Package ‘webmockr’

July 23, 2024

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  1.0.0

License  MIT + file LICENSE

URL  https://github.com/ropensci/webmockr,
     https://books.ropensci.org/http-testing/,
     https://docs.ropensci.org/webmockr/

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

Encoding  UTF-8

Language  en-US

Imports  curl, jsonlite, magrittr (>= 1.5), R6 (>= 2.1.3), urtools (>= 1.6.0), fauxpas, crul (>= 0.7.0), base64enc

Suggests  testthat, xml2, vcr, htttr, htttr2

RoxygenNote  7.3.2

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  2024-07-23 05:50:02 UTC
Contents

build_crul_request .............................................. 2
build_crul_response ............................................ 3
build_httr2_request .......................................... 3
build_httr2_response ......................................... 4
build_httr_request ........................................... 5
build_httr_response ........................................... 5
CrulAdapter ......................................................... 6
enable ............................................................. 9
HashCounter ......................................................... 10
HttpLibAdapterRegistry .......................................... 11
httr2_mock ........................................................ 12
httr_mock .......................................................... 13
mocking-disk-writing ............................................. 13
mock_file .......................................................... 16
pluck_body ........................................................ 16
remove_request_stub .............................................. 17
RequestPattern ....................................................... 17
RequestRegistry ..................................................... 20
RequestSignature ................................................... 22
request_registry ................................................... 24
Response ............................................................... 25
StubbedRequest ...................................................... 29
StubCounter ......................................................... 33
StubRegistry ........................................................ 34
stub_registry ....................................................... 37
stub_registry_clear ............................................... 38
stub_request ........................................................ 38
to_raise .......................................................... 41
to_return .......................................................... 42
to_timeout ........................................................ 44
webmockr-defunct .................................................. 45
webmockr_configure ............................................... 45
webmockr_reset .................................................... 46
wi_th ............................................................. 47

Index 49

build_crul_request  Build a crul request

Description

Build a crul request
**build_crul_response**

**Usage**

```
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**

- a crul response

---

**build_httr2_request**  
*Build an htr2 request*

**Description**

Build an htr2 request

**Usage**

```
build_httr2_request(x)
```

**Arguments**

- **x** an unexecuted htr2 request object
Value

a httr2_request

Description

Build a httr2 response (httr2_response)

Usage

build_httr2_response(req, resp)

Arguments

req a request

resp a response

Value

an httr2 response (httr2_response)

Examples

```r
## Not run:
# x <- Httr2Adapter$new()
# library(httr2)
# req <- request("https://r-project.org")
# req = req %>% req_body_json(list(x = 1, y = 2))
# #req$method <- ‘POST’
# stub_request("post", "https://r-project.org") %>%
# # to_return(status = 418, body = list(a = 5))
# stub = webmockr_stub_registry$request_stubs[[1]]
# stub$counter$.__enclos_env__$private$total <- 1
# resp = x$.__enclos_env__$private$build_stub_response(stub)
# resp = x$.__enclos_env__$private$build_response(req, resp)
# resp = x$.__enclos_env__$private$add_response_sequences(stub, resp)
# out
# out$body
# out$content

## End(Not run)
```
**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**  
a httr request

---

**build_httr_response**  
*Build a httr response*

---

**Description**  
Build a httr response

**Usage**  
`build_httr_response(req, resp)`

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

**Value**  
a httr response
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**
- Httr2Adapter for **httr2**

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.

Super class

```
webmockr::Adapter -> Httr2Adapter
```

Public fields

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

Methods

Public methods:

- `Httr2Adapter$clone()`

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

Usage:

```
Httr2Adapter$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(quiet = FALSE)

Arguments:
quiet (logical) suppress messages? default: FALSE

Returns: TRUE, invisibly

Method `disable()`: Disable the adapter

Usage:
Adapter$disable(quiet = FALSE)

Arguments:
quiet (logical) suppress messages? default: FALSE

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$clonedeep = FALSE)

Arguments:
deep Whether to make a deep clone.
enable

Enable or disable webmockr

Usage

```r
enable(adapter = NULL, options = list(), quiet = FALSE)
```
### enabled

```r
enabled(adapter = "crul")
```

### disable

```r
disable(adapter = NULL, options = list(), quiet = FALSE)
```

#### Arguments

- **adapter** *(character)* the adapter name, 'crul', 'httr', or 'httr2'. one or the other. if none given, we attempt to enable both adapters
- **options** list of options - ignored for now.
- **quiet** (logical) suppress messages? default: FALSE

