Package 'rurl'

June 19, 2026

Type Package

Title Parse, Clean, and Normalize URLs

Version 1.2.0

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.
The package includes a processed copy of the Public Suffix List from
<https://publicsuffix.org> for domain extraction.

License MIT + file LICENSE

Encoding UTF-8

Collate 'utils.R' 'domain.R' 'path-query.R' 'parse-phases.R' 'parse.R'
'accessors.R' 'canonical_join.R' 'zzz.R'

Imports utils, curl, stringi, punycoder (>= 1.0.0)

URL https://github.com/bart-turczynski/rurl

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

Suggests testthat (>= 3.0.0), knitr, rmarkdown, withr

Config/testthat/edition 3

Depends R (>= 3.5)

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Bart Turczynski [aut, cre]

Maintainer Bart Turczynski <bartek+rurl@turczynski.pl>

Repository CRAN

Date/Publication 2026-06-19 15:50:02 UTC

Contents

canonical_join . . . 2
get_clean_url . . . 4
get_domain . . . 7
get_fragment . . . 9
get_host . . . 9
get_parse_status . . . 11
get_password . . . 13
get_path . . . 13
get_port . . . 14
get_query . . . 15
get_scheme . . . 16
get_subdomain . . . 16
get_tld . . . 18
get_user . . . 18
get_userinfo . . . 19
rurl_cache_config . . . 20
rurl_cache_info . . . 21
rurl_clear_caches . . . 21
safe_parse_urls . . . 22

Index **25**

`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).

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.

Examples

```
A <- data.frame(
  URL = c("http://Example.com/Page", "http://example.com/Other"),
  ValA = 1:2, stringsAsFactors = FALSE
)
B <- data.frame(
  URL = c("https://www.example.com/Page/", "http://example.com/Miss"),
  ValB = c("x", "y"), stringsAsFactors = FALSE
)

canonical_join(
  A, B,
  protocol_handling = "strip",
  www_handling = "strip",
  case_handling = "lower_host",
  trailing_slash_handling = "strip"
)
```

`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. The result is a normalized canonical key composed of scheme, host, and path only; port, query, fragment, and userinfo are intentionally excluded (use `get_port`, `get_query`, `get_fragment`, or `get_userinfo` for those).

Usage

```
get_clean_url(
  url,
  protocol_handling = "keep",
  www_handling = "none",
  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"
)
```

Arguments

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

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep".

- "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added.
- "none": If a 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.

`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".

- "none": (Default) No normalization.
- "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 host as parsed by curl (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.

`path_encoding` How to handle percent-encoding in the path for 'clean_url'. Defaults to "keep".

- "keep": Leave the path percent-encoding untouched.
- "encode": Normalize by decoding first, then percent-encoding each segment (slashes preserved).
- "decode": Percent-decode UTF-8 sequences in the path.

Value

A character vector of cleaned URLs.

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"
```

`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")
)
```

Arguments

`url` A character vector of URLs.

`protocol_handling`

A character string specifying how to handle protocols. Defaults to "keep".

- "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added.
- "none": If a 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".

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.

Usage

```
get_fragment(url, protocol_handling = "keep")
```

Arguments

`url`	A character vector of URLs.
`protocol_handling`	A character string specifying how to handle protocols. Defaults to "keep". • "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added. • "none": If a 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".

Value

A character vector of fragments.

Examples

```
get_fragment("http://example.com/path#section")
```

get_host *Get URL hosts*

Description

Extracts the host component of a URL.

Usage

```
get_host(
  url,
  protocol_handling = "keep",
  www_handling = "none",
  subdomain_levels_to_keep = NULL,
  case_handling = c("lower", "keep", "upper", "lower_host")
)
```

Arguments

`url` A character vector of URLs.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep".

- "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added.
- "none": If a 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'.

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

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_parse_status` *Get the parse status of URLs*

Description

Get the parse status of URLs

Usage

```
get_parse_status(
  url,
  protocol_handling = "keep",
  www_handling = "none",
  subdomain_levels_to_keep = NULL
)
```

Arguments

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

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep".

- "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added.

- "none": If a 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'.

Value

A character vector with the parse status of each URL.

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_password *Get URL passwords*

Description

Extracts the password component of a URL.

