Package 'vcr'

February 12, 2019

Title Record 'HTTP' Calls to Disk

Description Record test suite 'HTTP' requests and replays them during future runs. A port of the Ruby gem of the same name (<https://github.com/vcr/vcr/>). Works by hooking into the 'webmockr' R package for matching 'HTTP' requests by various rules ('HTTP' method, 'URL', query parameters, headers, body, etc.), and then caching real 'HTTP' responses on disk in 'cassettes'. Subsequent 'HTTP' requests matching any previous requests in the same 'cassette' use a cached 'HTTP' response.

Version 0.2.6

URL https://github.com/ropensci/vcr/ (devel)
https://ropensci.github.io/http-testing-book/ (user manual)

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

License MIT + file LICENSE

Encoding UTF-8

LazyData true

VignetteBuilder knitr

Imports crul (>= 0.7.0), httr, webmockr (>= 0.3.4), urltools, yaml, R6, lazyeval, base64enc

Suggests jsonlite, testthat, knitr, rmarkdown, desc, crayon, cli

RoxygenNote 6.1.1

X-schema.org-applicationCategory Web

X-schema.org-keywords http, https, API, web-services, curl, mock, mocking, 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>)

Maintainer Scott Chamberlain <sckott@protonmail.com>

Repository CRAN

Date/Publication 2019-02-12 10:30:08 UTC

R topics documented:

`as.cassette` *Coerce names, etc. to cassettes*

Description

Coerce names, etc. to cassettes

Coerce to a cassette path

Usage

```
as.cassette(x, ...)

as.cassettepath(x)
```

Arguments

`x`	Input, a cassette name (character), or something that can be coerced to a cassette
`...`	further arguments passed on to `cassettes()` or [read_cassette_meta()

Value

a cassette of class `Cassette`

Examples

```
## Not run:
vcr_configure(dir = tempfile())
insert_cassette("foobar")
cassettes(on_disk = FALSE)
cassettes(on_disk = TRUE)
as.cassette("foobar", on_disk = FALSE)
eject_cassette() # eject the current cassette

# cleanup
unlink(file.path(tempfile(), "foobar.yml"))

## End(Not run)
```

---

cassettes *List cassettes, get current cassette, etc.*

Description

List cassettes, get current cassette, etc.

Usage

```
cassettes(on_disk = TRUE, verb = FALSE)

current_cassette()

cassette_path()
```

Arguments

`on_disk`	(logical) Check for cassettes on disk + cassettes in session (`TRUE`), or check for only cassettes in session (`FALSE`). Default: `TRUE`
`verb`	(logical) verbose messages

Details

- `cassettes()`: returns cassettes found in your R session, you can toggle whether we pull from those on disk or not
- `current_cassette()`: returns an empty list when no cassettes are in use, while it returns the current cassette (a `Cassette` object) when one is in use
- `cassette_path()`: just gives you the current directory path where cassettes will be stored

Examples

```
vcr_configure(dir = tempdir())

# list all cassettes
cassettes()
cassettes(on_disk = FALSE)

# list the currently active cassette
insert_cassette("stuffthings")
current_cassette()
eject_cassette()

cassettes()
cassettes(on_disk = FALSE)

# list the path to cassettes
cassette_path()
vcr_configure(file.path(tempdir(), "foo"))
cassette_path()

vcr_configure_reset()
```

---

crul_request *An HTTP request as prepared by the* **crul** *package*

Description

The object is a list, and is the object that is passed on to **webmockr** and **vcr** instead of routing through **crul** as normal. Used in examples/tests.

