Package ‘webmockr’

March 24, 2020

Title Stubbing and Setting Expectations on 'HTTP' Requests

Description Stubbing and setting expectations on 'HTTP' requests. Includes tools for stubbing 'HTTP' requests, including expected request conditions and response conditions. Match on 'HTTP' method, query parameters, request body, headers and more. Can be used for unit tests or outside of a testing context.

Version 0.6.2

License MIT + file LICENSE


BugReports https://github.com/ropensci/webmockr/issues

LazyData true

Encoding UTF-8

Language en-US

Imports curl, jsonlite, magrittr (>= 1.5), R6 (>= 2.1.3), urtools (>= 1.6.0), fauxpas, curl (>= 0.7.0)

Suggests testthat, xml2, vcr, htr

RoxygenNote 7.1.0

X-schema.org-applicationCategory Web

X-schema.org-keywords http, https, API, web-services, curl, mock, mocking, fakeweb, http-mocking, testing, testing-tools, tdd

X-schema.org-isPartOf https://ropensci.org

NeedsCompilation no

Author Scott Chamberlain [aut, cre] (<https://orcid.org/0000-0003-1444-9135>), Aaron Wolen [ctb] (<https://orcid.org/0000-0003-2542-2202>), rOpenSci [fnd] (https://ropensci.org)

Maintainer Scott Chamberlain <myrmecocystus+r@gmail.com>

Repository CRAN

Date/Publication 2020-03-24 21:00:02 UTC
## R topics documented:

- `webmockr-package` .................................................. 2
- `build_curl_request` ................................................. 3
- `build_curl_response` .............................................. 4
- `build_httr_request` ............................................... 4
- `build_httr_response` ............................................. 5
- `CrulAdapter` ....................................................... 5
- `enable` ............................................................. 8
- `HashCounter` ....................................................... 9
- `HttpLibAdapaterRegistry` ......................................... 10
- `httr_mock` .......................................................... 11
- `mocking-disk-writing` ............................................ 12
- `mock_file` .......................................................... 13
- `pluck_body` ........................................................ 14
- `remove_request_stub` ............................................ 14
- `RequestPattern` ................................................... 15
- `RequestRegistry` .................................................. 17
- `RequestSignature` ................................................ 19
- `request_registry` .................................................. 21
- `Response` .......................................................... 22
- `StubbedRequest` ................................................... 26
- `StubRegistry` ..................................................... 29
- `stub_registry` ..................................................... 32
- `stub_registry_clear` .............................................. 33
- `stub_request` ...................................................... 33
- `to_raise` ........................................................... 36
- `to_return` .......................................................... 36
- `to_timeout` ........................................................ 38
- `webmockr-defunct` ................................................ 38
- `webmockr_configure` ............................................ 39
- `wi_th` ............................................................... 40

### Index

<table>
<thead>
<tr>
<th>webmockr-package</th>
<th>webmockr</th>
</tr>
</thead>
<tbody>
<tr>
<td>webmockr-package</td>
<td>2</td>
</tr>
</tbody>
</table>

### Description

Stubbing and setting expectations on HTTP requests
Features

- Stubbing HTTP requests at low http client lib level
- Setting and verifying expectations on HTTP requests
- Matching requests based on method, URI, headers and body
- Supports multiple HTTP libraries, including **crul** and **httr**
- Integration with HTTP test caching library **vcr**

Author(s)

Scott Chamberlain <myrmecocystus+r@gmail.com>

Aaron Wolen

Examples

```r
library(webmockr)
stub_request("get", "https://httpbin.org/get")
stub_request("post", "https://httpbin.org/post")
stub_registry()
```

Description

Build a crul request

Usage

```r
build_crul_request(x)
```

Arguments

- `x` an unexecuted crul request object

Value

a crul request
build_crul_response    Build a crul response

Description
Build a crul response

Usage
build_crul_response(req, resp)

Arguments
req    a request
resp   a response

Value
da crul response

build_httr_request    Build a httr request

Description
Build a httr request

Usage
build_httr_request(x)

Arguments
x   an unexecuted httr request object

Value
da httr request
build_httr_response

---

**build_httr_response**  Build a http response

---

**Description**

Build a http response

**Usage**

```r
def build_httr_response(req, resp)
```

**Arguments**

- `req`: a request
- `resp`: a response

**Value**

a http response

---

**CrulAdapter**  Adapters for Modifying HTTP Requests

---

**Description**

Adapter is the base parent class used to implement `webmockr` support for different HTTP clients. It should not be used directly. Instead, use one of the client-specific adapters that `webmockr` currently provides:

- `CrulAdapter` for **cru**
- `HttrAdapter` for **httr**

**Details**

Note that the documented fields and methods are the same across all client-specific adapters.

**Super class**

```
webmockr::Adapter -> CrulAdapter
```

**Public fields**

- `client`  HTTP client package name
- `name`  adapter name
Methods

Public methods:

• **CrulAdapter$clone()**

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

*Usage:*

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

*Arguments:*

deep: Whether to make a deep clone.

Super class

```
webmockr::Adapter -> HttrAdapter
```

Public fields

- client: HTTP client package name
- name: adapter name

Methods

Public methods:

• **HttrAdapter$clone()**

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

*Usage:*

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

*Arguments:*

deep: Whether to make a deep clone.

