Package 'urltools'

October 12, 2022

Type Package

Title Vectorised Tools for URL Handling and Parsing

Version 1.7.3

Date 2019-04-14

Author Os Keyes [aut, cre], Jay Jacobs [aut, cre], Drew Schmidt [aut],
Mark Greenaway [ctb], Bob Rudis [ctb], Alex Pinto [ctb], Maryam Khezrzadeh [ctb], Peter Meilstrup [ctb],
Adam M. Costello [cph], Jeff Bezanson [cph], Peter Meilstrup [ctb], Xueyuan Jiang [ctb]

Maintainer Os Keyes <ironholds@gmail.com>

Description A toolkit for all URL-handling needs, including encoding and decoding, parsing, parameter extraction and modification. All functions are designed to be both fast and entirely vectorised. It is intended to be useful for people dealing with web-related datasets, such as server-side logs, although may be useful for other situations involving large sets of URLs.

License MIT + file LICENSE

LazyData TRUE

LinkingTo Rcpp

Imports Rcpp, methods, triebeard

Suggests testthat, knitr

URL https://github.com/Ironholds/urltools/

BugReports https://github.com/Ironholds/urltools/issues

VignetteBuilder knitr

RoxygenNote 6.1.1

Encoding UTF-8

Depends R (>= 2.10)

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-04-14 23:02:47 UTC

R topics documented:

`domain` *Get or set a URL's domain*

Description

as in the lubridate package, individual components of a URL can be both extracted or set using the relevant function call - see the examples.

Usage

```
domain(x)

domain(x) <- value
```

Arguments

`x`	a URL, or vector of URLs
`value`	a replacement value (or vector of replacement values) for x's scheme.

See Also

`scheme`, `port`, `path`, `parameters` and `fragment` for other accessors.

Examples

```
#Get a component
example_url <- "http://cran.r-project.org/submit.html"
domain(example_url)

#Set a component
domain(example_url) <- "en.wikipedia.org"
```

`fragment` *Get or set a URL's fragment*

Description

as in the lubridate package, individual components of a URL can be both extracted or set using the relevant function call - see the examples.

Usage

```
fragment(x)

fragment(x) <- value
```

Arguments

`x`	a URL, or vector of URLs
`value`	a replacement value (or vector of replacement values) for x's fragment. If NULL, the fragment will be removed entirely.

See Also

`scheme`, `domain`, `port`, `path` and `parameters` for other accessors.

Examples

```
#Get a component
example_url <- "http://en.wikipedia.org/wiki/Aaron_Halfaker?debug=true#test"
fragment(example_url)

#Set a component
fragment(example_url) <- "production"

#Remove a component
fragment(example_url) <- NULL
```

host_extract *Extract hosts*

Description

host_extract extracts the host from a vector of domain names. A host isn't the same as a domain - it could be the subdomain, if there are one or more subdomains. The host of en.wikipedia.org is en, while the host of wikipedia.org is wikipedia.

Usage

```
host_extract(domains)
```

Arguments

domains	a vector of domains, retrieved through url_parse or domain.

Value

a data.frame of two columns: domain, with the original domain names, and host, the identified host from the domain.

Examples

```
# With subdomains
has_subdomain <- domain("https://en.wikipedia.org/wiki/Main_Page")
host_extract(has_subdomain)

# Without
no_subdomain <- domain("https://ironholds.org/projects/r_shiny/")
host_extract(no_subdomain)
```

parameters *Get or set a URL's parameters*

Description

as in the lubridate package, individual components of a URL can be both extracted or set using the relevant function call - see the examples.

Usage

```
parameters(x)

parameters(x) <- value
```

Arguments

x	a URL, or vector of URLs
value	a replacement value (or vector of replacement values) for x's parameters. If NULL, the parameters will be removed entirely.

See Also

scheme, domain, port, path and fragment for other accessors.

