Package 'sched'

August 20, 2024

Title Request Scheduler

Version 1.0.2

Maintainer Pierrick Roger <pierrick.roger@cea.fr>

Description Offers classes and functions to contact web servers while enforcing scheduling rules required by the sites. The URL class makes it easy to construct a URL by providing parameters as a vector. The Request class allows to describes SOAP (Simple Object Access Protocol) or standard requests: URL, method (POST or GET), header, body. The Scheduler class controls the request frequency for each server address by mean of rules (Rule class). The RequestResult class permits to get the request status to handle error cases and the content.

URL https://gitlab.com/cnrgh/databases/r-sched

BugReports https://gitlab.com/cnrgh/databases/r-sched/-/issues

Depends R (>= 4.1)

License AGPL-3

Encoding UTF-8

Suggests roxygen2, testthat, knitr, rmarkdown, covr, lintr

Imports R6, fscache (>= 1.0.3), chk, lgr, methods, RCurl, tools, openssl

NeedsCompilation no

RoxygenNote 7.3.2

Collate 'Request.R' 'RequestResult.R' 'Rule.R' 'URL.R' 'request_fcts.R' 'Scheduler.R' 'package.R' 'rule_fcts.R'

VignetteBuilder knitr

Author Pierrick Roger [aut, cre] (<https://orcid.org/0000-0001-8177-4873>)

Repository CRAN

Date/Publication 2024-08-20 09:40:10 UTC

Contents

`sched-package` *sched: Request Scheduler*

Description

Offers classes and functions to contact web servers while enforcing scheduling rules required by the sites. The URL class makes it easy to construct a URL by providing parameters as a vector. The Request class allows to describes SOAP (Simple Object Access Protocol) or standard requests: URL, method (POST or GET), header, body. The Scheduler class controls the request frequency for each server address by mean of rules (Rule class). The RequestResult class permits to get the request status to handle error cases and the content.

Details

sched package.

sched offers classes and functions to contact web servers while enforcing scheduling rules required by the sites. The URL class makes it easy to construct a URL by providing parameters as a vector. The Request class allows to describes SOAP or standard requests: URL, method (POST or GET), header, body. The Scheduler class controls the request frequency for each server address by mean of rules (Rule class). The RequestResult class permits to get the request status to handle error cases and the content.

Author(s)

Maintainer: Pierrick Roger <pierrick.roger@cea.fr> (ORCID)

See Also

Rule, URL.

get_url_request_result *Send a request and get results.*

Description

Send the request described by a Request instance, using the provided user agent, and return the results.

Usage

```
get_url_request_result(
  request,
  useragent = NULL,
  ssl_verifypeer = TRUE,
  binary = FALSE
)
```

Arguments

request	A sched:Request object.
useragent	The user agent, as a character value. Example: "myapp ; my.name@my.addr"
ssl_verifypeer	Set to FALSE if you want to disable SSL verification for https sites. TRUE by default.
binary	Set to TRUE if the content to be retrieved is binary.

Value

The request result, as a character value.

Examples

```
# Retrieve the content of a web page
u <- sched::URL$new('https://httpbin.org/get')
content <- sched::get_url_request_result(sched::Request$new(u))
```

make_post_request *Make a POST request.*

Description

Construct a sched::Request object with a valid header for a POST request.

Usage

```
make_post_request(url, body, mime, soap_action = NULL, encoding = NULL)
```

Arguments

url	A sched::URL object.
body	The body of the POST request.
mime	The MIME type of the body. Example: "text/xml", "application/json".
soap_action	In case of a SOAP request, the SOAP action to contact, as a character string.
encoding	The encoding to use. A valid integer or string as required by RCurl.

Value

A sched::Request object.

Examples

```
# Prepare the URL and the request body
the_url <- sched::URL$new('https://httpbin.org/anything')
the_body <- '{"some_key": "my_value"}'

# Make the request object
my_request <- sched::make_post_request(the_url, body = the_body,
                                       mime = "application/json")
```

Request *Class Request.*

Description

Class Request.

Class Request.

Details

This class represents a Request object that can be used with the Request Scheduler.

Methods

Public methods:

- Request$new()
- Request$getUrl()
- Request$getMethod()
- Request$getEncoding()
- Request$getCurlOptions()
- Request$getUniqueKey()
- Request$getHeaderAsSingleString()
- Request$getBody()
- Request$print()
- Request$toString()
- Request$clone()

Method `new()`**:** Initializer.