Format

A list

---

eject_cassette *Eject a cassette*

Description

Eject a cassette

Usage

```
eject_cassette(cassette = NULL, options = list(),
  skip_no_unused_interactions_assertion = NULL)
```

Arguments

`cassette`	(character) a single cassette names to eject
`options`	(list) a list of options to apply to the eject process
`skip_no_unused_interactions_assertion`	(logical) If `TRUE`, this will skip the "no unused HTTP interactions" assertion enabled by the `allow_unused_http_interactions = FALSE` cassette option. This is intended for use when your test has had an error, but your test framework has already handled it - IGNORED FOR NOW

Value

The ejected cassette if there was one

See Also

`use_cassette()`, `insert_cassette()`

Examples

```
vcr_configure(dir = tempdir())
insert_cassette("hello")
(x <- current_cassette())

# by default does current cassette
x <- eject_cassette()
x
# can also select by cassette name
# eject_cassette(cassette = "hello")
```

`HTTPInteraction` *HTTPInteraction class*

Description

HTTPInteraction class

Arguments

`request`	A `Request` class object
`response`	A `VcrResponse` class object
`recorded_at`	Time http interaction recorded at

Details

Methods

`to_hash()` Create a hash from the HTTPInteraction object

`from_hash(hash)` Create a HTTPInteraction object from a hash

Examples

```
## Not run:
# make the request
library(vcr)
url <- "https://eu.httpbin.org/post"
body <- list(foo = "bar")
cli <- crul::HttpClient$new(url = url)
res <- cli$post(body = body)

# build a Request object
(request <- Request$new("POST", uri = url,
  body = body, headers = res$response_headers))
# build a VcrResponse object
(response <- VcrResponse$new(
   res$status_http(),
   res$response_headers,
   res$parse("UTF-8"),
   res$response_headers$status))

# make HTTPInteraction object
(x <- HTTPInteraction$new(request = request, response = response))
x$recorded_at
x$to_hash()

# make an HTTPInteraction from a hash with the object already made
x$from_hash(x$to_hash())

# Make an HTTPInteraction from a hash alone
my_hash <- x$to_hash()
HTTPInteraction$new()$from_hash(my_hash)

## End(Not run)
```

HTTPInteractionList *HTTPInteractionList class*

Description

HTTPInteractionList class

Arguments

`interactions`	(list) list of interaction class objects
`request_matchers`	(character) vector of request matchers
`allow_playback_repeats`	whether to allow playback repeats or not
`parent_list`	A list for empty objects, see NullList

`used_interactions`
: (list) Interactions that have been used. That is, interactions that are on disk in the current cassette, and a request has been made that matches that interaction

`request`
: The request from an object of class `HttpInteraction`

Details

Methods

`response_for(request)` Check if there's a matching interaction, returns a response object

`has_interaction_matching(request)` Check if has a matching interaction. returns boolean

`has_used_interaction_matching(request)` check if has used interactions matching a given request. returns boolean

`remaining_unused_interaction_count()` Number of unused interactions. returns numeric

`assert_no_unused_interactions()` Checks if there are no unused interactions left. returns boolean

Private Methods

`has_unused_interactions()` Are there any unused interactions? returns boolean

`matching_interaction_index_for()` asdfadf

`matching_used_interaction_for(request)` asdfadfs

`interaction_matches_request(request, interaction)` Check if a request matches an interaction (logical)

`from_hash()` Get a hash back.

`request_summary(z)` Get a request summary (character)

`response_summary(z)` Get a response summary (character)

Examples

```
## Not run:
vcr_configure(
 dir = tempdir(),
 record = "once"
)

# make interactions
## make the request
### turn off mocking
crul::mock(FALSE)
url <- "https://eu.httpbin.org/post"
cli <- crul::HttpClient$new(url = url)
res <- cli$post(body = list(a = 5))

## request
(request <- Request$new("POST", url, body, res$headers))
## response
(response <- VcrResponse$new(
```

```
   res$status_http(),
   res$response_headers,
   res$parse("UTF-8"),
   res$response_headers$status))
## make an interaction
(inter <- HTTPInteraction$new(request = request, response = response))

# make an interactionlist
(x <- HTTPInteractionList$new(
   interactions = list(inter),
   request_matchers = vcr_configuration()$match_requests_on
))
x$interactions
x$request_matchers
x$parent_list
x$parent_list$response_for()
x$parent_list$has_interaction_matching()
x$parent_list$has_used_interaction_matching()
x$parent_list$remaining_unused_interaction_count()
x$used_interactions
x$allow_playback_repeats
x$interactions
x$response_for(request)

## End(Not run)
```

