frag` and `http://h/p` mint the same key. Their structural state is still available from `safe_parse_url()` and the diagnostics helpers.

Ports normalize only where the standard makes them redundant: an explicit `:80` under `http` and `:443` under `https` compare equal to no port at all. Every other default stays literal, so `ftp://h:21/` and `ftp://h/` are distinct, and an inferred scheme normalizes nothing (`h.com:80/` is not `http://h.com/`).

Versioning

The policy carries a key version and a schema version, and both travel *inside* the framed key bytes. A key minted under different semantics can therefore never compare equal to one minted here, so no release can silently reinterpret a persisted key.

See Also

`get_url_key()` for the key itself, and url_join for the joins that consume it.

Examples

```
url_key_policy()

# Identity under one policy ...
get_url_key(c("http://example.com/", "https://example.com/"))

# ... and under a relaxed scheme mode.
p <- url_key_policy(scheme_equality = "http_https")
k <- get_url_key(c("http://example.com/", "https://example.com/"), p)
k[1] == k[2]
```

`url_profile` *Inspect a named parsing profile*

Description

Expands a named `profile` into the resolved bundle of low-level parser knobs it sets, running the exact same resolution the parse functions (`safe_parse_url`, `safe_parse_urls`, `get_clean_url`) apply when you pass `profile`. It never parses a URL; it answers "what does this profile actually do, and did my explicit overrides change it?".

Usage

```
url_profile(profile = NULL, ...)
```

Arguments

`profile`	A single profile name: one of `"browser"`, `"whatwg"`, `"rfc-syntax"`, `"seo"`, or the `"seo"` alias `"canonical"`.
`...`	Optional explicit knob overrides (e.g. `scheme_policy = "require"`), named as in `safe_parse_url`. Each override that the profile also sets replaces the profile's value and marks the result `customized`.

Details

Profiles are inspectable sugar that bundle the parser's acceptance, interpretation, leniency, and canonicalization axes under one name. Explicit arguments always override the profile (the iron rule); when any override is supplied, the resolved result is flagged `customized = TRUE` and rurl no longer claims the result matches the named profile exactly.

The recognized profiles are `"browser"` (browser-*like* http-prepending fix-up posture; not Chrome-faithful), `"whatwg"` (absolute-URL, no-base spec posture that *rejects* scheme-less input), `"rfc-syntax"` (RFC 3986 generic syntax as *parsing*, not normalization: case and dot-segments are preserved), and `"seo"` (rurl's origin-cleaning intent; `"canonical"` is an alias resolving identically to `"seo"`).

`"seo"` delivers rurl's definition of a *clean URL*: a **lossy policy projection** of a WHATWG-parsed URL (ADR 0017), never a separate, weaker construction. It selects `url_standard = "whatwg"` as the identity underneath — which is also what resolves `.` and `..` folder segments — and `host_encoding`

= "unicode", so the host is canonical in one direction regardless of whether the input spelled it in Unicode or Punycode. The remaining knobs are the projection: https, strip www, trailing slash and index page, **drop the whole query**, and drop a *default* port only (port_handling = "strip_default"): a non-default port such as :8080 names a different origin and survives (RFC 3986 §6.2.3; WHATWG URL Standard port state; RFC 6454 §4; RUL-016). The result makes no claim of resource equivalence and is not an HTML canonical URL — forcing https, stripping www and dropping the query can each change the resource addressed. As always, an explicit argument overrides the bundle, so query_handling = "filter" still buys tracker-only removal.

Value

A named list of the resolved knob → value pairs the profile sets (the same shape as an internal profile bundle), plus a trailing logical customized element.

Normalization versus editorial knobs in "seo"

The bundle applies two kinds of transform, and a reader cannot tell them apart from the profile name alone (RUL-017). **Normalization** is what a standard says yields the same resource: the WHATWG parse with its dot-segment resolution (url_standard = "whatwg"), host case folding and UTS #46 rendering (host_encoding = "unicode"), and removal of a *default* port only (port_handling = "strip_default"). **Editorial** is a claim about how the site is configured that no standard settles: forcing https (protocol_handling), folding www. (www_handling), stripping a trailing slash (trailing_slash_handling) and a terminal index page (index_page_handling), and dropping the whole query (query_handling = "drop"). Each editorial knob can change the addressed resource, and each is in the bundle on purpose: an SEO bundle exists to say which URL a page should be known as, which is a site policy. Pass any knob explicitly to turn it off, and use get_url_key when you need resource identity rather than a display string.

See Also

safe_parse_url, get_scheme_class

Examples

```
url_profile("browser")
url_profile("seo")
# canonical is an alias of seo:
identical(url_profile("canonical"), url_profile("seo"))
# explicit overrides win and flag the result customized:
url_profile("browser", scheme_policy = "require")
```

Index