Usage:

```
Request$new(
  url,
  method = c("get", "post"),
  header = NULL,
  body = NULL,
  encoding = NULL
)
```

Arguments:

`url` A `sched::URL` object.

`method` HTTP method. Either "get" or "post".

`header` The header of the POST method as a named character vector. The names are the fields of the header.

`body` The body as a character single value.

`encoding` The encoding to use. A valid integer or string as required by RCurl.

Returns: Nothing.

Examples:

```
# Create a GET request for the getCompleteEntity webservice of ChEBI
# database
request <- sched::Request$new(
  sched::URL$new(
    'https://www.ebi.ac.uk/webservices/chebi/2.0/test/getCompleteEntity',
    params=c(chebiId=15440)))

# Create a POST Request object for the records-batch-post webservice of
# ChemSpider database
request <- sched::Request$new(
  url=sched::URL$new(c('https://api.rsc.org/compounds/v1/', 'records',
                       'batch')),
```

```
  method='post', header=c('Content-Type'="", apikey='my-token'),
  body='{"recordIds": [2], "fields": ["SMILES","Formula","InChI"]}')
```

Method `getUrl()`**:** Gets the URL.

Usage:

```
Request$getUrl()
```

Returns: The URL of this Request object as a sched::URL object.

Examples:

```
# Create a GET request
request <- sched::Request$new(sched::URL$new('https://peakforest.org/'))

# Get the stored URL object
print(request$getUrl())
```

Method `getMethod()`**:** Gets the method.

Usage:

```
Request$getMethod()
```

Returns: The method as a character value.

Examples:

```
# Create a GET request
request <- sched::Request$new(sched::URL$new('https://peakforest.org/'))

# Get the stored method
print(request$getMethod())
```

Method `getEncoding()`**:** Gets the encoding.

Usage:

```
Request$getEncoding()
```

Returns: The encoding.

Examples:

```
# Create a GET request
request <- sched::Request$new(sched::URL$new('https://my.site.fr/'),
                              encoding='UTF-8')

# Get the stored encoding
print(request$getEncoding())
```

Method `getCurlOptions()`**:** Gets the options object to pass to cURL library.

Make a RCurl::CURLOptions object by calling RCurl::curlOptions() function. Useragent, header and body are passed as options if not NULL.

Usage:

```
Request$getCurlOptions(useragent = NULL)
```

Arguments:

useragent The user agent as a character value, or NULL.

Returns: An RCurl::CURLOptions object.

Examples:

```
# Create a POST Request object for the records-batch-post webservice of
# ChemSpider database
request <- sched::Request$new(
  url=sched::URL$new(c('https://api.rsc.org/compounds/v1/', 'records',
                       'batch')),
  method='post', header=c('Content-Type'="", apikey='my-token'),
  body='{"recordIds": [2], "fields": ["SMILES","Formula","InChI"]}')

# Get the associated RCurl options object
rcurl_opts <- request$getCurlOptions('myapp ; me@my.address')
```

Method `getUniqueKey()`**:** Gets a unique key to identify this request.
The key is an MD5 sum computed from the string representation of this request.

Usage:

```
Request$getUniqueKey()
```

Returns: A unique key as an MD5 sum.

Examples:

```
# Create a GET request
request <- sched::Request$new(sched::URL$new('https://peakforest.org/'))

# Get the MD5 sum of this request
print(request$getUniqueKey())
```

Method `getHeaderAsSingleString()`**:** Gets the HTTP header as a string, concatenating all its information into a single string.

Usage:

```
Request$getHeaderAsSingleString()
```

Returns: The header as a single character value.

Examples:

```
# Create a POST Request object for the records-batch-post webservice of
# ChemSpider database
request <- sched::Request$new(
  url=sched::URL$new(c('https://api.rsc.org/compounds/v1/', 'records',
                       'batch')),
  method='post', header=c('Content-Type'="", apikey='my-token'),
  body='{"recordIds": [2], "fields": ["SMILES","Formula","InChI"]}')

# Get back the POST header as a single string
print(request$getHeaderAsSingleString())
```

Method `getBody()`**:** Gets the body.

Usage:

```
Request$getBody()
```

Returns: The body as a single character value.