`http_interactions` *Get the http interactions of the current cassette*

Description

Get the http interactions of the current cassette

Usage

```
http_interactions()
```

Value

object of class `HTTPInteractionList` if there is a current cassette in use, or `NullList` if no cassette in use

Examples

```
## Not run:
vcr_configure(dir = tempdir())
insert_cassette("foo_bar")
webmockr::webmockr_allow_net_connect()
library(crul)
cli <- crul::HttpClient$new("https://eu.httpbin.org/get")
```

```
one <- cli$get(query = list(a = 5))
z <- http_interactions()
z
z$interactions
z$used_interactions
eject_cassette("foo_bar")
# cleanup
unlink(file.path(tempdir(), "foo_bar.yml"))

## End(Not run)
```

`insert_cassette` *Insert a cassette to record HTTP requests*

Description

Insert a cassette to record HTTP requests

Usage

```
insert_cassette(name, record = "once", match_requests_on = c("method",
  "uri"), update_content_length_header = FALSE,
  allow_playback_repeats = FALSE, serialize_with = "yaml",
  persist_with = "FileSystem", preserve_exact_body_bytes = FALSE,
  ignore_cassettes = FALSE)
```

Arguments

`name`	The name of the cassette. vcr will sanitize this to ensure it is a valid file name.
`record`	The record mode. Default: "once". In the future we'll support "once", "all", "none", "new_episodes". See recording for more information
`match_requests_on`	List of request matchers to use to determine what recorded HTTP interaction to replay. Defaults to `["method", "uri"]`. The built-in matchers are "method", "uri", "host", "path", "headers" and "body"
`update_content_length_header`	(logical) Whether or not to overwrite the `Content-Length` header of the responses to match the length of the response body. Default: `FALSE`
`allow_playback_repeats`	(logical) Whether or not to allow a single HTTP interaction to be played back multiple times. Default: `FALSE`.
`serialize_with`	(character) Which serializer to use. Valid values are "yaml" (default), the only one supported for now.
`persist_with`	(character) Which cassette persister to use. Default: "file_system". You can also register and use a custom persister.

preserve_exact_body_bytes
: (logical) Whether or not to base64 encode the bytes of the requests and responses for this cassette when serializing it. See also preserve_exact_body_bytes in vcr_configure(). Default: FALSE

ignore_cassettes
: (logical) turn **vcr** off and ignore cassette insertions (so that no error is raised). Default: FALSE

Value

an object of class Cassette

See Also

use_cassette(), eject_cassette()

Examples

```
## Not run:
library(vcr)
library(crul)
vcr_configure(dir = tempdir())
webmockr::webmockr_allow_net_connect()

(x <- insert_cassette(name = "leo5"))
current_cassette()
x$new_recorded_interactions
cli <- crul::HttpClient$new(url = "https://httpbin.org")
cli$get("get")
x$new_recorded_interactions
# very important when using inject_cassette: eject when done
x$eject() # same as eject_cassette("leo5")

# cleanup
unlink(file.path(tempdir(), "leo5.yml"))

## End(Not run)
```

---

lightswitch *Turn vcr on and off, check on/off status, and turn off for a given http call*

Description

Turn vcr on and off, check on/off status, and turn off for a given http call

Usage

```
turned_off(..., ignore_cassettes = FALSE)

turn_on()

turned_on()

turn_off(ignore_cassettes = FALSE)
```

Arguments

`...`	Any block of code to run, presumably an http request
`ignore_cassettes`	(logical) Controls what happens when a cassette is inserted while vcr is turned off. If `TRUE` is passed, the cassette insertion will be ignored; otherwise an error will be raised.

Details

- `turned_off()` - Turns vcr off for the duration of a block.
- `turn_off()` - Turns vcr off, so that it no longer handles every HTTP request
- `turn_on()` - turns vcr on
- `turned_on()` - Asks if vcr is turned on, gives a boolean

Examples

```
## Not run:
vcr_configure(dir = tempdir())

turn_on()
turned_on()
turn_off()

# turn off for duration of a block
library(crul)
turned_off({
 res <- HttpClient$new(url = "https://eu.httpbin.org/get")$get()
})
res

# turn completely off
turn_off()
library(webmockr)
crul::mock()
# HttpClient$new(url = "https://eu.httpbin.org/get")$get(verbose = TRUE)
turn_on()

## End(Not run)
```

NullList *Null list, an empty HTTPInteractionList object*

Description

Null list, an empty HTTPInteractionList object

Usage

```
NullList
```

Format

An object of class `R6ClassGenerator` of length 24.

real_http_connections_allowed *Are real http connections allowed?*

Description

Are real http connections allowed?

