Package 'webmockr'

May 8, 2026

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 2.2.0

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

Depends R(>= 4.1.0)

Imports curl, jsonlite, magrittr (>= 1.5), R6 (>= 2.1.3), urltools (>= 1.6.0), fauxpas, rlang, cli

Suggests testthat (>= 3.0.0), xml2, crul, httr, httr2, diffobj, withr

RoxygenNote 7.3.2

Config/testthat/edition 3

Config/testthat/parallel true

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] (ORCID:
<https://orcid.org/0000-0003-1444-9135>),
Aaron Wolen [ctb] (ORCID: <https://orcid.org/0000-0003-2542-2202>),
rOpenSci [fnd] (ROR: <https://ror.org/019jywm96>)

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

Repository CRAN

Date/Publication 2025-07-21 05:00:02 UTC

Contents

`enable` *Enable or disable webmockr*

Description

Enable or disable webmockr

Usage

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

enabled(adapter = "crul")

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 HttpLibAdapaterRegistry object. `enabled` returns a single boolean

---

`httr2_mock` *Turn on* `httr2` *mocking*

Description

Sets a callback that routes `httr2` requests through `webmockr`

Usage

```
httr2_mock(on = TRUE)
```

Arguments

on	(logical) TRUE to turn on, FALSE to turn off. default: TRUE

Value

Silently returns TRUE when enabled and FALSE when disabled.

httr_mock *Turn on* httr *mocking*

Description

Sets a callback that routes `httr` requests 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.

including *Partially match request query parameters or request bodies*

Description

For use inside `wi_th()`

Usage

```
including(x)

excluding(x)
```

Arguments

x	(list) a list; may support other classes in the future

Value

same as `x`, but with two attributes added:

- partial_match: always `TRUE`
- partial_type: the type of match, one of `include` or `exclude`

Headers

Matching on headers already handles partial matching. That is, `wi_th(headers = list(Fruit = "pear"))` matches any request that has any request header that matches - the request can have other request headers, but those don't matter as long as there is a match. These helpers (`including`/`excluding`) are needed for query parameters and bodies because by default matching must be exact for those.

Examples

```
including(list(foo = "bar"))
excluding(list(foo = "bar"))

# get just keys by setting values as NULL
including(list(foo = NULL, bar = NULL))

# in a stub
req <- stub_request("get", "https://httpbin.org/get")
req

## query
wi_th(req, query = list(foo = "bar"))
wi_th(req, query = including(list(foo = "bar")))
wi_th(req, query = excluding(list(foo = "bar")))

## body
wi_th(req, body = list(foo = "bar"))
wi_th(req, body = including(list(foo = "bar")))
wi_th(req, body = excluding(list(foo = "bar")))

# cleanup
stub_registry_clear()
```

`last_request` *Get the last HTTP request made*

Description

Get the last HTTP request made

Usage

```
last_request()
```

Value

`NULL` if no requests registered; otherwise the last registered request made as a `RequestSignature` class

Examples

```
# no requests
request_registry_clear()
last_request()

# a request is found
enable()
stub_request("head", "https://nytimes.com")
library(crul)
crul::ok("https://nytimes.com")
last_request()

# cleanup
request_registry_clear()
stub_registry_clear()
```

`last_stub` *Get the last stub created*

Description

Get the last stub created

Usage

```
last_stub()
```

Value

`NULL` if no stubs found; otherwise the last stub created as a `StubbedRequest` class

Examples

```
# no requests
stub_registry_clear()
last_stub()

# a stub is found
stub_request("head", "https://nytimes.com")
last_stub()

stub_request("post", "https://nytimes.com/stories")
last_stub()

# cleanup
stub_registry_clear()
```

mocking-disk-writing *Mocking writing to disk*

Description

Mocking writing to disk

Examples

```
# 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)
stub_registry_clear()

# 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 errror
out <- GET("https://httpbin.org/get", write_disk(f, overwrite = TRUE))
out
out$content
content(out, "text", encoding = "UTF-8")
stub_registry_clear()
```

```
# 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$body
out$headers
readLines(out$body)
stub_registry_clear()

# 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)
stub_registry_clear()

# 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")
stub_registry_clear()

# 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)
stub_registry_clear()

# disable mocking
disable()
```

`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\"}")
```

remove_request_stub *Remove a request stub*

Description

Remove a request stub

Usage

```
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(), stub_registry_clear()

Examples

```
(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,
  basic_auth = 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

`basic_auth` (list) vector of length 2 (username, password), 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

---

`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()
```

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()
```

`stub_body_diff` *Get a diff of a stub request body and a request body from an http request*

Description

Requires the Suggested package `diffobj`

Usage

