Package 'vcr'

July 23, 2025

Title Record 'HTTP' Calls to Disk

Version 2.0.0

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 recording real 'HTTP' requests/responses on disk in 'cassettes', and then replaying matching responses on subsequent requests.

License MIT + file LICENSE

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

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

Depends R (>= 4.1)

Imports cli, curl (>= 6.3.0), jsonlite, lifecycle, R6, rlang (>= 1.1.0), rprojroot, waldo, yaml

Suggests crul (>= 1.6.0), desc, httr, httr2 (>= 1.1.2), knitr, qs2, rmarkdown, roxygen2 (>= 7.2.1), testthat (>= 3.0.0), webfakes, withr

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first ause_cassette_re_record

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

X-schema.org-applicationCategory Web

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

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

NeedsCompilation no

Author Scott Chamberlain [aut, cre] (ORCID:
<https://orcid.org/0000-0003-1444-9135>),
Aaron Wolen [aut] (ORCID: <https://orcid.org/0000-0003-2542-2202>),
Maëlle Salmon [aut] (ORCID: <https://orcid.org/0000-0002-2815-0399>),
Daniel Possenriede [aut] (ORCID:
<https://orcid.org/0000-0002-6738-9845>),
Hadley Wickham [aut],
rOpenSci [fnd] (ROR: <https://ror.org/019jywm96>)

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2025-07-23 12:24:48 UTC

Contents

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

Description

List cassettes, get current cassette, etc.

Usage

```
cassettes()

current_cassette()

current_cassette_recording()

current_cassette_replaying()

cassette_path()
```

Details

- `cassettes()`: returns all active cassettes in the current session.
- `current_cassette()`: returns `NULL` when no cassettes are in use; returns the current cassette (a `Cassette` object) when one is in use
- `currrent_cassette_recording()` and `current_cassette_replaying()`: tell you if the current cassette is recording and/or replaying. They both return `FALSE` if there is no cassette in use.
- `cassette_path()`: returns the current directory path where cassettes will be stored

Examples

```
vcr_configure(dir = tempdir())

# list all cassettes
cassettes()

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

eject_cassette()
cassettes()


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

vcr_configure_reset()
```

`insert_example_cassette` *Use cassettes in examples*

Description

`insert_example_cassette()` is a wrapper around `insert_cassette()` that stores cassettes in `inst/_vcr/`. Call it in the first line of your examples (typically wrapped in `\dontshow{}`), and call `eject_cassette()` on the last line.

Run the example manually once to record the vignettte, then it will be replayed during `R CMD check`, ensuring that your example no longer uses the internet.

Usage

```
insert_example_cassette(
  name,
  package,
  record = NULL,
  match_requests_on = NULL,
  serialize_with = NULL,
  preserve_exact_body_bytes = NULL,
  re_record_interval = NULL
)
```

Arguments

name	The name of the cassette. This is used to name a file on disk, so it must be valid file name.
package	Package name.
record	Record mode. This will be "once" if package is under development, (i.e. loaded by devtools) and "none" otherwise. This makes it easy to record during development and ensure that cassettes HTTP requests are never made on CRAN. To re-record all cassettes, you can delete inst/_vcr then run pkgdown::build_reference(lazy = FALSE).
match_requests_on	Character vector of request matchers used to determine which recorded HTTP interaction to replay. The default matches on the "method", "uri", and either "body" (if present) or "body_json" (if the content-type is application/json). The full set of possible values are: • method: the HTTP method. • uri: the complete request URI, excluding the port. • uri_with_port: the complete request URI, including the port. • host: the **host** component of the URI. • path: the **path** component of the URI. • query: the **query** component of the URI. • body: the request body. • body_json: the request body, parsed as JSON. • header: all request headers. If more than one is specified, all components must match in order for the request to match. If not supplied, defaults to c("method", "uri"). Note that the request header and body will only be included in the cassette if match_requests_on includes "header" or "body" respectively. This keeps the recorded request as lightweight as possible.
serialize_with	(string) Which serializer to use: "yaml" (the default), "json", or "qs2".
preserve_exact_body_bytes	(logical) Force a binary (base64) representation of the request and response bodies? By default, vcr will look at the Content-Type header to determine if this is necessary, but if it doesn't work you can set preserve_exact_body_bytes = TRUE to force it.

re_record_interval
: (integer) How frequently (in seconds) the cassette should be re-recorded. Default: NULL (not re-recorded).

