Package 'realtest'

July 23, 2025

Type Package

Version 0.2.3

Date 2023-05-15

Title Where Expectations Meet Reality: Realistic Unit Testing

Description A framework for unit testing for realistic minimalists, where we distinguish between expected, acceptable, current, fallback, ideal, or regressive behaviour. It can also be used for monitoring third-party software projects for changes.

Depends R (>= 4.0)

Imports utils

BugReports https://github.com/gagolews/realtest/issues

URL https://realtest.gagolewski.com, https://github.com/gagolews/realtest

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Marek Gagolewski [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-0637-6028>)

Maintainer Marek Gagolewski <marek@gagolewski.com>

Repository CRAN

Date/Publication 2023-05-16 11:30:06 UTC

Contents

`about_realtest` *Where Expectations Meet Reality: Realistic Unit Testing in R*

Description

realtest is a framework for unit testing for realistic minimalists, where we distinguish between expected, acceptable, and undesirable behaviour.

Keywords: unit testing, software quality, expectation, undesired behaviour, continuous integration.

License: GNU General Public License version 2 or later.

Author(s)

Marek Gagolewski

See Also

The official online manual of **realtest** at `https://realtest.gagolewski.com/`

`E` *Test Which Expectations are Met*

Description

Performs a unit test and summarises the results.

Usage

```
E(
  expr,
  ...,
  value_comparer = getOption("realtest_value_comparer", identical),
  sides_comparer = getOption("realtest_sides_comparer", sides_similar),
  postprocessor = getOption("realtest_postprocessor", failstop)
)
```

Arguments

`expr`	an expression to be recorded (via `R`) and and compared with the prototypes
`...`	a sequence of 1 or more (possibly named) prototypes constructed via `R` or `P` (objects which are not of class `realtest_descriptor` will be passed to `P`); arguments whose names start with a dot (like `.label=value`) can be used to introduce metadata (e.g., additional details in natural language)
`value_comparer`	a two-argument function used (unless overridden by the prototype) to compare the values with each other, e.g., `identical` or `all.equal`
`sides_comparer`	a two-argument function used (unless overridden by the prototype) to compare the side effects (essentially: two lists) with each other, e.g., `sides_similar` or `ignore_differences`
`postprocessor`	a function to call on the generated `realtest_result`, e.g., `failstop`

Details

Each expression in the R language has a range of possible effects. The direct effect corresponds to the value generated by evaluating the expression. Side effects may include errors, warnings, text printed out on the console, etc., see `P` and `R`.

Arguments passed via `...` whose names do not start with a dot should be objects of class `realtest_descriptor` (otherwise they are passed to `P`). They define the prototypes against which the object generated by `expr` will be tested.

`value_comparer` and `sides_comparer` are 2-ary functions that return `TRUE` if two objects/side effect lists are equivalent and a character string summarising the differences (or any other kind or object) otherwise.

A test case is considered met, whenever `value_comparer(prototype[["value"]], object[["value"]])` and `sides_comparer(prototype[["sides"]], object[["sides"]])` are both `TRUE` for some `prototype`. The comparers may be overridden on a per-prototype basis, though. If `prototype[["value_comparer"]]` or `prototype[["sides_comparer"]]` are defined, these are used instead.

Value

The function creates an object of class `realtest_result`, which is a named list with at least the following components:

- `object` – an object of class `realtest_descriptor`, ultimately `R(expr)`,
- `prototypes` – a (possibly named) list of objects of class `realtest_descriptor` that were passed via `...`,
- `matches` – a (possibly empty) numeric vector of the indexes of the prototypes matching the object (can be named),
- `.dotted.names` – copied as-is from the arguments of the same name.

This object is then passed to the `postprocessor` which itself becomes responsible for generating the output value to be returned by the current function (and, e.g., throwing an error if the test fails).

Author(s)

Marek Gagolewski

See Also

The official online manual of **realtest** at https://realtest.gagolewski.com/

Related functions: P, R, test_dir

Examples

```
# the default result postprocessor throws an error on a failed test:
E(E(sqrt(4), P(7)), P(error=TRUE, stdout=TRUE))
E(sqrt(4), 2.0)  # the same as E(sqrt(4), P(2.0))
E(sin(pi), 0.0, value_comparer=all.equal)  # almost-equal
E(
  sample(c("head", "tail"), 1),
  .description="this call has two possible outcomes",
  "head",  # first prototype
  "tail"   # second prototype
)
E(sqrt(-1), P(NaN, warning=TRUE))  # a warning is expected
E(sqrt(-1), NaN, sides_comparer=ignore_differences) # do not test side effects
E(sqrt(-1), P(NaN, warning=NA))  # ignore warnings
E(
  paste0(1:2, 1:3),                  # expression to test - concatenation
  .description="partial recycling",  # info - what behaviour are we testing?
  best=P(                            # what we yearn for (ideally)
    c("11", "22", "13"),
    warning=TRUE
  ),
  pass=c("11", "22", "13"),          # this is the behaviour we have now
  bad=P(error=TRUE)                  # avoid regression
)
e <- E(sin(pi), best=0.0, pass=P(0.0, value_comparer=all.equal),
  .comment="well, this is not a symbolic language after all...")
print(e)
```

`failstop` *Example Test Result Postprocessors*

Description

Generally, test result postprocessors are used by the E function. `failstop` calls `str(r)` and throws an error if an expectation is not met, i.e., when `r[["matches"]]` is of length 0.