Examples

```
# Get the parameters
example_url <- "http://en.wikipedia.org/wiki/Aaron_Halfaker?debug=true"
parameters(example_url)

# Set the parameters
parameters(example_url) <- "debug=false"

# Remove the parameters
parameters(example_url) <- NULL
```

---

param_get *get the values of a URL's parameters*

Description

URLs can have parameters, taking the form of name=value, chained together with & symbols. param_get, when provided with a vector of URLs and a vector of parameter names, will generate a data.frame consisting of the values of each parameter for each URL.

Usage

```
param_get(urls, parameter_names = NULL)
```

Arguments

urls	a vector of URLs
parameter_names	a vector of parameter names. If NULL (default), will extract all parameters that are present.

Value

a data.frame containing one column for each provided parameter name. Values that cannot be found within a particular URL are represented by an NA.

See Also

url_parse for decomposing URLs into their constituent parts and param_set for inserting or modifying key/value pairs within a query string.

Examples

```
#A very simple example
url <- "https://google.com:80/foo.php?this_parameter=selfreferencing&hiphop=awesome"
parameter_values <- param_get(url, c("this_parameter","hiphop"))
```

---

param_remove *Remove key-value pairs from query strings*

Description

URLs often have queries associated with them, particularly URLs for APIs, that look like ?key=value&key=value&key=valu
param_remove allows you to remove key/value pairs while leaving the rest of the URL intact.

Usage

```
param_remove(urls, keys)
```

Arguments

urls	a vector of URLs. These should be decoded with url_decode but don't have to have been otherwise processed.
keys	a vector of parameter keys to remove.

Value

the original URLs but with the key/value pairs specified by keys removed. If the original URL is NA, NA will be returned; if a specified key is NA, nothing will be done with it.

See Also

param_set to modify values associated with keys, or param_get to retrieve those values.

Examples

```
# Remove multiple parameters from a URL
param_remove(urls = "https://en.wikipedia.org/wiki/api.php?action=list&type=query&format=json",
             keys = c("action","format"))
```

param_set *Set the value associated with a parameter in a URL's query.*

Description

URLs often have queries associated with them, particularly URLs for APIs, that look like ?key=value&key=value&key=valu
param_set allows you to modify key/value pairs within query strings, or even add new ones if they don't exist within the URL.

Usage

```
param_set(urls, key, value)
```

Arguments

urls	a vector of URLs. These should be decoded (with url_decode) but do not have to have been otherwise manipulated.
key	a string representing the key to modify the value of (or insert wholesale if it doesn't exist within the URL).
value	a value to associate with the key. This can be a single string, or a vector the same length as urls

Value

the original vector of URLs, but with modified/inserted key-value pairs. If the URL is NA, the returned value will be - if the key or value are, no insertion will be made.

See Also

param_get to retrieve the values associated with multiple keys in a vector of URLs, and param_remove to strip key/value pairs from a URL entirely.

Examples

```
# Set a URL parameter where there's already a key for that
param_set("https://en.wikipedia.org/api.php?action=query", "action", "pageinfo")

# Set a URL parameter where there isn't.
param_set("https://en.wikipedia.org/api.php?list=props", "action", "pageinfo")
```

path *Get or set a URL's path*

Description

as in the lubridate package, individual components of a URL can be both extracted or set using the relevant function call - see the examples.

Usage

```
path(x)

path(x) <- value
```

Arguments

x	a URL, or vector of URLs
value	a replacement value (or vector of replacement values) for x's path. If NULL, the path will be removed entirely.

See Also

scheme, domain, port, parameters and fragment for other accessors.

Examples

```
# Get the path
example_url <- "http://cran.r-project.org:80/submit.html"
path(example_url)

# Set the path
path(example_url) <- "bin/windows/"

# Remove the path
path(example_url) <- NULL
```

port *Get or set a URL's port*

Description

as in the lubridate package, individual components of a URL can be both extracted or set using the relevant function call - see the examples.

