Package 'patrick'

January 27, 2025

Title Parameterized Unit Testing

Version 0.3.0

Description This is an extension of the 'testthat' package that lets you add parameters to your unit tests. Parameterized unit tests are often easier to read and more reliable, since they follow the DNRY (do not repeat yourself) rule.

License Apache License 2.0

URL https://github.com/google/patrick

BugReports https://github.com/google/patrick/issues

Depends R (>= 3.1)

Imports dplyr, glue, purrr, rlang, testthat (>= 3.1.5), tibble

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Repository CRAN

Date/Publication 2025-01-27 18:40:05 UTC

Author Michael Quinn [aut, cre], Michael Chirico [ctb]

Maintainer Michael Quinn <msquinn@google.com>

Contents

`with_parameters_test_that` *Execute a test with parameters.*

Description

This function is an extension of [testthat::test_that()] that lets you pass a series of testing parameters. These values are substituted into your regular testing code block, making it reusable and reducing duplication.

Usage

```
with_parameters_test_that(
  desc_stub,
  code,
  ...,
  .cases = NULL,
  .test_name = NULL,
  .interpret_glue = TRUE
)

cases(...)
```

Arguments

`desc_stub`	A string scalar. Used in creating the names of the parameterized tests.
`code`	Test code containing expectations.
`...`	Named arguments of test parameters. All vectors should have the same length.
`.cases`	A data frame where each row contains test parameters.
`.test_name`	An alternative way for providing test names. If provided, the name will be appended to the stub description in 'desc_stub'. If not provided, test names will be automatically generated.
`.interpret_glue`	Logical, default 'TRUE'. If 'FALSE', and glue-like markup in 'desc_stub' is ignored, otherwise [glue::glue_data()] is attempted to produce a more complete test description.

Details

You have a couple of options for passing parameters to you test. You can use named vectors/ lists. The function will assert that you have correct lengths before proceeding to test execution. Alternatively you can used a 'data.frame' or list in combination with the splice unquote operator `!!!`. Last, you can use the constructor 'cases()', which is similar to building a 'data.frame' rowwise. If you manually build the data frame, pass it in the '.cases' argument.

Naming test cases

If the user passes a character vector as '.test_name', each instance is combined with 'desc_stub' to create the completed test name. Similarly, the named argument from 'cases()' is combined with 'desc_stub' to create the parameterized test names. When names aren't provided, they will be automatically generated using the test data.

Names follow the pattern of "name=value, name=value" for all elements in a test case.

Examples

```
with_parameters_test_that("trigonometric functions match identities:",
  {
    testthat::expect_equal(expr, numeric_value)
  },
  expr = c(sin(pi / 4), cos(pi / 4), tan(pi / 4)),
  numeric_value = c(1 / sqrt(2), 1 / sqrt(2), 1),
  .test_name = c("sin", "cos", "tan")
)

# Run the same test with the cases() constructor
with_parameters_test_that(
  "trigonometric functions match identities",
  {
    testthat::expect_equal(expr, numeric_value)
  },
  cases(
    sin = list(expr = sin(pi / 4), numeric_value = 1 / sqrt(2)),
    cos = list(expr = cos(pi / 4), numeric_value = 1 / sqrt(2)),
    tan = list(expr = tan(pi / 4), numeric_value = 1)
  )
)

# If names aren't provided, they are automatically generated.
with_parameters_test_that(
  "trigonometric functions match identities",
  {
    testthat::expect_equal(expr, numeric_value)
  },
  cases(
    list(expr = sin(pi / 4), numeric_value = 1 / sqrt(2)),
    list(expr = cos(pi / 4), numeric_value = 1 / sqrt(2)),
    list(expr = tan(pi / 4), numeric_value = 1)
  )
)
# The first test case is named "expr=0.7071068, numeric_value="0.7071068"
# and so on.

# Or, pass a data frame of cases, perhaps using a helper function
make_cases <- function() {
  tibble::tribble(
    ~.test_name, ~expr, ~numeric_value,
    "sin", sin(pi / 4), 1 / sqrt(2),
    "cos", cos(pi / 4), 1 / sqrt(2),
    "tan", tan(pi / 4), 1
```

```
  )
}

with_parameters_test_that(
  "trigonometric functions match identities",
  {
    testthat::expect_equal(expr, numeric_value)
  },
  .cases = make_cases()
)
```

Index