Examples:

```
# Create a POST Request object for the records-batch-post webservice of
# ChemSpider database
request <- sched::Request$new(
  url=sched::URL$new(c('https://api.rsc.org/compounds/v1/', 'records',
                       'batch')),
  method='post', header=c('Content-Type'="", apikey='my-token'),
  body='{"recordIds": [2], "fields": ["SMILES","Formula","InChI"]}')

# Get back the POST body
print(request$getBody())
```

Method `print()`**:** Displays information about this instance.

Usage:

```
Request$print()
```

Returns: self as invisible.

Examples:

```
# Create a GET request
request <- sched::Request$new(sched::URL$new('https://peakforest.org/'))

# Print the Request object
print(request)
```

Method `toString()`**:** Gets a string representation of this instance.

Usage:

```
Request$toString()
```

Returns: A single string giving a representation of this instance.

Examples:

```
# Create a GET request
request <- sched::Request$new(sched::URL$new('https://peakforest.org/'))

# Get the string representation of this request
print(request$toString())
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Request$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create a GET request for the getCompleteEntity webservice of ChEBI database
request <- sched::Request$new(
  sched::URL$new(
    'https://www.ebi.ac.uk/webservices/chebi/2.0/test/getCompleteEntity',
    params=c(chebiId=15440)))

# Get an MD5 key, unique to this request
key <- request$getUniqueKey()

# Print the request
print(request)


## ------------------------------------------------
## Method `Request$new`
## ------------------------------------------------

# Create a GET request for the getCompleteEntity webservice of ChEBI
# database
request <- sched::Request$new(
  sched::URL$new(
    'https://www.ebi.ac.uk/webservices/chebi/2.0/test/getCompleteEntity',
    params=c(chebiId=15440)))

# Create a POST Request object for the records-batch-post webservice of
# ChemSpider database
request <- sched::Request$new(
  url=sched::URL$new(c('https://api.rsc.org/compounds/v1/', 'records',
                       'batch')),
  method='post', header=c('Content-Type'="", apikey='my-token'),
  body='{"recordIds": [2], "fields": ["SMILES","Formula","InChI"]}')


## ------------------------------------------------
## Method `Request$getUrl`
## ------------------------------------------------

# Create a GET request
request <- sched::Request$new(sched::URL$new('https://peakforest.org/'))

# Get the stored URL object
print(request$getUrl())

## ------------------------------------------------
## Method `Request$getMethod`
## ------------------------------------------------

# Create a GET request
request <- sched::Request$new(sched::URL$new('https://peakforest.org/'))

# Get the stored method
```

```
print(request$getMethod())

## ------------------------------------------------
## Method `Request$getEncoding`
## ------------------------------------------------

# Create a GET request
request <- sched::Request$new(sched::URL$new('https://my.site.fr/'),
                              encoding='UTF-8')

# Get the stored encoding
print(request$getEncoding())

## ------------------------------------------------
## Method `Request$getCurlOptions`
## ------------------------------------------------

# Create a POST Request object for the records-batch-post webservice of
# ChemSpider database
request <- sched::Request$new(
  url=sched::URL$new(c('https://api.rsc.org/compounds/v1/', 'records',
                       'batch')),
  method='post', header=c('Content-Type'="", apikey='my-token'),
  body='{"recordIds": [2], "fields": ["SMILES","Formula","InChI"]}')

# Get the associated RCurl options object
rcurl_opts <- request$getCurlOptions('myapp ; me@my.address')


## ------------------------------------------------
## Method `Request$getUniqueKey`
## ------------------------------------------------

# Create a GET request
request <- sched::Request$new(sched::URL$new('https://peakforest.org/'))

# Get the MD5 sum of this request
print(request$getUniqueKey())


## ------------------------------------------------
## Method `Request$getHeaderAsSingleString`
## ------------------------------------------------

# Create a POST Request object for the records-batch-post webservice of
# ChemSpider database
request <- sched::Request$new(
  url=sched::URL$new(c('https://api.rsc.org/compounds/v1/', 'records',
                       'batch')),
  method='post', header=c('Content-Type'="", apikey='my-token'),
  body='{"recordIds": [2], "fields": ["SMILES","Formula","InChI"]}')

# Get back the POST header as a single string
```

```
print(request$getHeaderAsSingleString())


## ------------------------------------------------
## Method `Request$getBody`
## ------------------------------------------------

# Create a POST Request object for the records-batch-post webservice of
# ChemSpider database
request <- sched::Request$new(
  url=sched::URL$new(c('https://api.rsc.org/compounds/v1/', 'records',
                       'batch')),
  method='post', header=c('Content-Type'="", apikey='my-token'),
  body='{"recordIds": [2], "fields": ["SMILES","Formula","InChI"]}')

# Get back the POST body
print(request$getBody())


## ------------------------------------------------
## Method `Request$print`
## ------------------------------------------------

# Create a GET request
request <- sched::Request$new(sched::URL$new('https://peakforest.org/'))

# Print the Request object
print(request)


## ------------------------------------------------
## Method `Request$toString`
## ------------------------------------------------

# Create a GET request
request <- sched::Request$new(sched::URL$new('https://peakforest.org/'))

# Get the string representation of this request
print(request$toString())
```

`RequestResult` *Class RequestResult.*

Description

Class RequestResult.