Usage

```
port(x)

port(x) <- value
```

Arguments

x	a URL, or vector of URLs
value	a replacement value (or vector of replacement values) for x's port. If NULL, the port will be entirely removed.

See Also

scheme, domain, path, parameters and fragment for other accessors.

Examples

```
# Get the port
example_url <- "http://cran.r-project.org:80/submit.html"
port(example_url)

# Set the port
port(example_url) <- "12"

# Remove the port
port(example_url) <- NULL
```

---

puny_encode *Encode or Decode Internationalised Domains*

Description

puny_encode and puny_decode implement the encoding standard for internationalised (non-ASCII) domains and subdomains. You can use them to encode UTF-8 domain names, or decode encoded names (which start "xn–"), or both.

Usage

```
puny_encode(x)

puny_decode(x)
```

Arguments

x	a vector of URLs. These should be URL decoded using url_decode.

Value

a CharacterVector containing encoded or decoded versions of the entries in `x`. Invalid URLs (ones that are `NA`, or ones that do not successfully map to an actual decoded or encoded version) will be returned as `NA`.

See Also

`url_decode` and `url_encode` for percent-encoding.

Examples

```
# Encode a URL
puny_encode("https://www.bücher.com/foo")

# Decode the result, back to the original
puny_decode("https://www.xn--bcher-kva.com/foo")
```

---

scheme *Get or set a URL's scheme*

Description

as in the lubridate package, individual components of a URL can be both extracted or set using the relevant function call - see the examples.

Usage

```
scheme(x)

scheme(x) <- value
```

Arguments

`x`	a URL, or vector of URLs
`value`	a replacement value (or vector of replacement values) for x's scheme.

See Also

`domain`, `port`, `path`, `parameters` and `fragment` for other accessors.

Examples

```
#Get a component
example_url <- "http://cran.r-project.org/submit.html"
scheme(example_url)

#Set a component
scheme(example_url) <- "https"

# NA out the URL
scheme(example_url) <- NA_character_
```

`strip_credentials` *Get or remove user authentication credentials*

Description

authentication credentials appear before the domain name and look like *user:password*. Sometimes you want the removed, or retrieved; `strip_credentials` and `get_credentials` do precisely that

Usage

```
strip_credentials(urls)

get_credentials(urls)
```

Arguments

`urls`	a URL, or vector of URLs

Examples

```
# Remove credentials
strip_credentials("http://foo:bar@97.77.104.22:3128")

# Get credentials
get_credentials("http://foo:bar@97.77.104.22:3128")
```

`suffix_dataset` *Dataset of public suffixes*

Description

This dataset contains a registry of public suffixes, as retrieved from and defined by the public suffix list. It is sorted by how many periods(".") appear in the suffix, to optimise it for `suffix_extract`. It is a data.frame with two columns, the first is the list of suffixes and the second is our best guess at the comment or owner associated with the particular suffix.

Usage

```
data(suffix_dataset)
```

Format

A data.frame of 8030 rows and 2 columns

Note

Last updated 2016-07-31.

See Also

`suffix_extract` for extracting suffixes from domain names, and `suffix_refresh` for getting a new, totally-up-to-date dataset version.

---

suffix_extract *extract the suffix from domain names*

Description

domain names have suffixes - common endings that people can or could register domains under. This includes things like ".org", but also things like ".edu.co". A simple Top Level Domain list, as a result, probably won't cut it.

`suffix_extract` takes the list of public suffixes, as maintained by Mozilla (see `suffix_dataset`) and a vector of domain names, and produces a data.frame containing the suffix that each domain uses, and the remaining fragment.

Usage

```
suffix_extract(domains, suffixes = NULL)
```

Arguments

`domains`	a vector of damains, from `domain` or `url_parse`. Alternately, full URLs can be provided and will then be run through `domain` internally.
`suffixes`	a dataset of suffixes. By default, this is NULL and the function relies on `suffix_dataset`. Optionally, if you want more updated suffix data, you can provide the result of `suffix_refresh` for this parameter.