#### 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

---

### HashCounter

**Description**

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

**Public fields**

- **hash** *(list)* a list for internal use only, with elements key, sig, and count

**Methods**

**Public methods:**

- `HashCounter$put`
- `HashCounter$get`
- `HashCounter$clone`

**Method** put(): Register a request by it's key

*Usage:*

```r
HashCounter$put(req_sig)
```

*Arguments:*

- `req_sig` an object of class `RequestSignature`
HttpLibAdapaterRegistry

Returns: nothing returned; registers request and iterates internal counter

Method get(): Get a request by key

Usage:
HashCounter$get(req_sig)

Arguments:
req_sig  an object of class RequestSignature

Returns: (integer) the count of how many times the request has been made

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.

See Also

Other request-registry: RequestRegistry, request_registry()

Examples

x <- HashCounter$new()
x$hash
z <- RequestSignature$new(method = "get", uri = "https://httpbin.org/get")
x$put(z)
x$hash
x$get(z)
x$put(z)
x$get(z)

HttpLibAdapaterRegistry

HttpLibAdapaterRegistry

Description

http lib adapter registry

Public fields

adapters list
Methods

Public methods:

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

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

Usage:

```
HttpLibAdapaterRegistry$print(x, ...)  
```

Arguments:

- `x` self
- `...` ignored

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

Usage:

```
HttpLibAdapaterRegistry$register(x)  
```

Arguments:

- `x` an http lib adapter, e.g., `Cru1Adapter`

Returns: nothing, registers the library adapter

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

Usage:

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

Arguments:

- `deep` Whether to make a deep clone.

Examples

```
x <- HttpLibAdapaterRegistry$new()
x$register(Cru1Adapter$new())
x
x$adapters
x$adapters[[1]]$name
```

---

**httr2_mock**

Turn on httr2 mocking

---

**Description**

Sets a callback that routes httr2 requests through webmockr

**Usage**

```
httr2_mock(on = TRUE)  
```
**httr_mock**

**Arguments**

on (logical) TRUE to turn on, 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()
# getOption('httr2_mock')

# 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")

# httr2

library(httr2)

## 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

req <- request("https://httpbin.org/get")
out <- req_perform(req, path = f)
out
out$body
out
out$headers
readLines(out$body)

# Use mock_file to have webmockr handle file and contents

# crul
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)

## make a temp file
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")

# httr2
library(httr2)

## make a temp file
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
req <- request("https://httpbin.org/get")
out <- req_perform(req, path = f)
out

## view stubbed file content
out$body
readLines(out$body)

# disable mocking
disable()

## End(Not run)
mock_file

Mock file

Description

Mock file

Usage

mock_file(path, payload)

Arguments

path (character) a file path. required
payload (character) string to be written to the file given at path parameter. required

Value

a list with S3 class mock_file

Examples

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

pluck_body(x)

Arguments

x an unexecuted curl, httr or httr2 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**

remove_request_stub(stub)

**Arguments**

stub a request stub, of class StubbredRequest

**Value**

logical, TRUE if removed, FALSE if not removed

**See Also**

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

**Examples**

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

**RequestPattern**

RequestPattern class

**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

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

# make a request signature
rs <- RequestSignature$new(method = "get", uri = "http://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()

## query params included in url, not separately
(x <- RequestPattern$new(
    method = "get", uri = "https://httpbin.org/get?stuff=things"
))
x$to_s()
x$query_params

# 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$new(method = "any", uri = "http://foo.bar",
    options = list(headers = headers))
rs
```
```r
x$matches(rs)

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

## End(Not run)
```

---

**RequestRegistry**

**Description**

keeps track of HTTP requests

**Public fields**

request_signatures a HashCounter object

**Methods**

**Public methods:**

- `RequestRegistry$print()`
- `RequestRegistry$reset()`
- `RequestRegistry$register_request()`
- `RequestRegistry$times_executed()`
- `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:*

```r
RequestRegistry$register_request(request)
```

*Arguments:*

- `request` a character string of the request, serialized from a `RequestSignature$new(...)$to_s()`

*Returns:* nothing returned; registers the request

Method `times_executed()`: How many times has a request been made

*Usage:*