Usage

```
real_http_connections_allowed()
```

Value

boolean, `TRUE` if real HTTP requests allowed; `FALSE` if not

Examples

```
real_http_connections_allowed()
```

`recording` *vcr recording options*

Description

vcr recording options

once

The `once` record mode will:

- Replay previously recorded interactions.
- Record new interactions if there is no cassette file.
- Cause an error to be raised for new requests if there is a cassette file.

It is similar to the `new_episodes` record mode, but will prevent new, unexpected requests from being made (i.e. because the request URI changed or whatever).

`once` is the default record mode, used when you do not set one.

none

The `none` record mode will:

- Replay previously recorded interactions.
- Cause an error to be raised for any new requests.

This is useful when your code makes potentially dangerous HTTP requests. The `none` record mode guarantees that no new HTTP requests will be made.

new_episodes

The `new_episodes` record mode will:

- Record new interactions.
- Replay previously recorded interactions.

It is similar to the `once` record mode, but will **always** record new interactions, even if you have an existing recorded one that is similar (but not identical, based on the `match_request_on` option).

all

The `all` record mode will:

- Record new interactions.
- Never replay previously recorded interactions.

This can be temporarily used to force **vcr** to re-record a cassette (i.e. to ensure the responses are not out of date) or can be used when you simply want to log all HTTP requests.

`request-matching` *vcr request matching*

Description

There are a number of options, some of which are on by default, some of which can be used together, and some alone.

matching on method

Use the **method** request matcher to match requests on the HTTP method (i.e. GET, POST, PUT, DELETE, etc). You will generally want to use this matcher. The **method** matcher is used (along with the **uri** matcher) by default if you do not specify how requests should match.

matching on uri

Use the **uri** request matcher to match requests on the request URI. The **uri** matcher is used (along with the **method** matcher) by default if you do not specify how requests should match.

matching on host

Use the **host** request matcher to match requests on the request host. You can use this (alone, or in combination with **path**) as an alternative to **uri** so that non-deterministic portions of the URI are not considered as part of the request matching.

matching on path

Use the **path** request matcher to match requests on the path portion of the request URI. You can use this (alone, or in combination with **host**) as an alternative to **uri** so that non-deterministic portions of the URI

matching on query string

Use the **query** request matcher to match requests on the query string portion of the request URI. You can use this (alone, or in combination with others) as an alternative to **uri** so that non-deterministic portions of the URI are not considered as part of the request matching.

matching on body

Use the **body** request matcher to match requests on the request body.

matching on headers

Use the **headers** request matcher to match requests on the request headers.

RequestHandler *RequestHandler*

Description

RequestHandler

Arguments

request	The request from an object of class HttpInteraction

Details

Public Methods

handle(request) Top level function to interaction with. Handle the request

Private Methods

request_type(request) Get the request type

externally_stubbed() just returns FALSE

should_ignore() should we ignore the request, depends on request ignorer infrastructure that's not working yet

has_response_stub() Check if there is a matching response stub in the http interaction list

get_stubbed_response() Check for a response and get it

request_summary(request) get a request summary

on_externally_stubbed_request(request) on externally stubbed request do nothing

on_ignored_request(request) on ignored request, do something

on_recordable_request(request) on recordable request, record the request

on_unhandled_request(request) on unhandled request, run UnhandledHTTPRequestError