Public fields

- client: HTTP client package name
- name: adapter name

Methods

Public methods:

• **Adapter$new()**
• **Adapter$enable()**
• **Adapter$disable()**
• **Adapter$handle_request()**
• **Adapter$remove_stubs()**
• **Adapter$clone()**
Method new(): Create a new Adapter object

Usage:
Adapter$new()

Method enable(): Enable the adapter

Usage:
Adapter$enable()

Returns: TRUE, invisibly

Method disable(): Disable the adapter

Usage:
Adapter$disable()

Returns: FALSE, invisibly

Method handle_request(): All logic for handling a request

Usage:
Adapter$handle_request(req)

Arguments:
req a request

Returns: various outcomes

Method remove_stubs(): Remove all stubs

Usage:
Adapter$remove_stubs()

Returns: nothing returned; removes all request stubs

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

Usage:
Adapter$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

Examples

## Not run:
if (requireNamespace("httr", quietly = TRUE)) {
  # library(httr)

  # normal httr request, works fine
  # real <- GET("https://httpbin.org/get")
  # real

  # with webmockr
  # library(webmockr)
  ## turn on httr mocking
enable

Enable or disable webmockr

Description

Enable or disable webmockr

Usage

enable(adapter = NULL, options = list())

disabled(adapter = "crul")

disable(adapter = NULL, options = list())

Arguments

adapter (character) the adapter name, 'crul' or 'httr'. one or the other. if none given, we attempt to enable both adapters

options list of options - ignored for now.
HashCounter

Details
 enable() enables webmockr for all adapters. disable() disables webmockr for all adapters. enabled() answers whether webmockr is enabled for a given adapter.

Value
 enable() and disable() invisibly returns booleans for each adapter, as a result of running enable or disable, respectively, on each HttpLibAdapterRegistry object. enabled returns a single boolean.

Description
 hash with counter, to store requests, and count each time it is used.

Public fields
 hash (list) a list for internal use only.

Methods

Public methods:
- HashCounter$put()
- HashCounter$get()
- HashCounter$clone()

Method put(): Register a request by it's key
Usage:
HashCounter$put(key)
Arguments:
key a character string of the request, serialized from CrulAdapter or another adapter
Returns: nothing returned; registers request and iterates internal counter

Method get(): Get a request by key
Usage:
HashCounter$get(key)
Arguments:
key a character string of the request, serialized from CrulAdapter or another adapter
Returns: (character) an http request as a string

Method clone(): The objects of this class are cloneable with this method.
Usage:
HashCounter$clone(deep = FALSE)
Arguments:
deep Whether to make a deep clone.
HttpLibAdapterRegistry

See Also

Other request-registry: RequestRegistry, request_registry()

Examples

```r
x <- HashCounter$new()
x$put("foo bar")
x$put("foo bar")
x$put("hello world")
x$put("hello world")
x$put("hello world")
x$hash
```

HttpLibAdapterRegistry

Description

http lib adapter registry

Public fields

- adapters list

Methods

Public methods:

- `HttpLibAdapterRegistry$print()
- `HttpLibAdapterRegistry$register()
- `HttpLibAdapterRegistry$clone()

Method `print()`: print method for the HttpLibAdapterRegistry class

Usage:

`HttpLibAdapterRegistry$print(x, ...)

Arguments:

- `x` self

... ignored

Method `register()`: Register an http library adapter

Usage:

`HttpLibAdapterRegistry$register(x)

Arguments:

- `x` an http lib adapter, e.g., CrulAdapter
 Returns: nothing, registers the library adapter

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

Usage:
HttpLibAdapterRegistry$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

Examples

```r
x <- HttpLibAdapterRegistry$new()
x$register(CrulAdapter$new())
x
x$adapters
x$adapters[[1]]$name
```

---

**httr_mock**  

*Turn on httr mocking* Sets a callback that routes httr request through webmockr

---

**Description**

Turn on httr mocking Sets a callback that routes httr request through webmockr

**Usage**

`httr_mock(on = TRUE)`

**Arguments**

- `on` (logical) set to TRUE to turn on, and FALSE to turn off. default: TRUE

**Value**

Silently returns TRUE when enabled and FALSE when disabled.
mocking-disk-writing  

Mocking writing to disk

Description

Mocking writing to disk

Examples

```r
## Not run:
# enable mocking
enable()

# Write to a file before mocked request

# crul
library(crul)
## make a temp file
f <- tempfile(fileext = "json")
## write something to the file
cat("\"hello\":\"world\")\n", file = f)
readLines(f)
## make the stub
stub_request("get", "https://httpbin.org/get") %>%
to_return(body = file(f))
## make a request
(out <- HttpClient$new("https://httpbin.org/get")$get(disk = f))
out$content
readLines(out$content)

# httr
library(httr)
## make a temp file
f <- tempfile(fileext = "json")
## write something to the file
cat("\"hello\":\"world\")\n", file = f)
readLines(f)
## make the stub
stub_request("get", "https://httpbin.org/get") %>%
to_return(body = file(f),
  headers = list('content-type' = "application/json"))
## make a request
## with httr, you must set overwrite=TRUE or you'll get an error
out <- GET("https://httpbin.org/get", write_disk(f, overwrite=TRUE))
out
out$content
content(out, "text", encoding = "UTF-8")

# Use mock_file to have webmockr handle file and contents
```
library(crul)

f <- tempfile(fileext = ".json")

## make the stub
stub_request("get", "https://httpbin.org/get") %>%
  to_return(body = mock_file(f, "{"hello": "mars"\}n"))

## make a request
(out <- crul::HttpClient$new("https://httpbin.org/get")$get(disk = f))
out$content
readLines(out$content)

## httr
library(httr)

f <- tempfile(fileext = ".json")