Usage

```
failstop(r)
```

Arguments

`r`	object of class `realtest_result`, see E

Details

These are example postprocessors. You are encouraged to write your own ones that will suit your own needs. Explore their source code for some inspirations. It's an open source (and free!) project after all.

For `failstop`, you can always create a function `str.realtest_result` implementing the pretty printing of an error message.

Value

Returns `r`, invisibly.

Author(s)

Marek Gagolewski

See Also

The official online manual of **realtest** at `https://realtest.gagolewski.com/`

`ignore_differences` *Example Object and Side Effect Comparers*

Description

Example two-argument functions to compare direct or indirect effects of two test descriptors (see `P` and `R`). These can be passed as `value_comparer` and `sides_comparer` to `E`.

Usage

```
ignore_differences(x, y)

sides_similar(x, y)
```

Arguments

`x`	prototype or part thereof
`y`	object under scrutiny or part thereof

Details

Notable built-in (base R) comparers include `identical` (the strictest possible) and `all.equal` (can ignore, amongst others, round-off errors; note that it is an S3 generic).

`ignore_differences` is a dummy comparer that always returns `TRUE`. Hence, it does not discriminate between anything.

`sides_similar` is useful when comparing side effect lists. It defines the following semantics for the prototypical values:

- non-existent, NULL, or FALSE – a side effect must not occur,
- NA – ignore whatsoever,
- TRUE – a side effect occurs, but the details are irrelevant (e.g., 'some warning' as opposed to "NaNs produced")
- otherwise – a character vector with message(s) matched exactly.

You can define any comparers of your own liking: the possibilities are endless. For example:

- a comparer for side effects based on regular expressions or wildcards (e.g., ".not converged.*"),
- a comparer that tests whether all elements in a vector are equal to TRUE,
- a comparer that verifies whether each element in a vector falls into a specified interval,
- a comparer that ignores all the object attributes (possibly in combination with other comparers),

and so forth.

Value

Each comparer should yield TRUE if the test condition is considered met or anything else otherwise. However, it is highly recommended that in the latter case, a single string with a short summary of the differences be returned, as in all.equal.

Author(s)

Marek Gagolewski

See Also

The official online manual of **realtest** at https://realtest.gagolewski.com/

P *Manually Create a Test Descriptor Prototype*

Description

Allows for formulating expectations like 'the desired outcome is c(1, 2, 3), with a warning' or 'an error should occur'.

Usage

```
P(
  value = NULL,
  error = NULL,
  warning = NULL,
  message = NULL,
  stdout = NULL,
  stderr = NULL,
  value_comparer = NULL,
  sides_comparer = NULL
)
```

Arguments

`value`	object (may of course be equipped with attributes)
`error`, `warning`, `message`	conditions expected to occur, see `stop`, `warning`, and `message`
`stdout`, `stderr`	character data expected on `stdout` and `stderr`, respectively
`value_comparer`, `sides_comparer`	optional two-argument functions which may be used to override the default comparers used by `E`

Details

If `error`, `warning`, `message`, `stdout`, or `stderr` are `NULL`, then no side effects of particular kinds are included in the output.

The semantics is solely defined by the `sides_comparer`. `E` by default uses `sides_similar` (see its description therein), although you are free to override it manually or via a global option.

Value

A list of class `realtest_descriptor` with named components:

- `value`,
- `sides` (optional) – a list with named elements `error`, `warnings`, `messages`, `stdout`, and `stderr`; those which are missing are assumed to be equal to `NULL`,
- `value_comparer` (optional) – a function object,
- `sides_comparer` (optional) – a function object.

Other functions are free to add more named components, and do with them whatever they please.

Author(s)

Marek Gagolewski

See Also

The official online manual of **realtest** at `https://realtest.gagolewski.com/`

Related functions: `E`, `R`

Examples

```
# the desired outcome is c(1L, 2L, 3L):
P(1:3)
# expecting c(1L, 2L, 3L), with a warning:
P(1:3, warning=TRUE)
# note, however, that it is the sides_comparer that defines the semantics
```

`print.realtest_results_summary` *Summarise and Display Test Results*

Description

An example (write your own which will better suit your needs) way to summarise the results returned by `test_dir`.

Usage

```
## S3 method for class 'realtest_results_summary'
print(x, label_fail = "fail", ...)

## S3 method for class 'realtest_results'
summary(object, label_pass = "pass", label_fail = "fail", ...)
```

Arguments

`x`	object returned by `summary.realtest_results`
`label_fail`	single string labelling failed test cases
`...`	currently ignored
`object`	list of objects of class `realtest_result`, see `E`.
`label_pass`	single string denoting the default name for unnamed prototypes

Value

`print.realtest_results_summary` returns `x`, invisibly.