Examples

```
## Not run:
# record mode: once
vcr_configure(
 dir = tempdir(),
 record = "once"
)

data(crul_request)
crul_request$url$handle <- curl::new_handle()
crul_request
x <- RequestHandler$new(crul_request)
# x$handle()

# record mode: none
```

```
vcr_configure(
 dir = tempdir(),
 record = "none"
)
data(crul_request)
crul_request$url$handle <- curl::new_handle()
crul_request
insert_cassette("testing_record_mode_none", record = "none")
file.path(vcr_c$dir, "testing_record_mode_none.yml")
x <- RequestHandlerCrul$new(crul_request)
# x$handle()
crul_request$url$url <- "https://api.crossref.org/works/10.1039/c8sm90002g/"
crul_request$url$handle <- curl::new_handle()
z <- RequestHandlerCrul$new(crul_request)
# z$handle()
eject_cassette("testing_record_mode_none")

## End(Not run)
```

`RequestHandlerCrul` *RequestHandlerCrul - methods for crul package*

Description

RequestHandlerCrul - methods for crul package

Usage

```
RequestHandlerCrul
```

Format

An object of class `R6ClassGenerator` of length 24.

Details

Public Methods

`handle(request)` Top level function to interaction with. Handle the request

Private Methods

`request_type(request)` Get the request type

`externally_stubbed()` just returns FALSE

`should_ignore()` should we ignore the request, depends on request ignorer infrastructure that's not working yet

`has_response_stub()` Check if there is a matching response stub in the http interaction list

`get_stubbed_response()` Check for a response and get it

`request_summary(request)` get a request summary

`on_externally_stubbed_request(request)` on externally stubbed request do nothing

`on_ignored_request(request)` on ignored request, do something

`on_recordable_request(request)` on recordable request, record the request

`on_unhandled_request(request)` on unhandled request, run UnhandledHTTPRequestError

Examples

```
## Not run:
vcr_configure(
 dir = tempdir(),
 record = "once"
)

data(crul_request)
crul_request$url$handle <- curl::new_handle()
crul_request
x <- RequestHandlerCrul$new(crul_request)
# x$handle()

## End(Not run)
```

`RequestHandlerHttr` *RequestHandlerHttr - methods for httr package*

Description

RequestHandlerHttr - methods for httr package

Usage

```
RequestHandlerHttr
```

Format

An object of class `R6ClassGenerator` of length 24.

Details

Public Methods

`handle(request)` Top level function to interaction with. Handle the request

Private Methods

`request_type(request)` Get the request type

`externally_stubbed()` just returns FALSE

`should_ignore()` should we ignore the request, depends on request ignorer infrastructure that's not working yet

`has_response_stub()` Check if there is a matching response stub in the http interaction list

`get_stubbed_response()` Check for a response and get it

`request_summary(request)` get a request summary

`on_externally_stubbed_request(request)` on externally stubbed request do nothing

`on_ignored_request(request)` on ignored request, do something

`on_recordable_request(request)` on recordable request, record the request

`on_unhandled_request(request)` on unhandled request, run UnhandledHTTPRequestError

Examples

```
## Not run:
vcr_configure(
 dir = tempdir(),
 record = "once"
)

library(httr)
load("~/httr_req.rda")
req
x <- RequestHandlerHttr$new(req)
# x$handle()

## End(Not run)
```

`RequestMatcherRegistry` *RequestMatcherRegistry class*

Description

RequestMatcherRegistry class

Arguments

`name`	matcher name
`func`	function that describes a matcher, should return a single boolean
`r1, r2`	two Request class objects

Details

Methods

`register(name, func)` Register a custom matcher.

`register_built_ins()` Register all built in matchers.

`try_to_register_body_as_json(r1, r2)` Try to register body as JSON.

Examples

```
## Not run:
(x <- RequestMatcherRegistry$new())
x$default_matchers
x$registry

## End(Not run)
```

`testing` *Using* **vcr** *for unit testing*

Description

Using **vcr** for unit testing

Using vcr with testthat

vcr supports use with the **testthat** package. Here's the steps to follow:

First, note that **vcr** only works with a single HTTP client for now: **crul**

- In addition to **testthat**, add **vcr** and **webmockr** to your Suggests in your DESCRIPTION file
- Add a file (named e.g., `vcr-config.R`) to your `tests/testthat/` directory. In that file add your **vcr** configuration settings. See vcr_configure for help on configuration settings.
- For any given test use the following:

```
use_cassette("foobar", {
  aa <- hello::world()
  expect_is(aa, "SomeClass")
  expect_equal(length(aa), 3)
})
```

And the tests will behave as normally.