## make the stub
stub_request("get", "https://httpbin.org/get") %>%
  to_return(
    body = mock_file(path = f, payload = "{"foo": "bar"}\")
  ,
    headers = list("content-type" = "application/json")
  )

## make a request
out <- GET("https://httpbin.org/get", write_disk(f))
out

## view stubbed file content
out$content
readLines(out$content)
content(out, "text", encoding = "UTF-8")

## disable mocking
disable()

## End(Not run)

---

**mock_file**

Mock file

### Description
Mock file

### Usage
mock_file(path, payload)

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>(character) a file path. required</td>
</tr>
<tr>
<td>payload</td>
<td>(character) string to be written to the file given at path parameter. required</td>
</tr>
</tbody>
</table>
Value

a list with S3 class mock_file

Examples

```r
mock_file(path = tempfile(), payload = "{"foo": "bar"}"")
```

---

pluck_body

*Extract the body from an HTTP request*

Description

Returns an appropriate representation of the data contained within a request body based on its encoding.

Usage

```r
pluck_body(x)
```

Arguments

- `x` an unexecuted curl or httr request object

Value

one of the following:

- NULL if the request is not associated with a body
- NULL if an upload is used not in a list
- list containing the multipart-encoded body
- character vector with the JSON- or raw-encoded body, or upload form file

---

remove_request_stub

*Remove a request stub*

Description

Remove a request stub

Usage

```r
remove_request_stub(stub)
```

Arguments

- `stub` a request stub, of class StubbedRequest
Value

logical, TRUE if removed, FALSE if not removed

See Also

Other stub-registry: StubRegistry, stub_registry_clear(), stub_registry()

Examples

(x <- stub_request("get", "https://httpbin.org/get"))
stub_registry()
remove_request_stub(x)
stub_registry()

<table>
<thead>
<tr>
<th>RequestPattern</th>
<th>RequestPattern class</th>
</tr>
</thead>
</table>

Description

class handling all request matchers

Public fields

method_pattern xxx
uri_pattern xxx
body_pattern xxx
headers_pattern xxx

Methods

Public methods:

• RequestPattern$new()
• RequestPattern$matches()
• RequestPattern$to_s()
• RequestPattern$clone()

Method new(): Create a new RequestPattern object

Usage:
RequestPattern$new(  
  method,
  uri = NULL,
  uri_regex = NULL,
  query = NULL,
  body = NULL,
  headers = NULL
)
Arguments:
method the HTTP method (any, head, options, get, post, put, patch, trace, or delete). "any" matches any HTTP method. required.
uri (character) request URI. required or uri_regex
uri_regex (character) request URI as regex. required or uri
query (list) query parameters, optional
body (list) body request, optional
headers (list) headers, optional

Returns: A new RequestPattern object

Method matches(): does a request signature match the selected matchers?
Usage:
RequestPattern$matches(request_signature)
Arguments:
request_signature a RequestSignature object
Returns: a boolean

Method to_s(): Print pattern for easy human consumption
Usage:
RequestPattern$to_s()
Returns: a string

Method clone(): The objects of this class are cloneable with this method.
Usage:
RequestPattern$clone(deep = FALSE)
Arguments:
deep Whether to make a deep clone.

See Also
pattern classes for HTTP method MethodPattern, headers HeadersPattern, body BodyPattern, and URI/URL UriPattern

Examples
## Not run:
(x <- RequestPattern$new(method = "get", uri = "https://httpbin.org/get"))
x$body_pattern
x$headers_pattern
x$method_pattern
x(uri_pattern
x$to_s()

# make a request signature
rs <- RequestSignature$new(method = "get", uri = "https://httpbin.org/get")
# check if it matches
x$matches(rs)

# regex uri
(x <- RequestPattern$new(method = "get", uri_regex = ".+ossref.org"))
x$uri_pattern
x$uri_pattern$to_s()
x$to_s()

# uri with query parameters
(x <- RequestPattern$new(
  method = "get", uri = "https://httpbin.org/get",
  query = list(foo = "bar")
))
x$to_s()

# just headers (via setting method=any & uri_regex=.+)
headers <- list(
  'User-Agent' = 'Apple',
  'Accept-Encoding' = 'gzip, deflate',
  'Accept' = 'application/json, text/xml, application/xml, */*')
x <- RequestPattern$new(
  method = "any",
  uri_regex = ".+",
  headers = headers)
x$to_s()
rs <- RequestSignature$new(method = "any", uri = "http://foo.bar",
  options = list(headers = headers))
rs
x$matches(rs)

# body
x <- RequestPattern$new(method = "post", uri = "https://httpbin.org/post",
  body = list(y = crul::upload(system.file("CITATION"))))
x$to_s()
rs <- RequestSignature$new(method = "post", uri = "https://httpbin.org/post",
  options = list(
    body = list(y = crul::upload(system.file("CITATION")))))
rs
x$matches(rs)

## End(Not run)

---

**Description**

keeps track of HTTP requests
Public fields

request_signatures  a HashCounter object

Methods

Public methods:

- RequestRegistry$print()
- RequestRegistry$reset()
- RequestRegistry$register_request()
- RequestRegistry$clone()

Method print(): print method for the RequestRegistry class

Usage:

RequestRegistry$print(x, ...)