Class RequestResult.

Details

Represents the result of a request.

Methods

Public methods:

- RequestResult$new()
- RequestResult$getContent()
- RequestResult$getRetry()
- RequestResult$getErrMsg()
- RequestResult$getStatus()
- RequestResult$getRetryAfter()
- RequestResult$getLocation()
- RequestResult$processRequestErrors()
- RequestResult$clone()

Method `new()`**:** New instance initializer.

Usage:

```
RequestResult$new(
  content = NULL,
  retry = FALSE,
  err_msg = NULL,
  status = 0,
  status_msg = "",
  retry_after = NULL,
  location = NULL
)
```

Arguments:

`content` The result content.

`retry` If request should be resent.

`err_msg` Error message.

`status` HTTP status.

`status_msg` Status message.

`retry_after` Time after which to retry.

`location` New location.

Returns: Nothing.

Method `getContent()`**:** Get content.

Usage:

```
RequestResult$getContent()
```

Returns: The content as a character value or NULL.

Method `getRetry()`**:** Get the retry flag.

Usage:

```
RequestResult$getRetry()
```

Returns: TRUE if the URL request should be sent again, FALSE otherwise.

Method `getErrMsg()`**:** Get the error message.

Usage:

```
RequestResult$getErrMsg()
```

Returns: The error message as a character value or NULL.

Method `getStatus()`**:** Get the HTTP status of the response.

Usage:

```
RequestResult$getStatus()
```

Returns: The status as an integer.

Method `getRetryAfter()`**:** Get the time to wait before retrying.

Usage:

```
RequestResult$getRetryAfter()
```

Returns: The time.

Method `getLocation()`**:** Get the redirect location.

Usage:

```
RequestResult$getLocation()
```

Returns: The redirect location as a character value or NULL.

Method `processRequestErrors()`**:** Process possible HTTP error.

Usage:

```
RequestResult$processRequestErrors()
```

Returns: Nothing.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RequestResult$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Rule *Scheduling rule class.*

Description

Scheduling rule class.

Scheduling rule class.

Details

This class represents a scheduling rule, used to limit the number of events during a certain lap of time.

Methods

Public methods:

- Rule$new()
- Rule$getN()
- Rule$getLapTime()
- Rule$print()
- Rule$wait()
- Rule$clone()

Method `new()`**:** Initializer.

Usage:

```
Rule$new(n = 3L, lap = 1)
```

Arguments:

`n` Number of events during a time lap.

`lap` Duration of a time lap, in seconds.

Returns: Nothing.

Examples:

```
# Create a rule object with default parameters
r <- Rule$new()

# Create a rule object with 5 events allowed each second (default time)
r2 <- Rule$new(5L)

# Create a rule object with 5 events allowed each 3 seconds
r3 <- Rule$new(5L, 3)
```

Method `getN()`**:** Gets the number of events allowed during a lap time.

Usage:

```
Rule$getN()
```

Returns: Returns the number of events as an integer.

Examples:

```
r <- Rule$new()

#' Get the allowed number of events for a rule
print(r$getN())
```

Method `getLapTime()`**:** Gets the lap time.

The number of seconds during which N events are allowed.

Usage:

```
Rule$getLapTime()
```

Returns: Returns Lap time as a numeric.

Examples:

```
# Create a rule object with default parameters
r <- Rule$new()

#' Get the configured lap time for a rule
print(r$getLapTime())
```

Method `print()`**:** Displays information about this instance.

Usage:

```
Rule$print()
```

Returns: Nothing.

Examples:

```
# Create a rule object with default parameters
r <- Rule$new()

# Print information about a rule object
print(r)
```

Method `wait()`**:** Wait (sleep) until a new event is allowed.

Usage:

```
Rule$wait(do_sleep = TRUE)
```

Arguments:

`do_sleep` Debug parameter that turns off the call to Sys.sleep(). Use only for testing.

