Package 'mockery'

July 23, 2025

Title Mocking Library for R

Version 0.4.4

Description The two main functionalities of this package are creating mock objects (functions) and selectively intercepting calls to a given function that originate in some other function. It can be used with any testing framework available for R. Mock objects can be injected with either this package's own stub() function or a similar with_mock() facility present in the 'testthat' package.

License MIT + file LICENSE

URL https://github.com/r-lib/mockery

BugReports https://github.com/r-lib/mockery/issues

Imports testthat

Suggests knitr, R6, rmarkdown (>= 1.0)

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.3

Collate 'expectations.R' 'mockery.R' 'mock-object.R' 'stub.R'

NeedsCompilation no

Author Noam Finkelstein [aut],
Lukasz Bartnik [aut],
Jim Hester [aut],
Hadley Wickham [aut, cre]

Maintainer Hadley Wickham <hadley@rstudio.com>

Repository CRAN

Date/Publication 2023-09-26 18:50:02 UTC

Contents

`call-expectations` *Expectation: does the given call match the expected?*

Description

Together with `mock` can be used to verify whether the call expression (`expect_call`) and/or argument values (`expect_args`) match the expected.

Usage

```
expect_call(mock_object, n, expected_call)

expect_args(mock_object, n, ...)

expect_called(mock_object, n)
```

Arguments

`mock_object`	A `mock` object.
`n`	Call number or total number of calls.
`expected_call`	Expected call expression; will be compared unevaluated.
`...`	Arguments as passed in a call.

Details

With `expect_called` you can check how many times has the mock object been called.

Examples

```
library(testthat)

# expect call expression (signature)
m <- mock()
with_mock(summary = m, summary(iris))

# it has been called once
expect_called(m, 1)

# the first (and only) call's arguments matches summary(iris)
expect_call(m, 1, summary(iris))

# expect argument value
m <- mock()
a <- iris
with_mock(summary = m, summary(object = a))
expect_args(m, 1, object = a)
# is an equivalent to ...
```

```
expect_equal(mock_args(m)[[1]], list(object = a))
```

---

mock *Create and query a mocked function.*

Description

Mock object's primary use is to record calls that are made on the mocked function.

Usage

```
mock(..., cycle = FALSE, envir = parent.frame())

mock_args(m)

mock_calls(m)

## S3 method for class 'mock'
length(x)
```

Arguments

`...`	Values returned upon subsequent calls.
`cycle`	Whether to cycle over the return values. If `FALSE`, will fail if called too many times.
`envir`	Where to evaluate the expressions being returned.
`m`	A `mock`ed function.
`x`	A `mock`ed function.

Details

Optionally values/expressions can be passed via `...` for the mock object to return them upon subsequent calls. Expressions are evaluated in environment `envir` before being returned. If no value is passed in `...` then `NULL` is returned.

Passing an expression or a function call via `...` is also a way to implement side effects: keep track of the state of code under testing, throw an exception when a condition is met, etc.

`mock_calls` and `mock_args` can be used to access the list of calls made on a mocked function and a respective list of values of arguments passed to each of these calls.

Value

`mock()` returns a mocked function which can be then used with `with_mock`.

`mock_args()` returns a `list` of `lists` of argument values.

`mock_calls()` returns a `list` of `calls`.

`length.mock()` returns the number of calls invoked on `m`.

Examples

```
library(testthat)

m <- mock(1)
with_mock(summary = m, {
  expect_equal(summary(iris), 1)
  expect_called(m, 1)
  expect_call(m, 1, summary(iris))
  expect_args(m, 1, iris)
})

# multiple return values
m <- mock(1, "a", sqrt(3))
with_mock(summary = m, {
  expect_equal(summary(iris), 1)
  expect_equal(summary(iris), "a")
  expect_equal(summary(iris), 1.73, tolerance = .01)
})

# side effects
m <- mock(1, 2, stop("error"))
with_mock(summary = m, {
  expect_equal(summary(iris), 1)
  expect_equal(summary(iris), 2)
  expect_error(summary(iris), "error")
})

# accessing call expressions
m <- mock()
m(x = 1)
m(y = 2)
expect_equal(length(m), 2)
calls <- mock_calls(m)
expect_equal(calls[[1]], quote(m(x = 1)))
expect_equal(calls[[2]], quote(m(y = 2)))

# accessing values of arguments
m <- mock()
m(x = 1)
m(y = 2)
expect_equal(length(m), 2)
args <- mock_args(m)
expect_equal(args[[1]], list(x = 1))
expect_equal(args[[2]], list(y = 2))
```

---

mockery *R package to make mocking easier*

Description

There are great tools for unit testing in R out there already but they don't come with a lot of support for mock objects. This package aims at fixing that.

Examples

```
library(mockery)

m <- mock(TRUE, FALSE, TRUE)

# this will make summary call our mock function rather then
# UseMethod; thus, summary() will return values as above
stub(summary, 'UseMethod', m)

summary(iris) # returns TRUE
summary(cars) # returns FALSE
summary(co2)  # returns TRUE

## Not run:
library(testthat)

m <- mock(TRUE)
f <- function() read.csv('data.csv')

with_mock(read.csv = m, {
  f()
  expect_call(m, 1, read.csv('data.csv'))
})

## End(Not run)
```

`stub` *Replace a function with a stub.*

Description

The result of calling `stub` is that, when `where` is invoked and when it internally makes a call to `what`, `how` is going to be called instead.

Usage

```
stub(where, what, how, depth = 1)
```

Arguments

`where`	Function to be called that will in turn call `what`.
`what`	Name of the function you want to stub out (a `character` string).
`how`	Replacement function (also a `mock` function) or a return value for which a function will be created automatically.
`depth`	Specifies the depth to which the function should be stubbed

Details

This is much more limited in scope in comparison to with_mock which effectively replaces what everywhere. In other words, when using with_mock and regardless of the number of intermediate calls, how is always called instead of what. However, using this API, the replacement takes place only for a single function where and only for calls originating in that function.

Examples

```
f <- function() TRUE
g <- function() f()
stub(g, 'f', FALSE)

# now g() returns FALSE because f() has been stubbed out
g()

# you can stub multiple functions by calling stub() multiple times
f <- function() TRUE
g <- function() TRUE
h <- function() any(f(), g())
stub(h, 'f', FALSE)
stub(h, 'g', FALSE)

# now h() returns FALSE because both f() and g() have been stubbed out
h()
```

Index