Arguments:

- x  self
- ...  ignored

Method reset(): Reset the registry to no registered requests

Usage:

RequestRegistry$reset()

Returns: nothing returned; resets registry to no requests

Method register_request(): Register a request

Usage:

RequestRegistry$register_request(request)

Arguments:

- request  a character string of the request, serialized from CrulAdapter or another adapter

Returns: nothing returned; registers the request

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

Usage:

RequestRegistry$clone(deep = FALSE)

Arguments:

- deep  Whether to make a deep clone

See Also

stub_registry() and StubRegistry

Other request-registry: HashCounter, request_registry()
**Examples**

```r
x <- RequestRegistry$new()
x$register_request(request = "GET http://scottchamberlain.info")
x$register_request(request = "GET http://scottchamberlain.info")
x$register_request(request = "POST https://httpbin.org/post")
# print method to list requests
x

# hashes, and number of times each requested
x$request_signatures$hash

# reset the request registry
x$reset()
```

---

**Description**

General purpose request signature builder

**Public fields**

- `method` (character): an http method
- `uri` (character): a uri
- `body` (various): request body
- `headers` (list): named list of headers
- `proxies` (list): proxies as a named list
- `auth` (list): authentication details, as a named list
- `url` (internal use)
- `disk` (character): if writing to disk, the path

**Methods**

**Public methods:**

- `RequestSignature$new()`
- `RequestSignature$print()`
- `RequestSignature$to_s()`
- `RequestSignature$clone()`

**Method** `new()`: Create a new `RequestSignature` object

**Usage:**

```r
RequestSignature$new(method, uri, options = list())
```

**Arguments:**
method the HTTP method (any, head, options, get, post, put, patch, trace, or delete). "any" matches any HTTP method. required.
uri (character) request URI. required.
options (list) options. optional. See Details.

Returns: A new RequestSignature object

Method print(): print method for the RequestSignature class

Usage:
RequestSignature$print()

Arguments:
x self
... ignored

Method to_s(): Request signature to a string

Usage:
RequestSignature$to_s()

Returns: a character string representation of the request signature

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

Usage:
RequestSignature$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

Examples

# make request signature
x <- RequestSignature$new(method = "get", uri = "https://httpbin.org/get")
# method
x$method
# uri
x$uri
# request signature to string
x$to_s()

# headers
w <- RequestSignature$new(
  method = "get",
  uri = "https://httpbin.org/get",
  options = list(headers = list(\ User-Agent\ = "foobar", stuff = "things")))
)
w
w$headers
w$to_s()

# headers and body
bb <- RequestSignature$new(
method = "get",
uri = "https://httpbin.org/get",
options = list(
    headers = list('User-Agent' = "foobar", stuff = "things"),
    body = list(a = "tables")
)
bb
bb$headers
bb$body
bb$to_s()

# with disk path
f <- tempfile()
bb <- RequestSignature$new(
    method = "get",
    uri = "https://httpbin.org/get",
    options = list(disk = f)
)
bb
bb$disk
bb$to_s()

---

**request_registry**  
*List or clear requests in the request registry*

**Description**

List or clear requests in the request registry

**Usage**

```r
request_registry()
request_registry_clear()
```

**Details**

`request_registry()` lists the requests that have been made that webmockr knows about; `request_registry_clear()` resets the request registry (removes all recorded requests)

**Value**

an object of class `RequestRegistry`, print method gives the requests in the registry and the number of times each one has been performed

**See Also**

Other request-registry: HashCounter, RequestRegistry
**Examples**

```r
webmockr::enable()
stub_request("get", "https://httpbin.org/get") %>%
to_return(body = "success!", status = 200)

# nothing in the request registry
request_registry()

# make the request
z <- crul::HttpClient$new(url = "https://httpbin.org")$get("get")

# check the request registry - the request was made 1 time
request_registry()

# do the request again
z <- crul::HttpClient$new(url = "https://httpbin.org")$get("get")

# check the request registry - now it's been made 2 times, yay!
request_registry()

# clear the request registry
request_registry_clear()

webmockr::disable()
```

---

**Response**

<table>
<thead>
<tr>
<th>Description</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>custom webmockr http response class</td>
<td></td>
</tr>
</tbody>
</table>

**Public fields**

- url (character) a url
- body (various) list, character, etc
- content (various) response content/body
- request_headers (list) a named list
- response_headers (list) a named list
- options (character) list
- status_code (integer) an http status code
- exception (character) an exception message
- should_timeout (logical) should the response timeout?
Response

Methods

Public methods:
- Response\$new()
- Response\$print()
- Response\$set_url()
- Response\$get_url()
- Response\$set_request_headers()
- Response\$get_request_headers()
- Response\$set_response_headers()
- Response\$get_response_headers()
- Response\$set_body()
- Response\$get_body()
- Response\$set_status()
- Response\$get_status()
- Response\$set_exception()
- Response\$get_exception()
- Response\$clone()

Method new(): Create a new Response object
Usage:
Response\$new(options = list())
Arguments:
options (list) a list of options
Returns: A new Response object

Method print(): print method for the Response class
Usage:
Response\$print(x, ...)
Arguments:
x self
... ignored

Method set_url(): set the url for the response
Usage:
Response\$set_url(url)
Arguments:
url (character) a url
Returns: nothing returned; sets url

Method get_url(): get the url for the response
Usage:
Response\$get_url()
Method `set_request_headers()`: set the request headers for the response

Usage:
Response$set_request_headers(headers, capitalize = TRUE)

Arguments:
headers (list) named list
capitalize (logical) whether to capitalize first letters of each header; default: TRUE

Returns: nothing returned; sets request headers on the response

Method `get_request_headers()`: get the request headers for the response

Usage:
Response$get_request_headers()

Returns: (list) request headers, a named list

Method `set_response_headers()`: set the response headers for the response

Usage:
Response$set_response_headers(headers, capitalize = TRUE)