Returns: The time passed to wait, in seconds.

Examples:

```
# Create a rule object that allows 3 events each 0.02 seconds
r <- Rule$new(3, 0.02)

#' Loop for generating 20 events
i <- 0 # event index
while (i < 20) {
```

```
  # Wait until next event is allowed
  wait_time <- r$wait()
  print(paste("We have waited", wait_time,
    "second(s) and are now allowed to process event number", i))
  i <- i + 1
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Rule$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# Create a new Rule object:
rule <- sched::Rule$new(n=1,lap=0.2) # 1 event allowed each 2 seconds

# Wait to be allowed to process with first event:
rule$wait() # The first event will be allowed directly, no waiting time.
# Process your first event here
rule$wait() # The second event will be delayed 0.2 seconds. This time
            # includes the time passed between the first call to wait() and
            # this one.
# Process your second event here


## ------------------------------------------------
## Method `Rule$new`
## ------------------------------------------------

# Create a rule object with default parameters
r <- Rule$new()

# Create a rule object with 5 events allowed each second (default time)
r2 <- Rule$new(5L)

# Create a rule object with 5 events allowed each 3 seconds
r3 <- Rule$new(5L, 3)

## ------------------------------------------------
## Method `Rule$getN`
## ------------------------------------------------

r <- Rule$new()

#' Get the allowed number of events for a rule
print(r$getN())

## ------------------------------------------------
## Method `Rule$getLapTime`
```

```
## ------------------------------------------------

# Create a rule object with default parameters
r <- Rule$new()

#' Get the configured lap time for a rule
print(r$getLapTime())

## ------------------------------------------------
## Method `Rule$print`
## ------------------------------------------------

# Create a rule object with default parameters
r <- Rule$new()

# Print information about a rule object
print(r)

## ------------------------------------------------
## Method `Rule$wait`
## ------------------------------------------------

# Create a rule object that allows 3 events each 0.02 seconds
r <- Rule$new(3, 0.02)

#' Loop for generating 20 events
i <- 0 # event index
while (i < 20) {
  # Wait until next event is allowed
  wait_time <- r$wait()
  print(paste("We have waited", wait_time,
    "second(s) and are now allowed to process event number", i))
  i <- i + 1
}
```

`Scheduler` *Class for scheduling web requests.*

Description

Class for scheduling web requests.

Class for scheduling web requests.

Details

The Scheduler class controls the frequency of access to web sites, through the definiton of access rules (`Rule` class). It handles GET and POST requests, as well as file downloading. It can use a cache system to store request results and avoid resending identical requests.

Methods

Public methods:

- Scheduler$new()
- Scheduler$setRule()
- Scheduler$sendRequest()
- Scheduler$downloadFile()
- Scheduler$getUrlString()
- Scheduler$getUrl()
- Scheduler$deleteRules()
- Scheduler$getNbRules()
- Scheduler$setOffline()
- Scheduler$isOffline()
- Scheduler$clone()

Method `new()`**:** New instance initializer.

There should be only one Scheduler instance in an application. There is no sense in having two or more instances, since they will ignore each other and break the access frequency rules when they contact the same sites.

Usage:

```
Scheduler$new(
  default_rule = Rule$new(),
  ssl_verifypeer = TRUE,
  nb_max_tries = 10L,
  cache_dir = tools::R_user_dir("sched", which = "cache"),
  user_agent = NULL,
  dwnld_timeout = 3600
)
```

Arguments:

`default_rule` The default_rule to use when none has been defined for a site.

`ssl_verifypeer` If set to TRUE (default), SSL certificate will be checked, otherwise certificates will be ignored.

`nb_max_tries` Maximum number of tries when running a request.

`cache_dir` Set the path to the file system cache. Set to NULL to disable the cache system. The cache system will save downloaded content and reuse it later for identical requests.

`user_agent` The application name and contact address to send to the contacted web server.

`dwnld_timeout` The timeout used by `downloadFile()` method, in seconds.

Returns: Nothing.

Examples:

```
# Create a scheduler instance with a custom default_rule
scheduler <- sched::Scheduler$new(default_rule=sched::Rule$new(10, 1),
                                  cache_dir = NULL)
```

Method `setRule()`**:** Defines a rule for a site.

Defines a rule for a site. The site is identified by its hostname. Each time a request will be made to this host (i.e.: the URL contains the defined hostname), the scheduling rule will be applied in order to wait (sleep) if nedeed before sending the request.