Value

a data.frame of four columns, "host" "subdomain", "domain" & "suffix". "host" is what was passed in. "subdomain" is the subdomain of the suffix. "domain" contains the part of the domain name that came before the matched suffix. "suffix" is, well, the suffix.

See Also

suffix_dataset for the dataset of suffixes.

Examples

```
# Using url_parse
domain_name <- url_parse("http://en.wikipedia.org")$domain
suffix_extract(domain_name)

# Using domain()
domain_name <- domain("http://en.wikipedia.org")
suffix_extract(domain_name)

## Not run:
#Relying on a fresh version of the suffix dataset
suffix_extract(domain("http://en.wikipedia.org"), suffix_refresh())

## End(Not run)
```

---

suffix_refresh *Retrieve a public suffix dataset*

Description

urltools comes with an inbuilt dataset of public suffixes, suffix_dataset. This is used in suffix_extract to identify the top-level domain within a particular domain name.

While updates to the dataset will be included in each new package release, there's going to be a gap between changes to the suffixes list and changes to the package. Accordingly, the package also includes suffix_refresh, which generates and returns a *fresh* version of the dataset. This can then be passed through to suffix_extract.

Usage

```
suffix_refresh()
```

Value

a dataset equivalent in format to suffix_dataset.

See Also

suffix_extract to extract suffixes from domain names, or suffix_dataset for the inbuilt, default version of the data.

Examples

```
## Not run:
new_suffixes <- suffix_refresh()

## End(Not run)
```

`tld_dataset` *Dataset of top-level domains (TLDs)*

Description

This dataset contains a registry of top-level domains, as retrieved from and defined by the IANA.

Usage

```
data(tld_dataset)
```

Format

A vector of 1275 elements.

Note

Last updated 2016-07-20.

See Also

`tld_extract` for extracting TLDs from domain names, and `tld_refresh` to get an updated version of this dataset.

`tld_extract` *Extract TLDs*

Description

`tld_extract` extracts the top-level domain (TLD) from a vector of domain names. This is distinct from the suffixes, extracted with `suffix_extract`; TLDs are *top* level, while suffixes are just domains through which internet users can publicly register domains (the difference between `.org.uk` and `.uk`).

Usage

```
tld_extract(domains, tlds = NULL)
```

Arguments

domains	a vector of domains, retrieved through url_parse or domain.
tlds	a dataset of TLDs. If NULL (the default), tld_extract relies on urltools' tld_dataset; otherwise, you can pass in the result of tld_refresh.

Value

a data.frame of two columns: domain, with the original domain names, and tld, the identified TLD from the domain.

See Also

suffix_extract for retrieving suffixes (distinct from TLDs).

Examples

```
# Using the inbuilt dataset
domains <- domain("https://en.wikipedia.org/wiki/Main_Page")
tld_extract(domains)

# Using a refreshed one
tld_extract(domains, tld_refresh())
```

---

tld_refresh *Retrieve a TLD dataset*

Description

urltools comes with an inbuilt dataset of top level domains (TLDs), tld_dataset. This is used in tld_extract to identify the top-level domain within a particular domain name.

While updates to the dataset will be included in each new package release, there's going to be a gap between changes to TLDs and changes to the package. Accordingly, the package also includes tld_refresh, which generates and returns a *fresh* version of the dataset. This can then be passed through to tld_extract.

Usage

```
tld_refresh()
```

Value

a dataset equivalent in format to tld_dataset.

See Also

tld_extract to extract suffixes from domain names, or tld_dataset for the inbuilt, default version of the data.

Examples

```
## Not run:
new_tlds <- tld_refresh()

## End(Not run)
```

`urltools` *Tools for handling URLs*

Description

This package provides functions for URL encoding and decoding, parsing, and parameter extraction, designed to be both fast and entirely vectorised. It is intended to be useful for people dealing with web-related datasets, such as server-side logs.