Arguments:
headers (list) named list
capitalize (logical) whether to capitalize first letters of each header; default: TRUE

Returns: nothing returned; sets response headers on the response

Method `get_response_headers()`: get the response headers for the response

Usage:
Response$get_response_headers()

Returns: (list) response headers, a named list

Method `set_body()`: set the body of the response

Usage:
Response$set_body(body, disk = FALSE)

Arguments:
body (various types)
disk (logical) whether its on disk; default: FALSE

Returns: nothing returned; sets body on the response

Method `get_body()`: get the body of the response

Usage:
Response$get_body()

Returns: various

Method `set_status()`: set the http status of the response
**Response**

*Usage:*  
Response$set_status(status)

*Arguments:*  
status (integer) the http status

*Returns:*  
nothing returned; sets the http status of the response

**Method** get_status(): get the http status of the response  
*Usage:*  
Response$get_status()

*Returns:*  
(integer) the http status

**Method** set_exception(): set an exception  
*Usage:*  
Response$set_exception(exception)

*Arguments:*  
exception (character) an exception string

*Returns:*  
nothing returned; sets an exception

**Method** get_exception(): get the exception, if set  
*Usage:*  
Response$get_exception()

*Returns:*  
(character) an exception

**Method** clone(): The objects of this class are cloneable with this method.  
*Usage:*  
Response$clone(deep = FALSE)

*Arguments:*  
deep Whether to make a deep clone.

**Examples**  
```r
## Not run:
(x <- Response$new())
x$set_url("https://httpbin.org/get")
x

x$set_request_headers(list('Content-Type' = "application/json"))
x
x$request_headers

x$set_response_headers(list('Host' = "httpbin.org"))
x
x$response_headers
```
```python
x$set_status(404)

x$set_body("hello world")

x$set_exception("exception")

## End(Not run)
```

### Description

stubbed request class underlying `stub_request()`

### Public fields

- `method` (xx) xx
- `uri` (xx) xx
- `uri_regex` (xx) xx
- `uri_parts` (xx) xx
- `host` (xx) xx
- `query` (xx) xx
- `body` (xx) xx
- `request_headers` (xx) xx
- `response_headers` (xx) xx
- `responses_sequences` (xx) xx
- `status_code` (xx) xx
- `timeout` (xx) xx
- `exceptions` (xx) xx
- `raise` (xx) xx
Methods

Public methods:

•StubbedRequest$new()
•StubbedRequest$print()
•StubbedRequest$with()
•StubbedRequest$to_return()
•StubbedRequest$to_timeout()
•StubbedRequest$to_raise()
•StubbedRequest$to_s()
•StubbedRequest$clone()

Method new(): Create a new StubbedRequest object

Usage:
StubbedRequest$new(method, uri = NULL, uri_regex = NULL)

Arguments:
method the HTTP method (any, head, get, post, put, patch, or delete). "any" matches any HTTP
method. required.
uri (character) request URI. either this or uri_regex required
uri_regex (character) request URI as regex. either this or uri required

Returns: A new StubbedRequest object

Method print(): print method for the StubbedRequest class

Usage:
StubbedRequest$print(x, ...)

Arguments:
x self
... ignored

Method with(): Set expectations for what’s given in HTTP request

Usage:
StubbedRequest$with(query = NULL, body = NULL, headers = NULL)

Arguments:
query (list) request query params, as a named list. optional
body (list) request body, as a named list. optional
headers (list) request headers as a named list. optional.

Returns: nothing returned; sets only

Method to_return(): Set expectations for what’s returned in HTTP response

Usage:
StubbedRequest$to_return(status, body, headers)

Arguments:
status (numeric) an HTTP status code
body (list) response body, one of: character, json, list, raw, numeric, NULL, FALSE, or a file connection (other connection types not supported)
headers (list) named list, response headers. optional.

Returns: nothing returned; sets what to be returned

Method to_timeout(): Response should time out
Usage:
StubbedRequest$to_timeout()

Returns: nothing returned

Method to_raise(): Response should raise an exception x
Usage:
StubbedRequest$to_raise(x)
Arguments:
x (character) an exception message

Returns: nothing returned

Method to_s(): Response as a character string
Usage:
StubbedRequest$to_s()

Returns: (character) the response as a string

Method clone(): The objects of this class are cloneable with this method.
Usage:
StubbedRequest$clonedeep = FALSE)
Arguments:
deep Whether to make a deep clone.

See Also
stub_request()

Examples

## Not run:
x <- StubbedRequest$new(method = "get", uri = "api.crossref.org")
x$method
x$uri
x$with(headers = list('User-Agent' = 'R', apple = "good"))
x$to_return(status = 200, body = "fooban", headers = list(a = 5))
x
x$to_s()

# raw body
x <- StubbedRequest$new(method = "get", uri = "api.crossref.org")
x$to_return(status = 200, body = raw(0), headers = list(a = 5))
x$to_s()

# file path
x <- StubbedRequest$new(method = "get", uri = "api.crossref.org")
f <- tempfile()
x$to_return(status = 200, body = file(f), headers = list(a = 5))
x
x$to_s()
unlink(f)

# to_file(): file path and payload to go into the file
# payload written to file during mocked response creation
x <- StubbedRequest$new(method = "get", uri = "api.crossref.org")
f <- tempfile()
x$to_return(status = 200, body = mock_file(f, "{"foo": "bar"}"),
   headers = list(a = 5))
x
x$to_s()
unlink(f)

