Package ‘xpectr’

June 18, 2020

Title Generates Expectations for 'testthat' Unit Testing

Version 0.4.0

Description Helps systematize and ease the process of building unit tests with the 'testthat' package by providing tools for generating expectations.

License MIT + file LICENSE

URL https://github.com/ludvigolsen/xpectr

BugReports https://github.com/ludvigolsen/xpectr/issues

Depends R (>= 3.5.0)

Imports clipr (>= 0.7.0), checkmate (>= 2.0.0), dplyr, fansi (>= 0.4.1), lifecycle, plyr, rlang, rstudioapi (>= 0.10), stats, testthat (>= 2.3.1), tibble, utils, withr (>= 2.0.0)

Suggests data.table, knitr, rmarkdown

RdMacros lifecycle

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.0

VignetteBuilder knitr
**assertCollectionAddin**

**Topics documented:**

assertCollectionAddin ........................................... 2  
capture_parse_eval_side_effects ........................... 3  
capture_side_effects ........................................ 4  
dputSelectedAddin ............................................. 6  
element_classes ................................................. 7  
element_lengths ................................................ 8  
element_types .................................................. 9  
gxs_function ..................................................... 10  
gxs_selection ................................................... 13  
initializeGXSFunctionAddin .................................... 16  
initializeTestthatAddin ....................................... 17  
insertExpectationsAddin ....................................... 18  
navigateTestFileAddin ......................................... 20  
num_total_elements ......................................... 21  
prepare_insertion ............................................... 22  
set_test_seed .................................................. 23  
simplified_formals ........................................... 23  
smpl .............................................................. 24  
stop_if .......................................................... 25  
strip ............................................................. 26  
strip_msg ........................................................ 27  
suppress_mw ..................................................... 29  
wrapStringAddin ................................................. 30  
xpectr .......................................................... 31

Index 32

assertCollectionAddin  *Inserts code for a checkmate assert collection*

**Description**

**Experimental**

RStudio Addin: Inserts code for initializing and reporting a checkmate assert collection. See `Details` for how to set a key command.

**Usage**

`assertCollectionAddin(add_comments = TRUE, insert = TRUE, indentation = NULL)`

**Arguments**

- **add_comments**: Whether to add comments around. (Logical)  
  This makes it easy for a user to create their own addin without the comments.
- **insert**: Whether to insert the code via `rstudioapi::insertText()` or return it. (Logical)  
  **N.B.** Mainly intended for testing the addin programmatically.
- **indentation**: Indentation of the code. (Numeric)  
  **N.B.** Mainly intended for testing the addin programmatically.
**Details**

**How to set up a key command in RStudio:**

After installing the package. Go to:
- Tools >> Addins >> Browse Addins >> Keyboard Shortcuts.
- Find "Insert checkmate AssertCollection Code" and press its field under Shortcut.
- Press desired key command, e.g. Alt+C.

Press Apply.
Press Execute.

**Value**

Inserts the following (excluding the ----):

```r
# Check arguments ####
assert_collection <- checkmate::makeAssertCollection()
# checkmate::assert_ ,add = assert_collection)
checkmate::reportAssertions(assert_collection)
# End of argument checks ####
```

Returns NULL invisibly.

**Author(s)**

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

**See Also**

Other addins: `dputSelectedAddin()`, `initializeGXSFunctionAddin()`, `initializeTestthatAddin()`, `insertExpectationsAddin()`, `navigateTestFileAddin()`, `wrapStringAddin()`

---

**Description**

Wraps string in `capture_side_effects()` before parsing and evaluating it. The side effects (error, warnings, messages) are returned in a list.
When capturing an error, no other side effects are captured.

**Usage**

```r
capture_parse_eval_side_effects(
  string,
  envir = NULL,
  copy_env = FALSE,
  reset_seed = FALSE,
  disable_crayon = TRUE
)
```
**Arguments**

- **string**: String of code that can be parsed and evaluated in `envir`.
- **envir**: Environment to evaluate in. Defaults to `parent.frame()`.
- **copy_env**: Whether to use deep copies of the environment when capturing side effects. (Logical)
  Disabled by default to save memory but is often preferable to enable, e.g. when the function alters non-local variables before throwing its error/warning/message.
- **reset_seed**: Whether to reset the random state on exit. (Logical)
- **disable_crayon**: Whether to disable `crayon` formatting. This can remove ANSI characters from the messages. (Logical)

**Value**

- named list with the side effects.