See Also

the package vignette.

`url_compose` *Recompose Parsed URLs*

Description

Sometimes you want to take a vector of URLs, parse them, perform some operations and then rebuild them. `url_compose` takes a data.frame produced by `url_parse` and rebuilds it into a vector of full URLs (or: URLs as full as the vector initially thrown into url_parse).

This is currently a 'beta' feature; please do report bugs if you find them.

Usage

```
url_compose(parsed_urls)
```

Arguments

`parsed_urls`	a data.frame sourced from `url_parse`

See Also

`scheme` and other accessors, which you may want to run URLs through before composing them to modify individual values.

Examples

```
#Parse a URL and compose it
url <- "http://en.wikipedia.org"
url_compose(url_parse(url))
```

`url_decode` *Encode or decode a URI*

Description

encodes or decodes a URI/URL

Usage

```
url_decode(urls)

url_encode(urls)
```

Arguments

`urls`	a vector of URLs to decode or encode.

Details

URL encoding and decoding is an essential prerequisite to proper web interaction and data analysis around things like server-side logs. The relevant IETF RfC mandates the percentage-encoding of non-Latin characters, including things like slashes, unless those are reserved.

Base R provides `URLdecode` and `URLencode`, which handle URL encoding - in theory. In practise, they have a set of substantial problems that the urltools implementation solves::

- No vectorisation: Both base R functions operate on single URLs, not vectors of URLs. This means that, when confronted with a vector of URLs that need encoding or decoding, your only option is to loop from within R. This can be incredibly computationally costly with large datasets. url_encode and url_decode are implemented in C++ and entirely vectorised, allowing for a substantial performance improvement.
- No scheme recognition: encoding the slashes in, say, http://, is a good way of making sure your URL no longer works. Because of this, the only thing you can encode in URLencode (unless you refuse to encode reserved characters) is a partial URL, lacking the initial scheme, which requires additional operations to set up and increases the complexity of encoding or decoding. url_encode detects the protocol and silently splits it off, leaving it unencoded to ensure that the resulting URL is valid.
- ASCII NULs: Server side data can get very messy and sometimes include out-of-range characters. Unfortunately, URLdecode's response to these characters is to convert them to NULs, which R can't handle, at which point your URLdecode call breaks. `url_decode` simply ignores them.

Value

a character vector containing the encoded (or decoded) versions of "urls".

See Also

puny_decode and puny_encode, for punycode decoding and encoding.

Examples

```
url_decode("https://en.wikipedia.org/wiki/File:Vice_City_Public_Radio_%28logo%29.jpg")
url_encode("https://en.wikipedia.org/wiki/File:Vice_City_Public_Radio_(logo).jpg")

## Not run:
#A demonstrator of the contrasting behaviours around out-of-range characters
URLdecode("%gIL")
url_decode("%gIL")

## End(Not run)
```

---

url_parse *split URLs into their component parts*

Description

url_parse takes a vector of URLs and splits each one into its component parts, as recognised by RfC 3986.

Usage

```
url_parse(urls)
```

Arguments

urls	a vector of URLs

Details

It's useful to be able to take a URL and split it out into its component parts - for the purpose of hostname extraction, for example, or analysing API calls. This functionality is not provided in base R, although it is provided in parse_url; that implementation is entirely in R, uses regular expressions, and is not vectorised. It's perfectly suitable for the intended purpose (decomposition in the context of automated HTTP requests from R), but not for large-scale analysis.

Note that user authentication/identification information is not extracted; this can be found with get_credentials.

Value

a data.frame consisting of the columns scheme, domain, port, path, query and fragment. See the 'relevant IETF RfC for definitions. If an element cannot be identified, it is represented by an empty string.

See Also

`param_get` for extracting values associated with particular keys in a URL's query string, and `url_compose`, which is `url_parse` in reverse.

Examples

```
url_parse("https://en.wikipedia.org/wiki/Article")
```

Index