# uri_regex
(x <- StubbedRequest$new(method = "get", uri_regex = ".+ossref.org"))
x$method
x$uri
x$to_s()

# to timeout
(x <- StubbedRequest$new(method = "get", uri_regex = ".+ossref.org"))
x$to_s()
x$to_timeout()
x$to_s()

# to raise
library(fauxpas)
(x <- StubbedRequest$new(method = "get", uri_regex = ".+ossref.org"))
x$to_s()
x$to_raise(HTTPBadGateway)
x$to_s()

## End(Not run)

---

**StubRegistry**

**StubRegistry**

### Description

stub registry to keep track of **StubbedRequest** stubs
Public fields

request_stubs (list) list of request stubs
global_stubs (list) list of global stubs

Methods

Public methods:

• StubRegistry$print()
• StubRegistry$register_stub()
• StubRegistry$find_stubbed_request()
• StubRegistry$request_stub_for()
• StubRegistry$remove_request_stub()
• StubRegistry$remove_all_request_stubs()
• StubRegistry$is_registered()
• StubRegistry$clone()

Method print(): print method for the StubRegistry class

Usage:
StubRegistry$print(x, ...)

Arguments:

x  self
...  ignored

Method register_stub(): Register a stub

Usage:
StubRegistry$register_stub(stub)

Arguments:

stub  an object of type StubbedRequest

Returns:  nothing returned; registers the stub

Method find_stubbed_request(): Find a stubbed request

Usage:
StubRegistry$find_stubbed_request(req)

Arguments:

req  an object of class RequestSignature

Returns:  an object of type StubbedRequest, if matched

Method request_stub_for(): Find a stubbed request

Usage:
StubRegistry$request_stub_for(request_signature)

Arguments:

request_signature  an object of class RequestSignature
Returns: logical, 1 or more

Method remove_request_stub(): Remove a stubbed request by matching request signature

Usage:
StubRegistry$remove_request_stub(stub)

Arguments:
stub  an object of type StubbedRequest

Returns: nothing returned; removes the stub from the registry

Method remove_all_request_stubs(): Remove all request stubs

Usage:
StubRegistry$remove_all_request_stubs()

Returns: nothing returned; removes all request stubs

Method is_registered(): Find a stubbed request

Usage:
StubRegistry$is_registered(x)

Arguments:
x  an object of class RequestSignature

Returns: nothing returned; registers the stub

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

Usage:
StubRegistry$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

See Also

Other stub-registry: remove_request_stub(), stub_registry_clear(), stub_registry()

Examples

## Not run:
# Make a stub
stub1 <- StubbedRequest$new(method = "get", uri = "api.crossref.org")
stub1$with(headers = list('User-Agent' = 'R'))
stub1$to_return(status = 200, body = "foobar", headers = list())
stub1

# Make another stub
stub2 <- StubbedRequest$new(method = "get", uri = "api.crossref.org")
stub2

# Put both stubs in the stub registry
reg <- StubRegistry$new()
stub_registry

---

### Description

List stubs in the stub registry.

### Usage

`stub_registry()`

### Value

An object of class `StubRegistry`. Print method gives the stubs in the registry.

### See Also

Other stub-registry: `StubRegistry`, `remove_request_stub()`, `stub_registry_clear()`

### Examples

```r
# make a stub
stub_request("get", "https://httpbin.org/get") %>%
  to_return(body = "success!", status = 200)

# check the stub registry, there should be one in there
stub_registry()

# make another stub
stub_request("get", "https://httpbin.org/get") %>%
  to_return(body = "woopsy", status = 404)

# check the stub registry, now there are two there
stub_registry()

# to clear the stub registry
stub_registry_clear()
```
### stub_registry_clear

**Clear the stub registry**

**Description**
Clear all stubs

**Usage**
```
stub_registry_clear()
```

**Value**
nothing, well technically an empty list invisibly, but it’s not anything useful

**See Also**
Other stub-registry: `StubRegistry`, `remove_request_stub()`, `stub_registry()`

**Examples**
```
(x <- stub_request("get", "https://httpbin.org/get"))
stub_registry()
stub_registry_clear()
stub_registry()
```

### stub_request

**Stub an http request**

**Description**
Stub an http request

**Usage**
```
stub_request(method = "get", uri = NULL, uri_regex = NULL)
```

**Arguments**

- **method** (character) HTTP method, one of "get", "post", "put", "patch", "head", "delete", "options" - or the special "any" (for any method)
- **uri** (character) The request uri. Can be a full uri, partial, or a regular expression to match many incantations of a uri. required.
- **uri_regex** (character) A URI represented as regex. See examples
Details

Internally, this calls `StubbedRequest` which handles the logic.

See `stub_registry()` for listing stubs, `stub_registry_clear()` for removing all stubs and `remove_request_stub()` for removing specific stubs.

If multiple stubs match the same request, we use the first stub. So if you want to use a stub that was created after an earlier one that matches, remove the earlier one(s).

Value

an object of class `StubbedRequest`, with print method describing the stub.