If a rule already exists for this hostname, it will be replaced.

Usage:

```
Scheduler$setRule(host, n = 3L, lap = 1)
```

Arguments:

`host` The hostname of the site.

`n` Number of events during a time lap.

`lap` Duration of a time lap, in seconds.

Returns: Nothing.

Examples:

```
# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Define a rule with default values
scheduler$setRule('www.ebi.ac.uk')

# Define a rule with custome values
scheduler$setRule('my.other.site', n=10, lap=3)
```

Method `sendRequest()`**:** Sends a request, and retrieves content result.

Usage:

```
Scheduler$sendRequest(request, cache_read = TRUE)
```

Arguments:

`request` A `sched::Request` instance.

`cache_read` If set to TRUE and the cache system is enabled, the cache system will be searched for the request and the cached result returned. In any case, if the the cache system is enabled, and the request sent, the retrieved content will be stored into the cache.

Returns: The results returned by the contacted server, as a single string value.

Examples:

```
# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Define a scheduling rule of 7 requests every 2 seconds
scheduler$setRule('www.ebi.ac.uk', n=7, lap=2)

# Create a request object
u <- 'https://www.ebi.ac.uk/webservices/chebi/2.0/test/getCompleteEntity'
url <- sched::URL$new(url=u, params=c(chebiId=15440))
request <- sched::Request$new(url)
```

```
# Send the request and get the content result
content <- scheduler$sendRequest(request)
```

Method `downloadFile()`**:** Downloads the content of a URL and save it into the specified destination file.
This method works for any URL, even if it has been written with heavy files in mind. Since it uses `utils::download.file()` which saves the content directly on disk, the cache system is not used.

Usage:

```
Scheduler$downloadFile(url, dest_file, quiet = FALSE, timeout = NULL)
```

Arguments:

`url` The URL to access, as a sched::URL object.

`dest_file` A path to a destination file.

`quiet` The quiet parameter for `utils::download.file()`.

`timeout` The timeout in seconds. Defaults to value provided in initializer.

Returns: Nothing.

Examples:

```
# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Create a temporary directory
tmp_dir <- tempdir()

# Download a file
u <- sched::URL$new(
    'https://gitlab.com/cnrgh/databases/r-sched/-/raw/main/README.md',
    c(ref_type='heads'))
scheduler$downloadFile(u, file.path(tmp_dir, 'README.md'))

# Remove the temporary directory
unlink(tmp_dir, recursive = TRUE)
```

Method `getUrlString()`**:** Builds a URL string, using a base URL and parameters to be passed.
The provided base URL and parameters are combined into a full URL string.
DEPRECATED. Use the `sched::URL` class and its method `toString()` instead.

Usage:

```
Scheduler$getUrlString(url, params = list())
```

Arguments:

`url` A URL string.

`params` A list of URL parameters.

Returns: The full URL string as a single character value.

Examples:

```
# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Create a URL string
url.str <- scheduler$getUrlString(
  'https://www.ebi.ac.uk/webservices/chebi/2.0/test/getCompleteEntity',
  params=c(chebiId=15440))
```

Method `getUrl()`**:** Sends a request and get the result.
DEPRECATED. Use method `sendRequest()` instead.

Usage:

```
Scheduler$getUrl(
  url,
  params = list(),
  method = c("get", "post"),
  header = NULL,
  body = NULL,
  encoding = NULL
)
```

Arguments:

`url` A URL string.

`params` A list of URL parameters.

`method` The method to use. Either 'get' or 'post'.

`header` The header to send.

`body` The body to send.

`encoding` The encoding to use.

Returns: The results of the request.

Examples:

```
# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Send request
content <- scheduler$getUrl(
  'https://www.ebi.ac.uk/webservices/chebi/2.0/test/getCompleteEntity',
  params=c(chebiId=15440))
```

Method `deleteRules()`**:** Removes all defined rules, including the ones automatically defined using default_rule.

Usage:

```
Scheduler$deleteRules()
```

Returns: Nothing.

Examples:

```
# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Define a rule with custome values
scheduler$setRule('my.other.site', n=10, lap=3)

# Delete all defined rules
scheduler$deleteRules()
```

Method `getNbRules()`**:** Gets the number of defined rules, including the ones automatically defined using default_rule.

Usage:

```
Scheduler$getNbRules()
```

Returns: The number of rules defined.