```r
RequestRegistry$times_executed(request_pattern)
```

*Arguments:*

- `request_pattern` an object of class `RequestPattern`

*Details:* if no match is found for the request pattern, 0 is returned

*Returns:* integer, the number of times the request has been made

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

*Usage:*

```r
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$new()
z1 <- RequestSignature$new("get", "http://scottchamberlain.info")
z2 <- RequestSignature$new("post", "https://httpbin.org/post")
x$register_request(request = z1)
x$register_request(request = z1)
x$register_request(request = z2)
# print method to list requests
x

# more complex requests
w <- RequestSignature$new(
  method = "get",
  uri = "https://httpbin.org/get",
  options = list(headers = list("User-Agent" = "foobar", stuff = "things"))
)
w$to_s()
x$register_request(request = w)
x
```
# RequestSignature

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
- **fields** (various) request body details
- **output** (various) request output details, disk, memory, etc
Methods

Public methods:
- `RequestSignature$new()
- `RequestSignature$print()
- `RequestSignature$to_s()
- `RequestSignature$clone()

Method new(): Create a new RequestSignature object

Usage:
`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**

request_registry()

request_registry_clear()
Response

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

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()
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?

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()
Returns: (character) a 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
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

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

x$set_response_headers(list('Host' = "httpbin.org"))

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)
- `uri` (xx)
- `uri_regex` (xx)
- `uri_parts` (xx)
- `host` (xx)
- `query` (xx)
Methods

Public methods:
- `StubbedRequest$new()`
- `StubbedRequest$print()`
- `StubbedRequest$with()`
- `StubbedRequest$to_return()`
- `StubbedRequest$to_timeout()`
- `StubbedRequest$to_raise()`
- `StubbedRequest$to_s()`
- `StubbedRequest$reset()`
- `StubbedRequest$clone()`

Method `new()`: Create a new StubbedRequest object

Usage:
```r
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. `webmockr` can match uri's without the "http" scheme, but does not match if the scheme is "https". required, unless `uri_regex` given. See `UriPattern` for more.
- `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:
```r
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,
    basic_auth = 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.
basic_auth (character) basic authentication. 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 reset(): Reset the counter for the stub
Usage:
StubbedRequest$reset()
Returns: nothing returned; resets stub counter to no requests

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

Usage:
StubbedRequest$clone(deep = FALSE)

Arguments:
deepl Whether to make a deep clone.

See Also

stub_request()

Examples

```r
## 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$to_return(status = 200, body = "foobar", headers = list(a = 5))
x
x$to_s()

# many to_return's
x <- StubbedRequest$new(method = "get", uri = "httpbin.org")
x$to_return(status = 200, body = "foobar", headers = list(a = 5))
x$to_return(status = 200, body = "bears", headers = list(b = 6))
x
x$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
x$to_s()

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

# basic auth
x <- StubbedRequest$new(method = "get", uri = "api.crossref.org")
x$with(basic_auth = c("foo", "bar"))
x$to_s()
x

# 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$url_regex
x$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)

<table>
<thead>
<tr>
<th>StubCounter</th>
<th>StubCounter</th>
</tr>
</thead>
</table>

**Description**

hash with counter to store requests and count number of requests made against the stub

**Public fields**

hash (list) a list for internal use only, with elements key, sig, and count
Methods

Public methods:

- `StubCounter$put()`
- `StubCounter$count()`
- `StubCounter$clone()`

Method `put()`: Register a request by its key

Usage:
`StubCounter$put(x)`

Arguments:
- `x` an object of class `RequestSignature`

Returns: nothing returned; registers request & iterates internal counter

Method `count()`: Get the count of number of times any matching request has been made against this stub

Usage:
`StubCounter$count()`

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

Usage:
`StubCounter$clone(deep = FALSE)`

Arguments:
- `deep` Whether to make a deep clone.

Examples

```r
x <- StubCounter$new()
x
x$hash
x$count()
z <- RequestSignature$new(method = "get", uri = "https://httpbin.org/get")
x$put(z)
x$count()
x$put(z)
x$count()
```

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, count = TRUE)

Arguments:
- request_signature an object of class RequestSignature
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(), stub_registry_clear()

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$new()
reg$register_stub(stub = stub1)
reg$register_stub(stub = stub2)
reg
reg$request_stubs

## End(Not run)

### stub_registry

**stub_registry**

List stubs in the 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_request

Description
Clear an http request

Usage
stub_request()
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).

Note on wi_th(): If you pass query values are coerced to character class in the recorded stub. You can pass numeric, integer, etc., but all will be coerced to character.