Mocking writing to disk

See `mocking-disk-writing`

See Also

`wi_th()`, `to_return()`, `to_timeout()`, `to_raise()`, `mock_file()`

Examples

```r
## Not run:
# basic stubbing
stub_request("get", "https://httpbin.org/get")
stub_request("post", "https://httpbin.org/post")

# any method, use "any"
stub_request("any", "https://httpbin.org/get")

# list stubs
stub_registry()

# request headers
stub_request("get", "https://httpbin.org/get") %>%
  wi_th(headers = list("User-Agent" = "R"))

# request body
stub_request("post", "https://httpbin.org/post") %>%
  wi_th(body = list(foo = "bar"))
stub_registry()
library(crlul)
x <- crul::HttpClient$new(url = "https://httpbin.org")
crlul::mock()
x$post('post', body = list(foo = 'bar'))

# add expectation with to_return
stub_request("get", "https://httpbin.org/get") %>%
  wi_th(
    query = list(hello = "world"),
    headers = list("User-Agent" = "R")) %>%
  to_return(status = 200, body = "stuff", headers = list(a = 5))
```
# list stubs again
stub_registry()

# regex
stub_request("get", uri_regex = ".+ample\..")

# set stub an expectation to timeout
stub_request("get", "https://httpbin.org/get") %>% to_timeout()
x <- crul::HttpClient$new(url = "https://httpbin.org")
res <- x$get('get')

# raise exception
library(fauxpas)
stub_request("get", "https://httpbin.org/get") %>% to_raise(HTTPAccepted)
stub_request("get", "https://httpbin.org/get") %>% to_raise(HTTPAccepted, HTTPGone)
x <- crul::HttpClient$new(url = "https://httpbin.org")
stub_request("get", "https://httpbin.org/get") %>% to_raise(HTTPBadGateway)
crul::mock()
x$get('get')

# pass a list to .list
z <- stub_request("get", "https://httpbin.org/get")
wi_th(z, .list = list(query = list(foo = "bar")))

# just body
stub_request("any", uri_regex = ".+") %>%
  wi_th(body = list(foo = 'bar'))
library(crul)
x <- crul::HttpClient$new(url = "https://httpbin.org")
crul::mock()
x$post('post', body = list(foo = 'bar'))
x$put('put', body = list(foo = 'bar'))

# just headers
headers <- list(
  'Accept-Encoding' = 'gzip, deflate',
  'Accept' = 'application/json, text/xml, application/xml, */*')
stub_request("any", uri_regex = ".+") %>%
  wi_th(headers = headers)
library(crul)
x <- crul::HttpClient$new(url = "https://httpbin.org", headers = headers)
crul::mock()
x$post('post')
x$put('put', body = list(foo = 'bar'))
x$get('put', query = list(stuff = 3423234L))

# clear all stubs
stub_registry()
stub_registry_clear()

## End(Not run)
**to_raise**  
*Set raise error condition*

**Description**
Set raise error condition

**Usage**

to_raise(.data, ...)

**Arguments**

<table>
<thead>
<tr>
<th>.data</th>
<th>input. Anything that can be coerced to a StubbedRequest class object</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>One or more HTTP exceptions from the fauxpas package. Run grep(&quot;HTTP&quot;, getNamespaceExports(&quot;fauxpas&quot;), value = TRUE) for a list of possible exceptions</td>
</tr>
</tbody>
</table>

**Details**
The behavior in the future will be:
When multiple exceptions are passed, the first is used on the first mock, the second on the second mock, and so on. Subsequent mocks use the last exception
But for now, only the first exception is used until we get that fixed

**Value**
an object of class StubbedRequest, with print method describing the stub

**Note**
see examples in stub_request()

---

**to_return**  
*Expectation for what’s returned from a stubbed request*

**Description**
Set response status code, response body, and/or response headers

**Usage**

to_return(.data, ..., .list = list())
Arguments

.data input. Anything that can be coerced to a StubbedRequest class object

... Comma separated list of named variables. accepts the following: status, body, headers. See Details for more.

.list named list, has to be one of 'status', 'body', and/or 'headers'. An alternative to passing in via .... Don’t pass the same thing to both, e.g. don’t pass ’status’ to ...., and also ’status’ to this parameter

Details

Values for status, body, and headers:

- status: (numeric/integer) three digit status code
- body: various: character, json, list, raw, numeric, NULL, FALSE, a file connection (other connection types not supported), or a mock_file function call (see mock_file())
- headers: (list) a named list, must be named

response headers are returned with all lowercase names and the values are all of type character. if numeric/integer values are given (e.g., to_return(headers = list(a = 10))), we’ll coerce any numeric/integer values to character.

Value

an object of class StubbedRequest, with print method describing the stub

Note

see more examples in stub_request()

Examples

# first, make a stub object
(req <- stub_request("post", "https://httpbin.org/post"))

# add status, body and/or headers
to_return(req, status = 200)
to_return(req, body = "stuff")
to_return(req, body = list(a = list(b = "world")))
to_return(req, headers = list(a = 5))
to_return(req, status = 200, body = "stuff", headers = list(a = 5))

# .list - pass in a named list instead
to_return(req, .list = list(body = list(foo = "bar")))
### to_timeout

**Set timeout as an expected return on a match**

**Description**

Set timeout as an expected return on a match

**Usage**

```r
to_timeout(.data)
```

**Arguments**

- `.data` input. Anything that can be coerced to a `StubbedRequest` class object

**Value**

an object of class `StubbedRequest`, with print method describing the stub

**Note**

see examples in `stub_request()`

---

**webmockr-defunct**

*Defunct functions in webmockr*

**Description**

- `webmockr_enable()`: Function removed, see `enable()`
- `webmockr_disable()`: Function removed, see `disable()`
- `to_return_`: Only `to_return()` is available now
- `wi_th_`: Only `wi_th()` is available now
webmockr_configure

---

Description

webmockr configuration

Usage

webmockr_configure(
  allow_net_connect = FALSE,
  allow_localhost = FALSE,
  allow = NULL,
  net_http_connect_on_start = FALSE,
  show_stubbing_instructions = FALSE,
  query_values_notation = FALSE,
  show_body_diff = FALSE
)

webmockr_configure_reset()

webmockr_configuration()

webmockr_allow_net_connect()

webmockr_disable_net_connect(allow = NULL)

webmockr_net_connect_allowed(uri = NULL)