```
stub_body_diff(stub = last_stub(), request = last_request())
```

Arguments

`stub`	object of class `StubbedRequest`. required. default is to call `last_stub()`, which gets the last stub created
`request`	object of class `RequestSignature`. required. default is to call `last_request()`, which gets the last stub created

Details

Returns error message if either `stub` or `request` are `NULL`. Even though you may not intentionally pass in a `NULL`, the return values of `last_stub()` and `last_request()` when there's nothing found is `NULL`.

Under the hood the Suggested package `diffobj` is used to do the comparison.

Value

object of class `Diff` from the **diffobj** package

See Also

`webmockr_configure()` to toggle `webmockr` showing request body diffs when there's not a match. `stub_body_diff()` is offered as a manual way to compare requests and stubs - whereas turning on with `webmockr_configure()` will do the diff for you.

Examples

```
# stops with error if no stub and request
request_registry_clear()
stub_registry_clear()
stub_body_diff()

# Gives diff when there's a stub and request found - however, no request body
stub_request("get", "https://hb.opencpu.org/get")
enable()
library(crul)
HttpClient$new("https://hb.opencpu.org")$get(path = "get")
stub_body_diff()

# Gives diff when there's a stub and request found - with request body
stub_request("post", "https://hb.opencpu.org/post") %>%
  wi_th(body = list(apple = "green"))
enable()
library(crul)
HttpClient$new("https://hb.opencpu.org")$post(
  path = "post", body = list(apple = "red")
)
stub_body_diff()

# Gives diff when there's a stub and request found - with request body
stub_request("post", "https://hb.opencpu.org/post") %>%
  wi_th(body = "the quick brown fox")
HttpClient$new("https://hb.opencpu.org")$post(
  path = "post", body = "the quick black fox"
)
stub_body_diff()
```

---

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

```
# 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` *stub_registry_clear*

Description

Clear all stubs in the stub registry

Usage

```
stub_registry_clear()
```

Value

an empty list invisibly

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 or partial uri. **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. See the "uri vs. uri_regex" section
`uri_regex`	(character) A URI represented as regex. required, if `uri` not given. See examples and the "uri vs. uri_regex" section

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

Error handling

To construct stubs, one uses `stub_request()` first - which registers the stub in the stub registry. Any additional calls to modify the stub with for example `wi_th()` or `to_return()` can error. In those error cases we ideally want to remove (unregister) the stub because you certainly don't want a registered stub that is not exactly what you intended.

When you encounter an error creating a stub you should see a warning message that the stub has been removed, for example:

```
stub_request("get", "https://httpbin.org/get") %>%
  wi_th(query = mtcars)
#> Error in `wi_th()`:
#> ! z$query must be of class list or partial
#> Run `rlang::last_trace()` to see where the error occurred.
#> Warning message:
#> Encountered an error constructing stub
#> • Removed stub
#> • To see a list of stubs run stub_registry()
```

Note

Trailing slashes are dropped from stub URIs before matching

See Also

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

Examples

```
# 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()

# clear all stubs
stub_registry()
stub_registry_clear()
```

`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 connetion 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()`.

Note

see more examples in `stub_request()`

Examples

```
# first, make a stub object
foo <- function() {
  stub_request("post", "https://httpbin.org/post")
}

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

# .list - pass in a named list instead
foo() %>% to_return(.list = list(body = list(foo = "bar")))

# multiple responses using chained `to_return()`
foo() %>%
  to_return(body = "stuff") %>%
  to_return(body = "things")

# many of the same response using the times parameter
foo() %>% to_return(body = "stuff", times = 3)
```

---

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()`

`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` *webmockr configuration*

Description

webmockr configuration

Usage

```
webmockr_configure(
  allow_net_connect = FALSE,
  allow_localhost = FALSE,
  allow = NULL,
  show_stubbing_instructions = TRUE,
  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)
show_stubbing_instructions	(logical) Default: TRUE. If FALSE, stubbing instructions are not shown
show_body_diff	(logical) Default: FALSE. If TRUE show's a diff of the stub's request body and the http request body. See also stub_body_diff() for manually comparing request and stub bodies. Under the hood the Suggested package diffobj is required to do the comparison.
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

```
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")

# show body diff
webmockr_configure(show_body_diff = TRUE)

# cleanup
webmockr_configure_reset()
```

`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

Value

nothing

See Also

`stub_registry_clear()` `request_registry_clear()`

Examples

```
# webmockr_reset()
```

wi_th *Set additional parts of a stubbed request*

Description

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

Usage

```
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

with 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). for the special case of an empty request body use NA instead of NULL because with NULL we can't determine if the user did not supply a body or they supplied NULL to indicate an empty body.
- headers: (list) a named list
- basic_auth: (character) a length two vector, username and password. 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). Only basic authentication supported https://en.wikipedia.org/wiki/Basic_access_authentication.

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()`

See Also

`including()`

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 = 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")))

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

# partial matching, query params
## including
wi_th(req, query = including(list(foo = "bar")))
## excluding
wi_th(req, query = excluding(list(foo = "bar")))

# partial matching, body
## including
wi_th(req, body = including(list(foo = "bar")))
```

```
## excluding
wi_th(req, body = excluding(list(foo = "bar")))

# basic auth
## including
wi_th(req, body = including(list(foo = "bar")))
## excluding
wi_th(req, body = excluding(list(foo = "bar")))
```

Index