The first request will make a real HTTP request. Following requests will pull from the cached responses on cassette.

`UnhandledHTTPRequestError` *UnhandledHTTPRequestError*

Description

UnhandledHTTPRequestError

Arguments

request	a request
cassette	a cassette, the current cassette

Details

How this error class is used: If record="once" we trigger this.

Users can use vcr in the context of both use_cassette and insert_cassette.

For the former, all requests go through the call_block But for the latter, requests go through webmockr

Where is one place where we can put UnhandledHTTPRequestError that will handle both use_cassette and insert_cassette?

Error situations where this is invoked

- record=once AND there's a new request that doesn't match the one in the cassette on disk
  - in webmockr: if no stub found and there are recorded interactions on the cassette, and record = once, then error with UnhandledHTTPRequestError
    - but if record != once, then allow it, unless record == none
- others?

Examples

```
vcr_configure(dir = tempdir())
cassettes()
insert_cassette("turtle")
request <- Request$new("post", 'https://eu.httpbin.org/post?a=5',
  "", list(foo = "bar"))

err <- UnhandledHTTPRequestError$new(request)
err$request_description()
err$current_matchers()
err$match_request_on_headers()
err$match_request_on_body()
err$formatted_headers()
cat(err$formatted_headers(), "\n")
cat(err$cassettes_description(), "\n")
cat(err$cassettes_list(), "\n")
err$formatted_suggestions()
cat(err$format_bullet_point('foo bar', 1), "\n")
err$suggestion_for("use_new_episodes")
err$suggestions()
err$no_cassette_suggestions()
err$record_mode_suggestion()
err$has_used_interaction_matching()
err$match_requests_on_suggestion()

# err$construct_message()
```

```
# cleanup
unlink(tempdir())
```

`use_cassette` *Use a cassette*

Description

Use a cassette

Usage

```
use_cassette(name, ..., record = "once",
  match_requests_on = c("method", "uri"),
  update_content_length_header = FALSE, allow_playback_repeats = FALSE,
  serialize_with = "yaml", persist_with = "FileSystem",
  preserve_exact_body_bytes = FALSE)
```

Arguments

`name`	The name of the cassette. vcr will sanitize this to ensure it is a valid file name.
`...`	a block of code to evalulate, wrapped in curly braces. required. if you don't pass a code block you'll get a stop message. if you can't pass a code block use instead `insert_cassette()`
`record`	The record mode. Default: "once". In the future we'll support "once", "all", "none", "new_episodes". See recording for more information
`match_requests_on`	List of request matchers to use to determine what recorded HTTP interaction to replay. Defaults to `["method", "uri"]`. The built-in matchers are "method", "uri", "host", "path", "headers" and "body"
`update_content_length_header`	(logical) Whether or not to overwrite the `Content-Length` header of the responses to match the length of the response body. Default: `FALSE`
`allow_playback_repeats`	(logical) Whether or not to allow a single HTTP interaction to be played back multiple times. Default: `FALSE`.
`serialize_with`	(character) Which serializer to use. Valid values are "yaml" (default), the only one supported for now.
`persist_with`	(character) Which cassette persister to use. Default: "file_system". You can also register and use a custom persister.
`preserve_exact_body_bytes`	(logical) Whether or not to base64 encode the bytes of the requests and responses for this cassette when serializing it. See also `preserve_exact_body_bytes` in `vcr_configure()`. Default: `FALSE`

Details

A run down of the family of top level **vcr** functions

- `use_cassette` Initializes a cassette. Returns the inserted cassette.
- `insert_cassette` Internally used within `use_cassette`
- `eject_cassette` ejects the current cassette. The cassette will no longer be used. In addition, any newly recorded HTTP interactions will be written to disk.