See wi_th() for details on request body/query/headers and to_return() for details on how response status/body/headers are handled

Value

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

uri vs. uri_regex

When you use uri, we compare the URIs without query params AND also the query params themselves without the URIs.

When you use uri_regex we don’t compare URIs and query params; we just use your regex string defined in uri_regex as the pattern for a call to grepl

Mocking writing to disk

See mocking-disk-writing

Note

Trailing slashes are dropped from stub URIs before matching

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") %>%
```
```r
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(crul)
x <- crul::HttpClient$new(url = "https://httpbin.org")
crul::mock()
x$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'))
## with crul
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'))
## with httr
```

library(httr)
httr_mock()
POST('https://example.com', body = list(foo = 'bar'))
PUT('https://google.com', 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(crun)
x <- crun::HttpClient$new(url = "https://httpbin.org", headers = headers)
x$post('post')
x$put('put', body = list(foo = 'bar'))
x$get('put', query = list(stuff = 3423234L))

# many responses
## the first response matches the first to_return call, and so on
stub_request("get", "https://httpbin.org/get") %>%
  to_return(status = 200, body = "foobar", headers = list(a = 5)) %>%
  to_return(status = 200, body = "bears", headers = list(b = 6))
con <- crun::HttpClient$new(url = "https://httpbin.org")
con$get("get")$parse("UTF-8")

## OR, use times with to_return() to repeat the same response many times
library(fauxpas)
stub_request("get", "https://httpbin.org/get") %>%
  to_return(status = 200, body = "apple-pie", times = 2) %>%
  to_raise(HTTPUnauthorized)
con$get("get")$parse("UTF-8")
con$get("get")$parse("UTF-8")
con$get("get")$parse("UTF-8")

# 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

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

... One or more HTTP exceptions from the fauxpas package. Run grep("HTTP*",
getNamespaceExports("fauxpas"), value = TRUE) for a list of possible exceptions

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

Raise vs. Return

to_raise() always raises a stop condition, while to_return(status=xyz) only sets the status code on the returned HTTP response object. So if you want to raise a stop condition then to_raise() is what you want. But if you don’t want to raise a stop condition use to_return(). Use cases for each vary. For example, in a unit test you may have a test expecting a 503 error; in this case to_raise() makes sense. In another case, if a unit test expects to test some aspect of an HTTP response object that httr, httr2, or crul typically returns, then you’ll want to_return().

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(), times = 1)
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
times (integer) number of times the given response should be returned; default: 1. value must be greater than or equal to 1. Very large values probably don’t make sense, but there’s no maximum value. See Details.

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 conntetion 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

multiple to_return()

You can add more than one to_return() to a webmockr stub (including to_raise(), to_timeout()). Each one is a HTTP response returned. That is, you’ll match to an HTTP request based on stub_request() and wi_th(); the first time the request is made, the first response is returned; the second time the request is made, the second response is returned; and so on.

Be aware that webmockr has to track number of requests (see request_registry()), and so if you use multiple to_return() or the times parameter, you must clear the request registry in order to go back to mocking responses from the start again. webmockr_reset() clears the stub registry and the request registry, after which you can use multiple responses again (after creating your stub(s) again of course)

Raise vs. Return

to_raise() always raises a stop condition, while to_return(status=xyz) only sets the status code on the returned HTTP response object. So if you want to raise a stop condition then to_raise() is what you want. But if you don’t want to raise a stop condition use to_return(). Use cases for each vary. For example, in a unit test you may have a test expecting a 503 error; in this case to_raise() makes sense. In another case, if a unit test expects to test some aspect of an HTTP response object that httr, httr2, or crul typically returns, then you’ll want to_return().
to_timeout

Set timeout as an expected return on a match

description
Set timeout as an expected return on a match

usage
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()
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 configuration**

**Description**

webmockr configuration

**Usage**

```r
webmockr_configure(
  allow_net_connect = FALSE,
  allow_localhost = FALSE,
  allow = NULL,
  show_stubbing_instructions = TRUE
)
```

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)
showStubbingInstructions
  (logical) Default: TRUE. If FALSE, stubbing instructions are not shown
uri
  (character) a URI/URL as a character string - to determine whether or not it is allowed

webmockr_allow_net_connect

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)
```

---

webmockr_reset webmockr_reset

Description

Clear all stubs and the request counter

Usage

webmockr_reset()