**Author(s)**

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

**See Also**

Other capturers: `capture_side_effects()`

**Examples**

```
# Attach package
library(xpectr)

capture_parse_eval_side_effects("stop('hi!')")
capture_parse_eval_side_effects("warning('hi!')")
capture_parse_eval_side_effects("message('hi!')")
```

**Description**

Captures errors, warnings, and messages from an expression.
In case of an error, no other side effects are captured.
Simple wrapper for testthat's `capture_error()`, `capture_warnings()` and `capture_messages()`.
Note: Evaluates `expr` up to three times.
Usage

capture_side_effects(
    expr,
    envir = NULL,
    copy_env = FALSE,
    reset_seed = FALSE,
    disable_crayon = TRUE
)

Arguments

expr  Expression.
envir Environment to evaluate in. Defaults to parent.frame().
copy_env Whether to use deep copies of the environment when capturing side effects. (Logical)
            Disabled by default to save memory but is often preferable to enable, e.g. when the function alters non-local variables before throwing its error/warning/message.
reset_seed Whether to reset the random state on exit. (Logical)
disable_crayon Whether to disable crayon formatting. This can remove ANSI characters from the messages. (Logical)

Value

named list with the side effects.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also

Other capturers: capture_parse_eval_side_effects()

Examples

# Attach packages
library(xpectr)

fn <- function(raise = FALSE){
    message("Hi! I'm Kevin, your favorite message!"")
    warning("G'Day Mam! I'm a warning to the world!")
    message("Kevin is ma name! Yesss!"")
    warning("Hopefully the whole world will see me :o")
    if (isTRUE(raise)){
        stop("Lord Evil Error has arrived! Yeehaaa")
    }
    "the output"
}

capture_side_effects(fn())
capture_side_effects(fn(raise = TRUE))
capture_side_effects(fn(raise = TRUE), copy_env = TRUE)
dputSelectedAddin

Replaces selected code with its dput() output

Description

Experimental

RStudio Addin: Runs `dput()` on the selected code and inserts it instead of the selection.

See `Details` for how to set a key command.

Usage

dputSelectedAddin(selection = NULL, insert = TRUE, indentation = 0)

Arguments

- **selection**: String of code. (Character)
  
  E.g. "stop('This gives an expect_error test')."

  **N.B.** Mainly intended for testing the addin programmatically.

- **insert**: Whether to insert the expectations via `rstudioapi::insertText()` or return them. (Logical)

  **N.B.** Mainly intended for testing the addin programmatically.

- **indentation**: Indentation of the selection. (Numeric)

  **N.B.** Mainly intended for testing the addin programmatically.

Details

**How:** Parses and evaluates the selected code string, applies `dput()` and inserts the output instead of the selection.

**How to set up a key command in RStudio:**

After installing the package. Go to:

Tools >> Addins >> Browse Addins >> Keyboard Shortcuts.

Find "dput() Selected" and press its field under Shortcut.

Press desired key command, e.g. Alt+D.

Press Apply.

Press Execute.

Value

Inserts the output of running `dput()` on the selected code.

Does not return anything.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also

Other addins: `assertCollectionAddin()`, `initializeGXSFunctionAddin()`, `initializeTestthatAddin()`, `insertExpectationsAddin()`, `navigateTestFileAddin()`, `wrapStringAddin()`
Description

Experimental

Applies `class()` to each element of `x` (without recursion). When `class()` returns multiple strings, the first class string is returned.

Usage

```r
element_classes(x, keep_names = FALSE)
```

Arguments

- `x` List with elements.
- `keep_names` Whether to keep existing names. (Logical)

Details

Gets first string in `class()` for all elements.

Value

The main class of each element.

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

See Also

Other element descriptors: `element_lengths()`, `element_types()`, `num_total_elements()`

Examples

```r
# Attach packages
library(xpectr)

l <- list("a" = c(1,2,3), "b" = "a", "c" = NULL)

element_classes(l)
element_classes(l, keep_names = TRUE)
```
element_lengths

Gets the length of each element

Description

Experimental

Applies `length()` to each element of `x` (without recursion).