Value

an object of class `Cassette`

Behavior

This function handles a few different scenarios:

- when everything runs smoothly, and we return a `Cassette` class object so you can inspect the cassette, and the cassette is ejected
- when there is an invalid parameter input on cassette creation, we fail with a useful message, we don't return a cassette, and the cassette is ejected
- when there is an error in calling your passed in code block, we return with a useful message, and since we use `on.exit()` the cassette is still ejected even though there was an error, but you don't get an object back

Cassettes on disk

Note that *"eject"* only means that the R session cassette is no longer in use. If any interactions were recorded to disk, then there is a file on disk with those interactions.

Using with tests (specifically testthat)

There's a few ways to get correct line numbers for failed tests and one way to not get correct line numbers:

Correct: Either wrap your `test_that()` block inside your `use_cassette()` block, OR if you put your `use_cassette()` block inside your `test_that()` block put your `testthat` expectations outside of the `use_cassette()` block.

Incorrect: By wrapping the `use_cassette()` block inside your `test_that()` block with your **testthat** expectations inside the `use_cassette()` block, you'll only get the line number that the `use_cassette()` block starts on.

See Also

`insert_cassette()`, `eject_cassette()`

Examples

```
## Not run:
library(vcr)
library(crul)
vcr_configure(dir = tempdir())

use_cassette(name = "apple7", {
  cli <- HttpClient$new(url = "https://httpbin.org")
  resp <- cli$get("get")
})
readLines(file.path(tempdir(), "apple7.yml"))

# preserve exact body bytes - records in base64 encoding
use_cassette("things4", {
  cli <- crul::HttpClient$new(url = "https://httpbin.org")
  bbb <- cli$get("get")
}, preserve_exact_body_bytes = TRUE)
## see the body string value in the output here
readLines(file.path(tempdir(), "things4.yml"))

# cleanup
unlink(file.path(tempdir(), c("things4.yml", "apple7.yml")))


# with httr
library(vcr)
library(httr)
vcr_configure(dir = tempdir(), log = TRUE)

use_cassette(name = "stuff350", {
  res <- GET("https://httpbin.org/get")
})
readLines(file.path(tempdir(), "stuff350.yml"))

use_cassette(name = "catfact456", {
  res <- GET("https://catfact.ninja/fact")
})

# record mode: none
library(crul)
vcr_configure(dir = tempdir())

## make a connection first
conn <- crul::HttpClient$new("https://eu.httpbin.org")
## this errors because 'none' disallows any new requests
# use_cassette("none_eg", (res2 <- conn$get("get")), record = "none")
## first use record mode 'once' to record to a cassette
one <- use_cassette("none_eg", (res <- conn$get("get")), record = "once")
one; res
## then use record mode 'none' to see it's behavior
two <- use_cassette("none_eg", (res2 <- conn$get("get")), record = "none")
two; res2
```

```
## End(Not run)
```

---

use_vcr *Setup vcr for a package*

Description

Setup vcr for a package

Usage

```
use_vcr(dir = ".", verbose = TRUE)
```

Arguments

dir	(character) path to package root. default's to current directory
verbose	(logical) print progress messages. default: TRUE

Value

only messages about progress, returns invisible()

---

vcr *vcr: Record HTTP Calls to Disk*

Description

vcr records test suite 'HTTP' requests and replay them during future runs.

Details

Check out the http testing book for a lot more documentation on `vcr`, `webmockr`, and `crul`

Backstory

A Ruby gem of the same name (`VCR`, `https://github.com/vcr/vcr`) was created many years ago and is the original. Ports in many languages have been done. Check out that GitHub repo for all the details on how the canonical version works.

Main functions

The use_cassette function is most likely what you'll want to use. It sets the cassette you want to record to, inserts the cassette, and then ejects the cassette, recording the interactions to the cassette.

Instead, you can use insert_cassette, but then you have to make sure to use eject_cassette.

vcr configuration

vcr_configure is the function to use to set R session wide settings. See it's manual file for help.

Record modes

See recording for help on record modes.

Request matching

See request-matching for help on the many request matching options.