Examples:

```
# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Get the number of defined rules
print(scheduler$getNbRules())
```

Method `setOffline()`**:** Enables or disables offline mode.
If the offline mode is enabled, an error will be raised when the class attemps to send a request. This mode is mainly useful when debugging the usage of the cache system.

Usage:

```
Scheduler$setOffline(offline)
```

Arguments:

`offline` Set to TRUE to enable offline mode, and FALSE otherwise.

Returns: Nothing.

Examples:

```
# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Enable offline mode
scheduler$setOffline(TRUE)
```

Method `isOffline()`**:** Tests if offline mode is enabled.

Usage:

```
Scheduler$isOffline()
```

Returns: TRUE is offline mode is enabled, FALSE otherwise.

Examples:

```
# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Test if offline mode is enabled
if (scheduler$isOffline())
  print("Scheduler is offline.")
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Scheduler$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# Create a scheduler instance without cache
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Define a rule with default values
scheduler$setRule('www.ebi.ac.uk')

# Create a request object
u <- 'https://www.ebi.ac.uk/webservices/chebi/2.0/test/getCompleteEntity'
url <- sched::URL$new(url=u, params=c(chebiId=15440))
request <- sched::Request$new(url)

# Send the request and get the content result
content <- scheduler$sendRequest(request)


## ------------------------------------------------
## Method `Scheduler$new`
## ------------------------------------------------

# Create a scheduler instance with a custom default_rule
scheduler <- sched::Scheduler$new(default_rule=sched::Rule$new(10, 1),
                                  cache_dir = NULL)


## ------------------------------------------------
## Method `Scheduler$setRule`
## ------------------------------------------------

# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Define a rule with default values
scheduler$setRule('www.ebi.ac.uk')

# Define a rule with custome values
```

```
scheduler$setRule('my.other.site', n=10, lap=3)


## ------------------------------------------------
## Method `Scheduler$sendRequest`
## ------------------------------------------------

# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Define a scheduling rule of 7 requests every 2 seconds
scheduler$setRule('www.ebi.ac.uk', n=7, lap=2)

# Create a request object
u <- 'https://www.ebi.ac.uk/webservices/chebi/2.0/test/getCompleteEntity'
url <- sched::URL$new(url=u, params=c(chebiId=15440))
request <- sched::Request$new(url)

# Send the request and get the content result
content <- scheduler$sendRequest(request)


## ------------------------------------------------
## Method `Scheduler$downloadFile`
## ------------------------------------------------

# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Create a temporary directory
tmp_dir <- tempdir()

# Download a file
u <- sched::URL$new(
    'https://gitlab.com/cnrgh/databases/r-sched/-/raw/main/README.md',
    c(ref_type='heads'))
scheduler$downloadFile(u, file.path(tmp_dir, 'README.md'))

# Remove the temporary directory
unlink(tmp_dir, recursive = TRUE)


## ------------------------------------------------
## Method `Scheduler$getUrlString`
## ------------------------------------------------

# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Create a URL string
url.str <- scheduler$getUrlString(
  'https://www.ebi.ac.uk/webservices/chebi/2.0/test/getCompleteEntity',
  params=c(chebiId=15440))
```

```
## ------------------------------------------------
## Method `Scheduler$getUrl`
## ------------------------------------------------

# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Send request
content <- scheduler$getUrl(
  'https://www.ebi.ac.uk/webservices/chebi/2.0/test/getCompleteEntity',
  params=c(chebiId=15440))


## ------------------------------------------------
## Method `Scheduler$deleteRules`
## ------------------------------------------------

# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Define a rule with custome values
scheduler$setRule('my.other.site', n=10, lap=3)

# Delete all defined rules
scheduler$deleteRules()


## ------------------------------------------------
## Method `Scheduler$getNbRules`
## ------------------------------------------------

# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Get the number of defined rules
print(scheduler$getNbRules())


## ------------------------------------------------
## Method `Scheduler$setOffline`
## ------------------------------------------------

# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Enable offline mode
scheduler$setOffline(TRUE)


## ------------------------------------------------
## Method `Scheduler$isOffline`
```

```
## ------------------------------------------------

# Create a scheduler instance
scheduler <- sched::Scheduler$new(cache_dir = NULL)

# Test if offline mode is enabled
if (scheduler$isOffline())
  print("Scheduler is offline.")
```

URL *URL class.*

Description

URL class.

URL class.