Arguments

allow_net_connect
  (logical) Default: FALSE

allow_localhost
  (logical) Default: FALSE

allow
  (character) one or more URI/URL to allow (and by extension all others are not allowed)

net_http_connect_on_start
  (logical) Default: FALSE. ignored for now

show_stubbing_instructions
  (logical) Default: FALSE. ignored for now

query_values_notation
  (logical) Default: FALSE. ignored for now

show_body_diff
  (logical) Default: FALSE. ignored for now

uri
  (character) a URI/URL as a character string - to determine whether or not it is allowed
If there are stubs found for a request, even if net connections are allowed (by running `webmockr_allow_net_connect()`), the stubbed response will be returned. If no stub is found, and net connections are allowed, then a real HTTP request can be made.

### Examples

```r
## Not run:
webmockr_configure()
webmockr_configure(
  allow_localhost = TRUE
)
webmockr_configuration()
webmockr_configure_reset()

webmockr_allow_net_connect()
webmockr_net_connect_allowed()

# disable net connect for any URIs
webmockr_disable_net_connect()
### gives NULL with no URI passed
webmockr_net_connect_allowed()
# disable net connect EXCEPT FOR given URIs
webmockr_disable_net_connect(allow = "google.com")
### is a specific URI allowed?
webmockr_net_connect_allowed("google.com")

## End(Not run)
```

### wi_th

Set additional parts of a stubbed request

**Description**

Set query params, request body, and/or request headers

**Usage**

```r
wi_th(.data, ..., .list = list())
```

**Arguments**

- **.data**: input. Anything that can be coerced to a StubbedRequest class object
- **...**: Comma separated list of named variables. accepts the following: query, body, headers.
- **.list**: named list, has to be one of 'query', 'body', and/or 'headers'. An alternative to passing in via ... . Don’t pass the same thing to both, e.g. don’t pass 'query' to ..., and also 'query' to this parameter
Details

with is a function in the base package, so we went with wi_th

Values for query, body, and headers:

- query: (list) a named list
- body: various, including character string, list, raw, numeric, upload (crul::upload or httr::upload_file, they both create the same object in the end)
- headers: (list) a named list

Value

an object of class StubbedRequest, with print method describing the stub

Note

see more examples in stub_request()

Examples

```r
# first, make a stub object
req <- stub_request("post", "https://httpbin.org/post")

# add body
# list
wi_th(req, body = list(foo = "bar"))
# string
wi_th(req, body = '{"foo": "bar"}')
# raw
wi_th(req, body = charToRaw('{"foo": "bar"}'))
# numeric
wi_th(req, body = 5)
# an upload
wi_th(req, body = crul::upload(system.file("CITATION")))
# wi_th(req, body = httr::upload_file(system.file("CITATION")))

# add query - has to be a named list
wi_th(req, query = list(foo = "bar"))

# add headers - has to be a named list
wi_th(req, headers = list(foo = "bar"))
wi_th(req, headers = list('User-Agent' = "webmockr/v1", hello="world"))

# .list - pass in a named list instead
wi_th(req, .list = list(body = list(foo = "bar")))
```
Index

*Topic package
  *webmockr-package, 2
  Adapter (CrulAdapter), 5
  BodyPattern, 16
  build_crl_request, 3
  build_crl_response, 4
  build_httr_request, 4
  build_httr_response, 5
  CrulAdapter, 5, 9, 10, 18
  disable (enable), 8
disable(), 38
  enable, 8
enable(), 38
  enabled (enable), 8
  HashCounter, 9, 18, 21
  HeadersPattern, 16
  HttpLibAdapterRegistry, 9, 10
  httl_mock, 11
  HttrAdapter (CrulAdapter), 5
  MethodPattern, 16
  mock_file, 13
  mock_file(), 34, 37
  mocking-disk-writing, 12, 34
  pluck_body, 14
  remove_request_stub, 14, 31–33
  remove_request_stub(), 34
  request_registry, 10, 18, 21
  request_registry_clear
    (request_registry), 21
  RequestPattern, 15
  RequestRegistry, 10, 17, 21
  RequestSignature, 16, 19, 30, 31

  Response, 22
  stub_registry, 15, 31, 32, 33
  stub_registry(), 18, 34
  stub_registry_clear, 15, 31, 32, 33
  stub_registry_clear(), 34
  stub_request, 33
  stub_request(), 26, 28, 36–38, 41
  StubbedRequest, 26, 29–31, 34
  StubRegistry, 15, 18, 29, 32, 33
to_raise, 36
to_raise(), 34
to_return, 36
to_return(), 34, 38
to_return_(), 38
to_timeout, 38
to_timeout(), 34

  UriPattern, 16
  webmockr (webmockr-package), 2
  webmockr-defunct, 38
  webmockr-package, 2
  webmockr::Adapter, 5, 6
  webmockr_allow_net_connect
    (webmockr_configuration), 39
  webmockr_configuration
    (webmockr_configuration), 39
  webmockr_configuration, 39
  webmockr_configuration_reset
    (webmockr_configuration), 39
  webmockr_disable(), 38
  webmockr_disable_net_connect
    (webmockr_configuration), 39
  webmockr_enable(), 38
  webmockr_net_connect_allowed
    (webmockr_configuration), 39
  wi_th, 40
  wi_th(), 34, 38
  wi_th_, 38