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

---

vcr_configure *Configuration*

Description

Configuration

Usage

```
vcr_configure(dir = ".", record = "once",
  match_requests_on = c("method", "uri"),
  allow_unused_http_interactions = TRUE, serialize_with = "yaml",
  persist_with = "FileSystem", ignore_hosts = NULL,
  ignore_localhost = FALSE, ignore_request = NULL,
  uri_parser = "crul::url_parse", preserve_exact_body_bytes = FALSE,
  turned_off = FALSE, ignore_cassettes = FALSE,
  re_record_interval = NULL, clean_outdated_http_interactions = NULL,
  allow_http_connections_when_no_cassette = FALSE, cassettes = list(),
  linked_context = NULL, log = FALSE, log_opts = list(file =
  "vcr.log", log_prefix = "Cassette", date = TRUE),
  filter_sensitive_data = NULL)

vcr_configure_reset()

vcr_configuration()

vcr_config_defaults()
```

Arguments

`dir`	Cassette directory
`record`	(character) One of 'all', 'none', 'new_episodes', or 'once'. See recording
`match_requests_on`	vector of matchers. Default: (`method`, `uri`) See request-matching for details.
`allow_unused_http_interactions`	(logical) Default: `TRUE`
`serialize_with`	(character) only option is "yaml"
`persist_with`	(character) only option is "FileSystem"
`ignore_hosts`	(character) Vector of hosts to ignore. e.g., localhost, or google.com. These hosts are ignored and real HTTP requests allowed to go through
`ignore_localhost`	(logical) Default: `FALSE`
`ignore_request`	List of requests to ignore
`uri_parser`	the uri parser, default: `crul::url_parse()`
`preserve_exact_body_bytes`	(logical) preserve exact body bytes for
`turned_off`	(logical) VCR is turned on by default. Default: `FALSE`
`ignore_cassettes`	(logical) Ignore cassettes. You can set this to `TRUE` when you don't have a cassette in use but still want to make HTTP requests. Otherwise, you can't make requests unless a cassette is in use. Default: `FALSE`
`re_record_interval`	(numeric) When given, the cassette will be re-recorded at the given interval, in seconds.
`clean_outdated_http_interactions`	(logical) Should outdated interactions be recorded back to file. Default: `FALSE`
`allow_http_connections_when_no_cassette`	(logical) Determines how vcr treats HTTP requests that are made when no vcr cassette is in use. When `TRUE`, requests made when there is no vcr cassette in use will be allowed. When `FALSE` (default), an UnhandledHTTPRequestError error will be raised for any HTTP request made when there is no cassette in use
`cassettes`	(list) don't use
`linked_context`	(logical) linked context
`log`	(logical) should we log important vcr things? Default: `FALSE`
`log_opts`	(list) Additional logging options. Options include: • file: one of a file path to log to or "console" • log_prefix: default: "Cassette". We insert the cassette name after that prefix, then the rest of the message • more to come

filter_sensitive_data
: (list) named list of values to replace. format is: `list(thing_to_replace = thing_to_replace_it_wit`
We replace all instances of `thing_to_replace` with `thing_to_replace_it_with`.
Before recording (writing to a cassette) we do the replacement and then when reading from the cassette we do the reverse replacement to get back to the real data. Before record replacement happens in internal function `write_interactions()`, while before playback replacement happens in internal function `YAML$deserialize_path()`

Examples

```
vcr_configure(dir = tempdir())
vcr_configure(dir = tempdir(), record = "all")
vcr_configuration()
vcr_config_defaults()
vcr_configure(tempdir(), ignore_localhost = TRUE)

# logging
vcr_configure(tempdir(), log = TRUE,
  log_opts = list(file = file.path(tempdir(), "vcr.log")))
vcr_configure(tempdir(), log = TRUE, log_opts = list(file = "console"))
vcr_configure(tempdir(), log = TRUE,
 log_opts = list(
   file = file.path(tempdir(), "vcr.log"),
   log_prefix = "foobar"
))
vcr_configure(tempdir(), log = FALSE)

# filter sensitive data
vcr_configure(tempdir(),
  filter_sensitive_data = list(foo = "<bar>")
)
vcr_configure(tempdir(),
  filter_sensitive_data = list(foo = "<bar>", hello = "<world>")
)
```

Index