Usage

```
get_password(url, protocol_handling = "keep")
```

Arguments

`url`	A character vector of URLs.
`protocol_handling`	A character string specifying how to handle protocols. Defaults to "keep". • "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added. • "none": If a 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".

Value

A character vector of passwords.

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")
)
```

Arguments

`url`	A character vector of URLs.
`protocol_handling`	A character string specifying how to handle protocols. Defaults to "keep". • "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added. • "none": If a 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.

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")
```

Arguments

`url`	A character vector of URLs.
`protocol_handling`	A character string specifying how to handle protocols. Defaults to "keep". • "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added. • "none": If a 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".

Value

An integer vector of ports.

Examples

```
get_port("http://example.com:8080/path")
```

---

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
)
```

Arguments

`url`	A character vector of URLs.
`protocol_handling`	A character string specifying how to handle protocols. Defaults to "keep". • "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added. • "none": If a 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 and format="list", percent-decodes keys/values.

Value

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

Examples

```
get_query("http://example.com/path?a=1&b=2")
get_query("http://example.com/path?a=1&b=2", format = "list")
```

get_scheme *Get URL schemes*

Description

Extracts the scheme (protocol) of a URL.

Usage

```
get_scheme(url, protocol_handling = "keep")
```

Arguments

`url`	A character vector of URLs.
`protocol_handling`	A character string specifying how to handle protocols. Defaults to "keep". • "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added. • "none": If a 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".

Value

A character vector of URL schemes.

Examples

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

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")
)
```

Arguments

`url`	A character vector of URLs.
`protocol_handling`	A character string specifying how to handle protocols. Defaults to "keep". • "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added. • "none": If a 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.

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"))
```

Arguments

url	A character vector of URLs.
source	Which TLD source to use: "all", "icann", or "private".

Value

A character vector of TLDs.

Examples

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

---

get_user *Get URL user names*

Description

Extracts the user component of a URL.

Usage

```
get_user(url, protocol_handling = "keep")
```

Arguments

`url` A character vector of URLs.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep".

- "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added.
- "none": If a 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".

Value

A character vector of user names.

`get_userinfo` *Get URL userinfo*

Description

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

Usage

```
get_userinfo(url, protocol_handling = "keep")
```

Arguments

`url` A character vector of URLs.

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep".

- "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added.
- "none": If a 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".

Value

A character vector of userinfo values.

`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,
  domain = NULL,
  tld = NULL,
  puny_encode = NULL,
  puny_decode = NULL,
  max_full_parse = NULL
)
```

Arguments

`full_parse`	Logical; enable/disable the full URL parse cache.
`domain`	Logical; enable/disable the registered-domain cache.
`tld`	Logical; enable/disable the TLD-extraction 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, it is reset before the next new entry is stored, so its peak size never exceeds the bound; the default of `Inf` preserves the historical unbounded behavior. The `domain`, `tld`, `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(domain = 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`, `domain`, `tld`, `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 or if you've updated the PSL data.

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")
)
```

Arguments

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

`protocol_handling`
A character string specifying how to handle protocols. Defaults to "keep".

- "keep": If a scheme exists (http, https, ftp, ftps), it's used. If no scheme, "http://" is added.
- "none": If a 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".

- "none": (Default) No normalization.
- "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 host as parsed by curl (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.

`path_encoding`
: How to handle percent-encoding in the path for 'clean_url'. Defaults to "keep".
  - "keep": Leave the path percent-encoding untouched.
  - "encode": Normalize by decoding first, then percent-encoding each segment (slashes preserved).
  - "decode": Percent-decode UTF-8 sequences in the path.

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"`.

Examples

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

Index

canonical_join, 2

get_clean_url, 4
get_domain, 7
get_fragment, *4*, 9
get_host, 9
get_parse_status, 11
get_password, 13
get_path, 13
get_port, *4*, 14
get_query, *4*, 15
get_scheme, 16
get_subdomain, 16
get_tld, 18
get_user, 18
get_userinfo, *4*, 19

rurl_cache_config, 20, *21*
rurl_cache_info, *20*, 21
rurl_clear_caches, *20, 21*, 21

safe_parse_url, *22*, *24*
safe_parse_urls, *2, 3*, 22