Examples

```
# In this example I'm showing the insert and eject commands, but you'd
# usually wrap these in \dontshow{} so the user doesn't see them and
# think that they're something they need to copy.

insert_example_cassette("httpbin-get", package = "vcr")

req <- httr2::request("https://hb.cran.dev/get")
resp <- httr2::req_perform(req)

str(httr2::resp_body_json(resp))

eject_cassette()
```

---

is_recording *Determine if vcr is recording/replaying*

Description

local_cassette() and use_cassette() set the VCR_IS_RECORDING and VCR_IS_REPLAYING environment variables to make it easy to determine vcr state without taking a dependency on vcr. These functions show you how to use them; we expect you to copy and paste these functions into your own package

Usage

```
is_recording()

is_replaying()
```

Value

TRUE or FALSE.

lightswitch *Turn vcr on and off*

Description

- `turn_on()` and `turn_off()` turn on and off for the whole session.
- `turned_off(code)` temporarily turns off while `code` is running, guaranteeing that you make a real HTTP request.
- `turned_on()` reports on if vcr is turned on or not.
- `skip_if_vcr_off()` skips a test if vcr is turned off. This is occasionally useful if you're using a cassette to simulate a faked request, or if the real request would return different values (e.g. you're testing date parsing and the request returns the current date).

You can also control the default behaviour in a new session by setting the following environment variables before R starts:

- Use `VCR_TURN_OFF=true` to suppress all vcr usage, ignoring all cassettes. This is useful for CI/CD workflows where you want to ensure the test suite is run against the live API.
- Set `VCR_TURNED_OFF=true` to turn off vcr, but still use cassettes.

Usage

```
turn_on()

turn_off(ignore_cassettes = FALSE)

turned_off(code, ignore_cassettes = FALSE)

turned_on()

skip_if_vcr_off()
```

Arguments

`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. Default: `FALSE`
`code`	Any block of code to run, presumably an HTTP request.

Examples

```
# By default, vcr is turned on
turned_on()

# you can turn off for the rest of the session
turn_off()
```

```
turned_on()
# turn on again
turn_on()

# or just turn it on turn off temporarily
turned_off({
  # some HTTP requests here
  turned_on()
})
```

`setup_knitr` *Use vcr in vignettes*

Description

`setup_knitr()` registers a knitr hook to make it easy to use vcr inside a vignette. First call `setup_knitr()` in your setup chunk (or other chunk early in the document):

````
```{r setup}
#| include: false
vcr::setup_knitr()
```
````

Then, in a chunk where you want to use a cassette, set the `cassette` chunk option to the name of the cassette:

````
```{r}
#| cassette: name
req <- httr2::request("http://r-project.org")
resp <- httr2::req_perform(req)
```
````

Or if you have labelled the chunk, you can use `cassette: true` to use the chunk label as the cassette name:

````
```{r}
#| label: cassette-name
#| cassette: true
req <- httr2::request("http://r-project.org")
resp <- httr2::req_perform(req)
```
````

You can build your vignettes however you usually do (from the command line, with pkgdown, with RStudio/Positron, etc).

Usage

```
setup_knitr(prefix = NULL, dir = "_vcr", ...)
```

Arguments

prefix	An prefix for the cassette name to make cassettes unique across vignettes. Defaults to the file name (without extension) of the currently executing vignette.
dir	Directory where to create the cassette file. Default: '"_vcr"'.
...	Other arguments passed on to insert_cassette().

---

use_cassette *Use a cassette to record HTTP requests*

Description

use_cassette(...) uses a cassette for the code in ...; local_cassette() uses a cassette for the current function scope (e.g. for one test). Learn more in vignette("vcr").

Note that defaults for most arguments are controlled by vcr_configure(), so you may want to use that instead if you are changing the defaults for all cassettes.

Usage

```
use_cassette(
  name,
  ...,
  dir = NULL,
  record = NULL,
  match_requests_on = NULL,
  serialize_with = NULL,
  preserve_exact_body_bytes = NULL,
  re_record_interval = NULL,
  warn_on_empty = NULL
)

local_cassette(
  name,
  dir = NULL,
  record = NULL,
  match_requests_on = NULL,
  serialize_with = NULL,
  preserve_exact_body_bytes = NULL,
  re_record_interval = NULL,
  warn_on_empty = NULL,
  frame = parent.frame()
)
```