Usage

`element_lengths(x, keep_names = FALSE)`

Arguments

- `x` List with elements.
- `keep_names` Whether to keep existing names. (Logical)

Details

Simple wrapper for `unlist(lapply(x,length))`.

Value

The length of each element.

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

See Also

Other element descriptors: `element_classes()`, `element_types()`, `num_total_elements()`

Examples

```r
# Attach packages
library(xpectr)

l <- list("a" = c(1,2,3), "b" = 1, "c" = NULL)

element_lengths(l)
element_lengths(l, keep_names = TRUE)
```
element_types

Gets the type of each element

Description

Experimental

Applies `typeof()` to each element of `x` (without recursion).

Usage

`element_types(x, keep_names = FALSE)`

Arguments

- `x`: List with elements.
- `keep_names`: Whether to keep existing names. (Logical)

Details

Simple wrapper for `unlist(lapply(x, typeof))`.

Value

The type of each element.

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

See Also

Other element descriptors: `element_classes()`, `element_lengths()`, `num_total_elements()`

Examples

```r
# Attach packages
library(xpectr)

l <- list("a" = c(1,2,3), "b" = "a", "c" = NULL)

element_types(l)

element_types(l, keep_names = TRUE)
```
gxs_function
Generate testthat expectations for argument values in a function

Description

Experimental
Based on a set of supplied values for each function argument, a set of testthat expect_* statements are generated.

Included tests: The first value supplied for an argument is considered the valid baseline value. For each argument, we create tests for each of the supplied values, where the other arguments have their baseline value.

When testing a function that alters non-local variables, consider enabling `copy_env`.
See supported objects in details.

Usage

gxs_function(
  fn,
  args_values,
  extra_combinations = NULL,
  check_nulls = TRUE,
  indentation = 0,
  tolerance = "1e-4",
  round_to_tolerance = TRUE,
  strip = TRUE,
  sample_n = 30,
  envir = NULL,
  copy_env = FALSE,
  assign_output = TRUE,
  seed = 42,
  add_wrapper_comments = TRUE,
  add_test_comments = TRUE,
  start_with_newline = TRUE,
  end_with_newline = TRUE,
  out = "insert",
  parallel = FALSE
)

Arguments

fn Function to create tests for.
args_values The arguments and the values to create tests for. Should be supplied as a named list of lists, like the following:
  args_values = list(
    "x1" = list(1,2,3),
    "x2" = list("a","b","c")
  )
The first value for each argument (referred to as the 'baseline' value) should be valid (not throw an error/message/warning).
N.B. This is not checked but should lead to more meaningful tests.

N.B. Please define the list directly in the function call. This is currently necessary.

extra_combinations

Additional combinations to test. List of lists, where each combination is a named sublist.

E.g. the following two combinations:

```r
eextra_combinations = list(
  list("x1" = 4,"x2" = "b"),
  list("x1" = 7,"x2" = "c")
)
```

N.B. Unspecified arguments gets the baseline value.

If you find yourself adding many combinations, an additional `gxs_function()`
call with different baseline values might be preferable.

check_nulls

Whether to try all arguments with `NULL`. (Logical)

When enabled, you don’t need to add `NULL` to your `args_values`, unless it
should be the baseline value.

indentation

Indentation of the selection. (Numeric)

tolerance

The tolerance for numeric tests as a string, like "1e-4". (Character)

round_to_tolerance

Whether to round numeric elements to the specified tolerance. (Logical)

This is currently applied to numeric columns and vectors (excluding some lists).

strip

Whether to insert `strip_msg()` and `strip()` in tests of side effects. (Logical)

Sometimes testthat tests have differences in punctuation and newlines on different
systems. By stripping both the error message and the expected message of non-alphanumeric symbols, we can avoid such failed tests.

sample_n

The number of elements/rows to sample. Set to `NULL` to avoid sampling.

Inserts `smpl()` in the generated tests when sampling was used. A seed is set
internally, setting `sample.kind` as "Rounding" to ensure compatibility with R
versions < 3.6.0.