Details

this function runs stub_registry_clear() and request_registry_clear() - so you can run those two yourself to achieve the same thing
**wi_th**

**Value**

nothing

**See Also**

stub_registry_clear() request_registry_clear()

**Examples**

```r
# webmockr_reset()
```

---

**Set additional parts of a stubbed request**

**Description**

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

**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, basic_auth. See Details.
- `.list` named list, has to be one of query, body, headers and/or basic_auth. 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**

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

Values for query, body, headers, and basic_auth:

- query: (list) a named list. values are coerced to character class in the recorded stub. You can pass numeric, integer, etc., but all will be coerced to character.
- body: various, including character string, list, raw, numeric, upload (`crul::upload()`, `httr::upload_file()`, `curl::form_file()`), or `curl::form_data()` they both create the same object in the end)
- headers: (list) a named list
- basic_auth: (character) a length two vector, username and password. authentication type (basic/digest/ntlm/etc.) is ignored. that is, mocking authentication right now does not take into account the authentication type. We don’t do any checking of the username/password except to detect edge cases where for example, the username/password were probably not set by the user on purpose (e.g., a URL is picked up by an environment variable)
Note that there is no regex matching on query, body, or headers. They are tested for matches in the following ways:

- **query**: compare stubs and requests with `identical()`. This compares named lists, so both list names and values are compared.
- **body**: varies depending on the body format (list vs. character, etc.)
- **headers**: compare stub and request values with `==`. List names are compared with `%in%`. `basic_auth` is included in headers (with the name `Authorization`).

**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 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 = curl::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")))

# basic authentication
wi_th(req, basic_auth = c("user", "pass"))
wi_th(req, basic_auth = c("user", "pass"), headers = list(foo = "bar"))
Index

* request-registry
  HashCounter, 10
  request_registry, 24
  RequestRegistry, 20

* stub-registry
  remove_request_stub, 17
  stub_registry, 37
  stub_registry_clear, 38
  StubRegistry, 34

Adapter (CrulAdapter), 6
BodyPattern, 19
build_crul_request, 2
build_crul_response, 3
build_httr2_request, 3
build_httr2_response, 4
build_httr_request, 5
build_httr_response, 5
crul::upload(), 47
CrulAdapter, 6, 12
curl::form_data(), 47
curl::form_file(), 47
disable (enable), 9
disable(), 45
enable, 9
enable(), 45
enabled (enable), 9
grepl, 39
HashCounter, 10, 21, 25
HeadersPattern, 19
HttpLibAdapterRegistry, 10, 11
httr2_mock, 12
Httr2Adapter (CrulAdapter), 6
http::upload_file(), 47
httr_mock, 13
HttrAdapter (CrulAdapter), 6
MethodPattern, 19
mock_file, 16
mock_file(), 39, 43
mocking-disk-writing, 13, 39
pluck_body, 16
remove_request_stub, 17, 36–38
remove_request_stub(), 39
request_registry, 11, 21, 24
request_registry(), 43
request_registry_clear
  (request_registry), 24
request_registry_clear(), 46, 47
RequestPattern, 17
RequestRegistry, 11, 20, 25
RequestSignature, 18, 22, 35, 36
Response, 25
stub_registry, 17, 36, 37, 38
stub_registry(), 21, 39
stub_registry_clear, 17, 36, 37, 38
stub_registry_clear(), 39, 46, 47
stub_request, 38
stub_request(), 29, 32, 42, 44, 48
StubbedRequest, 29, 34–36, 39
StubCounter, 33
StubRegistry, 17, 21, 34, 37, 38
to_raise, 41
to_raise(), 39, 43
to_return, 42
to_return(), 39, 43
to_return_(), 47
to_timeout, 44
to_timeout(), 39, 43
UriPattern, 19, 30, 38
webmockr-defunct, 45
webmockr::Adapter, 6, 7
webmockr_allow_net_connect
  (webmockr_configure), 45
webmockr_configuration
  (webmockr_configure), 45
webmockr_configure, 45
webmockr_configure_reset
  (webmockr_configure), 45
webmockr_disable(), 45
webmockr_disable_net_connect
  (webmockr_configure), 45
webmockr_enable(), 45
webmockr_net_connect_allowed
  (webmockr_configure), 45
webmockr_reset, 46
webmockr_reset(), 43
wi_th, 47
wi_th(), 39, 45
wi_th_, 45