Arguments

`name`	The name of the cassette. This is used to name a file on disk, so it must be valid file name.
`...`	a block of code containing one or more requests (required). Use curly braces to encapsulate multi-line code blocks. If you can't pass a code block use `insert_cassette()` instead.
`dir`	The directory where the cassette will be stored. Defaults to `test_path("_vcr")`.
`record`	Record mode that dictates how HTTP requests/responses are recorded. Possible values are: • **once**, the default: Replays recorded interactions, records new ones if no cassette exists, and errors on new requests if cassette exists. • **none**: Replays recorded interactions, and errors on any new requests. Guarantees that no HTTP requests occur. • **new_episodes**: Replays recorded interactions and always records new ones, even if similar interactions exist. • **all**: Never replays recorded interactions, always recording new. Useful for re-recording outdated responses or logging all HTTP requests.
`match_requests_on`	Character vector of request matchers used to determine which recorded HTTP interaction to replay. The default matches on the `"method"`, `"uri"`, and either `"body"` (if present) or `"body_json"` (if the content-type is `application/json`). The full set of possible values are: • `method`: the HTTP method. • `uri`: the complete request URI, excluding the port. • `uri_with_port`: the complete request URI, including the port. • `host`: the **host** component of the URI. • `path`: the **path** component of the URI. • `query`: the **query** component of the URI. • `body`: the request body. • `body_json`: the request body, parsed as JSON. • `header`: all request headers. If more than one is specified, all components must match in order for the request to match. If not supplied, defaults to `c("method", "uri")`. Note that the request header and body will only be included in the cassette if `match_requests_on` includes "header" or "body" respectively. This keeps the recorded request as lightweight as possible.
`serialize_with`	(string) Which serializer to use: `"yaml"` (the default), `"json"`, or `"qs2"`.
`preserve_exact_body_bytes`	(logical) Force a binary (base64) representation of the request and response bodies? By default, vcr will look at the `Content-Type` header to determine if this is necessary, but if it doesn't work you can set `preserve_exact_body_bytes = TRUE` to force it.

re_record_interval
: (integer) How frequently (in seconds) the cassette should be re-recorded. Default: NULL (not re-recorded).

warn_on_empty
: (logical) Warn if the cassette is ejected but no interactions have been recorded. Default: NULL (inherits from global configuration).

frame
: Attach exit handlers to this environment. Typically, this should be either the current environment or a parent frame (accessed through parent.frame()). See vignette("withr", package = "withr") for more details.

See Also

insert_cassette() and eject_cassette() for the underlying functions.

---

vcr_configure *Global Configuration Options*

Description

Configurable options that define vcr's default behavior.

Usage

```
vcr_configure(
  dir,
  record,
  match_requests_on,
  serialize_with,
  json_pretty,
  ignore_hosts,
  ignore_localhost,
  preserve_exact_body_bytes,
  turned_off,
  re_record_interval,
  log,
  log_opts,
  filter_sensitive_data,
  filter_sensitive_data_regex,
  filter_request_headers,
  filter_response_headers,
  filter_query_parameters,
  write_disk_path,
  warn_on_empty_cassette
)

local_vcr_configure(..., .frame = parent.frame())

vcr_configure_reset()
```

```
vcr_configuration()

vcr_config_defaults()
```

Arguments

`dir`	Directory where cassettes are stored.
`record`	Record mode that dictates how HTTP requests/responses are recorded. Possible values are: • **once**, the default: Replays recorded interactions, records new ones if no cassette exists, and errors on new requests if cassette exists. • **none**: Replays recorded interactions, and errors on any new requests. Guarantees that no HTTP requests occur. • **new_episodes**: Replays recorded interactions and always records new ones, even if similar interactions exist. • **all**: Never replays recorded interactions, always recording new. Useful for re-recording outdated responses or logging all HTTP requests.
`match_requests_on`	Character vector of request matchers used to determine which recorded HTTP interaction to replay. The default matches on the `"method"`, `"uri"`, and either `"body"` (if present) or `"body_json"` (if the content-type is `application/json`). The full set of possible values are: • `method`: the HTTP method. • `uri`: the complete request URI, excluding the port. • `uri_with_port`: the complete request URI, including the port. • `host`: the **host** component of the URI. • `path`: the **path** component of the URI. • `query`: the **query** component of the URI. • `body`: the request body. • `body_json`: the request body, parsed as JSON. • `header`: all request headers. If more than one is specified, all components must match in order for the request to match. If not supplied, defaults to `c("method", "uri")`. Note that the request header and body will only be included in the cassette if `match_requests_on` includes "header" or "body" respectively. This keeps the recorded request as lightweight as possible.
`serialize_with`	(string) Which serializer to use: `"yaml"` (the default), `"json"`, or `"qs2"`.
`json_pretty`	(logical) want JSON to be newline separated to be easier to read? Or remove newlines to save disk space? default: `FALSE`.
`ignore_hosts`	(character) Vector of hosts to ignore. e.g., `"localhost"`, or `"google.com"`. These hosts are ignored and real HTTP requests are allowed to go through.
`ignore_localhost`	(logical) Default: `FALSE`

`preserve_exact_body_bytes`
: (logical) Force a binary (base64) representation of the request and response bodies? By default, vcr will look at the `Content-Type` header to determine if this is necessary, but if it doesn't work you can set `preserve_exact_body_bytes = TRUE` to force it.