`summary.realtest_results` returns an object of class `realtest_results_summary` which is a data frame summarising the test results, featuring the following columns:

- `call` – the name of the function tested,
- `match` – the name of the first matching prototype, `label_pass` if it is unnamed or `label_fail` if there is no match,
- `.file` (optional) – the name of the source file which defined the expectation,
- `.line` (optional) – line number,
- `.expr` (optional) – source code of the whole tested expression.

Author(s)

Marek Gagolewski

See Also

The official online manual of **realtest** at `https://realtest.gagolewski.com/`

Related functions: `test_dir`

Examples

```
# r <- test_dir("~/R/realtest/inst/realtest")  # some path
# s <- summary(r)  # summary.realtest_results
# print(s)  # print.realtest_results_summary
# stopifnot(!any(s[["match"]]=="fail"))  # halt if there are failed tests
```

R *Create a Result Descriptor by Recording Effects of an Expression Evaluation*

Description

Evaluates an expression and records its direct and indirect effects: the resulting value as well as the information whether any errors, warnings, or messages are generated and if anything is printed on `stdout` or `stderr`.

Usage

```
R(expr, ..., envir = parent.frame())
```

Arguments

`expr`	expression to be evaluated
`...`	further arguments to be passed to `P`
`envir`	environment where `expr` is to be evaluated

Details

Note that messages, warnings, and errors are typically written to `stderr`, but these are considered separately here. In other words, when testing expectations with `E`, e.g., the reference `stderr` should not include the anticipated diagnostic messages.

There may be other side effects, such as changing the state of the random number generator, modifying options or environment variables, modifying the calling or global environment (e.g., creating new global variables), attaching objects onto the search part (e.g., loading package namespaces), or plotting, but these will not be captured, at least, not by the current version of the **realtest** package.

Value

A list of class `realtest_descriptor`, see `P`, which this function calls. The additional named component `expr` gives the expression that generated the `value`. Moreover, `args` gives a named list of objects that appeared in `expr` (not including functions called).

If an effect of particular kind does not occur, it is not included in the resulting list. `stdout`, `stderr`, and `error` are at most single strings.

When an error occurs, `value` is `NULL`.

Author(s)

Marek Gagolewski

See Also

The official online manual of **realtest** at https://realtest.gagolewski.com/

Related functions: E, P

Examples

```
y <- 1:10; R(sum(y^2))
R(cat("a bit talkative, innit?"))
R(sqrt(c(-1, 0, 1, 2, 4)))
R(log("aaaargh"))
R({
    cat("STDOUT"); cat("STDERR", file=stderr()); message("MESSAGE");
    warning("WARNING"); warning("WARNING AGAIN"); cat("MORE STDOUT");
    message("ANOTHER MESSAGE"); stop("ERROR"); y; "NO RETURN VALUE"
})
```

---

source2 *Read and Evaluate Code from an R Script*

Description

A simplified alternative to source, which additionally sets some environment variables whilst executing a series of expressions to ease debugging.

Usage

```
source2(file, local = FALSE)
```

Arguments

file	usually a file name, see parse
local	specifies the environment where expressions will be evaluated, see source

Details

The function sets/updates the following environment variables while evaluating consecutive expressions:

- __FILE__ – path to the current file,
- __LINE__ – line number where the currently executed expression begins,
- __EXPR__ – source code defining the expression.

Value

This function returns nothing.

Author(s)

Marek Gagolewski

See Also

The official online manual of **realtest** at https://realtest.gagolewski.com/

Examples

```
# example error handler - report source file and line number
old_option_error <- getOption("error")
options(error=function()
   cat(sprintf(
       "Error in %s:%s.\n", Sys.getenv("__FILE__"), Sys.getenv("__LINE__")
   ), file=stderr()))
# now call source2() to execute an R script that throws some errors...
options(error=old_option_error)  # cleanup
```

---

test_dir *Gather All Test Results From R Scripts*

Description

Executes all R scripts in a given directory whose names match a given pattern and gathers all test result in a single list, which you can process however you desire.

The function does not fail if some tests are not met – you need to detect this yourself.

Usage

```
test_dir(
  path = "tests",
  pattern = "^realtest-.*\\.R$",
  recursive = FALSE,
  ignore.case = FALSE
)
```

Arguments

path	directory with scripts to execute
pattern	regular expression specifying the file names to execute
recursive	logical, see list.files
ignore.case	logical, see list.files

Value

Returns a list of all test results (of class `realtest_results`), each being an object of class `realtest_result`, see `E`, with additional fields `.file`, `.line`, and `.expr`, giving the location and the source code of the test instance.

Author(s)

Marek Gagolewski

See Also

The official online manual of **realtest** at https://realtest.gagolewski.com/

Related functions: `source2`, `summary.realtest_results`

Examples

```
# r <- test_dir("~/R/realtest/inst/realtest")  # some path
# s <- summary(r)  # summary.realtest_results
# print(s)  # print.realtest_results_summary
# stopifnot(!any(s[["match"]]=="fail"))  # halt if there are failed tests
```

Index