Details

This class represents a URL object that can be used in requests. It handles parameters as a list, making it easy to build URLs for contacting web services.

Methods

Public methods:

- `URL$new()`
- `URL$getDomain()`
- `URL$setUrl()`
- `URL$setParam()`
- `URL$print()`
- `URL$toString()`
- `URL$clone()`

Method `new()`**:** Initializer.

Usage:

```
URL$new(url = character(), params = character(), chomp_extra_slashes = TRUE)
```

Arguments:

`url` The URL to access, as a character vector.

`params` The list of parameters to append to this URL. If it is an unnamed list or vector, the values will be converted to strings and concatenated with the `&` separator. If it is a named list or vector, the names will be used as keys as in "name1=value1&name2=value2&...".

`chomp_extra_slashes` If set to TRUE, then slashes at the end and the beginning of each element of the url vector parameter will be removed before proper concatenation.

Returns: Nothing.

Examples:

```
# Create a URL object
url <- sched::URL$new("https://www.my.server/", c(param1=12,
                      param2='abc'))
```

Method `getDomain()`**:** Etracts the domain name from the URL.

Usage:

```
URL$getDomain()
```

Returns: The domain.

Examples:

```
# Create a URL object
url <- sched::URL$new("https://www.my.server/",
                      c(param1=12, param2='abc'))

# Extract the domain name
print(url$getDomain())
```

Method `setUrl()`**:** Sets the base URL string.

Usage:

```
URL$setUrl(url)
```

Arguments:

`url` The base URL string.

Returns: Nothing.

Examples:

```
# Create an empty URL object
url <- sched::URL$new()

# Set the URL
url$setUrl('https://www.my.server/')

# Convert the URL to a string
print(url$toString())
```

Method `setParam()`**:** Sets a parameter.

Usage:

```
URL$setParam(key, value)
```

Arguments:

`key` The parameter name.

`value` The value of the parameter.

Returns: Nothing.

Examples:

```
# Create an URL object
url <- sched::URL$new('https://www.my.server/')

# Set a parameter
url$setParam('a', 12)

# Convert the URL to a string
print(url$toString())
```

Method `print()`**:** Displays information about this instance.

Usage:

```
URL$print()
```

Returns: self as invisible.

Examples:

```
# Create an URL object
url <- sched::URL$new('https://www.my.server/')

# Print the URL object
print(url)
```

Method `toString()`**:** Gets the URL as a string representation.

Usage:

```
URL$toString(encode = TRUE)
```

Arguments:

`encode` If set to TRUE, then encodes the URL.

Returns: The URL as a string, with all parameters and values set.

Examples:

```
# Create an URL object
url <- sched::URL$new('https://www.my.server/', c(a=12))

# Convert the URL to a string
print(url$toString())
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
URL$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create a URL object from a base URL string and a list of parameters
base.url <- c("https://www.uniprot.org", "uniprot")
params <- c(query="reviewed:yes+AND+organism:9606",
       columns='id,entry name,protein names',
```

```
       format="tab")
url <- sched::URL$new(url=base.url, params=params)

# Print the URL converted to a string
print(url$toString())


## ------------------------------------------------
## Method `URL$new`
## ------------------------------------------------

# Create a URL object
url <- sched::URL$new("https://www.my.server/", c(param1=12,
                      param2='abc'))


## ------------------------------------------------
## Method `URL$getDomain`
## ------------------------------------------------

# Create a URL object
url <- sched::URL$new("https://www.my.server/",
                      c(param1=12, param2='abc'))

# Extract the domain name
print(url$getDomain())

## ------------------------------------------------
## Method `URL$setUrl`
## ------------------------------------------------

# Create an empty URL object
url <- sched::URL$new()

# Set the URL
url$setUrl('https://www.my.server/')

# Convert the URL to a string
print(url$toString())

## ------------------------------------------------
## Method `URL$setParam`
## ------------------------------------------------

# Create an URL object
url <- sched::URL$new('https://www.my.server/')

# Set a parameter
url$setParam('a', 12)

# Convert the URL to a string
print(url$toString())
```

```
## ------------------------------------------------
## Method `URL$print`
## ------------------------------------------------

# Create an URL object
url <- sched::URL$new('https://www.my.server/')

# Print the URL object
print(url)

## ------------------------------------------------
## Method `URL$toString`
## ------------------------------------------------

# Create an URL object
url <- sched::URL$new('https://www.my.server/', c(a=12))

# Convert the URL to a string
print(url$toString())
```

Index