`turned_off`
: (logical) VCR is turned on by default. Default: `FALSE`.

`re_record_interval`
: (integer) How frequently (in seconds) the cassette should be re-recorded. Default: `NULL` (not re-recorded).

`log`, `log_opts`
: See `vcr_configure_log()`.

`filter_sensitive_data`, `filter_sensitive_data_regex`
: Transform header and/or body in the request and response. Named list of substitutions to apply to the headers and body of the request and response. Format is `list(replacement = "original")` where `replacement` is a string that is matched exactly for `filter_sensitive_data` and a regular expression for `filter_sensitive_data_regex`.

`filter_request_headers`, `filter_response_headers`
: Filter request or response headers. Should be a list: unnamed components are removed, and named components are transformed. For example, `list("api_key")` would remove the `api_key` header and `list(api_key = "***")` would replace the `api_key` header with `***`.

  httr2's redacted headers are automatically removed.

`filter_query_parameters`
: Filter query parameters in the request. A list where unnamed components are removed, and named components are transformed. For example, `list("api_key")` would remove the `api_key` parameter and `list(api_key = "***")` would replace the `api_key` parameter with `***`.

`write_disk_path`
: (character) path to write files to for any requests that write responses to disk. By default this will be `{cassette-name}-files/` inside the cassette directory.

`warn_on_empty_cassette`
: (logical) Should a warning be thrown when an empty cassette is detected? Empty cassettes are cleaned up (deleted) either way. This option only determines whether a warning is thrown or not. Default: `TRUE`

`...`
: Configuration settings used to override defaults.

`.frame`
: Attach exit handlers to this environment. Typically, this should be either the current environment or a parent frame (accessed through `parent.frame()`). See `vignette("withr", package = "withr")` for more details.

Examples

```
vcr_configure(dir = tempdir())
vcr_configure(dir = tempdir(), record = "all")
vcr_configuration()
vcr_config_defaults()
vcr_configure(dir = tempdir(), ignore_hosts = "google.com")
vcr_configure(dir = tempdir(), ignore_localhost = TRUE)
```

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

`vcr_configure_log` *Configure vcr logging*

Description

By default, logging is disabled, but you can easily enable for the entire session with `vcr_configure_log()` or for just one test with `local_vcr_configure_log()`.

Usage

```
vcr_configure_log(
  log = TRUE,
  file = stderr(),
  include_date = NULL,
  log_prefix = "Cassette"
)

local_vcr_configure_log(
  log = TRUE,
  file = stderr(),
  include_date = NULL,
  log_prefix = "Cassette",
  frame = parent.frame()
)
```

Arguments

`log`	Should we log important vcr things?
`file`	A path or connection to log to
`include_date`	(boolean) Include date and time in each log entry.
`log_prefix`	"Cassette". We insert the cassette name after this prefix, followed by the rest of the message.
`frame`	Attach exit handlers to this environment. Typically, this should be either the current environment or a parent frame (accessed through `parent.frame()`). See `vignette("withr", package = "withr")` for more details.

Examples

```
# The default logs to stderr()
vcr_configure_log()

# But you might want to log to a file
vcr_configure_log(file = file.path(tempdir(), "vcr.log"))
```

`vcr_last_request` *Retrieve last vcr request/response*

Description

When debugging, it's often useful to see the last request and response respectively. These functions give you what would have been recorded to disk or replayed from disk. If the last request wasn't recorded, and there was no matching request, `vcr_last_response` will return `NULL`.

Usage

```
vcr_last_request()

vcr_last_response()
```

`vcr_test_path` *Locate file in tests directory*

Description

This function, similar to `testthat::test_path()`, is designed to work both interactively and during tests, locating files in the `tests/` directory.

Usage

```
vcr_test_path(...)
```

Arguments

`...`	Character vectors giving path components. Each character string gets added to the path, e.g., `vcr_test_path("a", "b")` becomes `tests/a/b` relative to the root of the package.

Value

A character vector giving the path

Note

`vcr_test_path()` assumes you are using testthat for your unit tests.

Examples

```
if (interactive()) {
vcr_test_path("fixtures")
}
```

Index