The order of the elements/rows is kept intact. No replacement is used, why no
oversampling will take place.

When testing a big `data.frame`, sampling the rows can help keep the test files
somewhat readable.

eenvir

Environment to evaluate in. Defaults to `parent.frame()`.

copy_env

Whether each combination should be tested in a deep copy of the environment.

(Logical)

Side effects will be captured in copies of the copy, why two copies of the envi-
ronment will exist at the same time.

Disabled by default to save memory but is often preferable to enable, e.g. when
the function changes non-local variables.

assign_output

Whether to assign the output of a function call or long selection to a variable.

This will avoid recalling the function and decrease cluttering. (Logical)

Heuristic: when the "selection" isn’t of a string and contains a parenthesis, it
is considered a function call. A selection with more than 30 characters will be
assigned as well.

The tests themselves can be more difficult to interpret, as you will have to look
at the assignment to see the object that is being tested.
seed  
  seed to set. (Whole number)
add_wrapper_comments  
  Whether to add intro and outro comments. (Logical)
add_test_comments  
  Whether to add comments for each test. (Logical)
start_with_newline  
  Whether to have a newline in the beginning/end. (Logical)
end_with_newline  
  Whether to have a newline in the beginning/end. (Logical)
out  
  Either "insert" or "return".
    "insert" (Default): Inserts the expectations via \texttt{rstudioapi::insertText()}.
    "return": Returns the expectations in a list.
  These can be prepared for insertion with \texttt{prepare_insertion()).
parallel  
  Whether to parallelize the generation of expectations. (Logical)
  Requires a registered parallel backend. Like with \texttt{doParallel::registerDoParallel).

Details

The following "types" are currently supported or intended to be supported in the future. Please suggest more types and tests in a GitHub issue!

Note: A set of fallback tests will be generated for unsupported objects.

<table>
<thead>
<tr>
<th>Type</th>
<th>Supported</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Side effects</td>
<td>Yes</td>
<td>Errors, warnings, and messages.</td>
</tr>
<tr>
<td>Vector</td>
<td>Yes</td>
<td>Lists are treated differently, depending on their structure.</td>
</tr>
<tr>
<td>Factor</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Data Frame</td>
<td>Yes</td>
<td>List columns (like nested tibbles) are currently skipped.</td>
</tr>
<tr>
<td>Matrix</td>
<td>Yes</td>
<td>Supported but could be improved.</td>
</tr>
<tr>
<td>Formula</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Array</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Dates</td>
<td>No</td>
<td>Base and lubridate.</td>
</tr>
<tr>
<td>ggplot2</td>
<td>No</td>
<td>This may be a challenge, but would be cool!</td>
</tr>
</tbody>
</table>

Value

Either NULL or the unprepared expectations as a character vector.

Author(s)

Ludvig Renbo Olsen, \texttt{<r-pkgs@ludvigolsen.dk>}

See Also

Other expectation generators: \texttt{gxs_selection()}, \texttt{initializeGXSFunctionAddin()}, \texttt{insertExpectationsAddin()}

Examples

# Attach packages
library(xpectr)

## Not run:
fn <- function(x, y, z)
  if (x > 3) stop("x > 3")
  if (y < 0) warning("y < 0")
  if (z == 10) message("z was 10!")
  x + y + z

# Create expectations
# Note: define the list in the call
gxs_function(fn,
  args_values = list(
    "x" = list(2, 4, NA),
    "y" = list(0, -1),
    "z" = list(5, 10))
)

# Add additional combinations

### End(Not run)

---

**gxs_selection**

*Generate testthat expectations from selection*

### Description

**Experimental**

Based on the selection (string of code), a set of testthat `expect_*` statements are generated. Example: If the selected code is the name of a `data.frame` object, it will create an `expect_equal` test for each column, along with a test of the column names, types and classes, dimensions, grouping keys, etc.

See supported objects in details.

When testing a function that alters non-local variables, consider enabling `copy_env`.

Feel free to suggest useful tests etc. in a GitHub issue!

Addin: `insertExpectationsAddin()`

### Usage

```r

```
indentation = 0,
tolerance = "1e-4",
round_to_tolerance = TRUE,
strip = TRUE,
sample_n = 30,
envir = NULL,
copy_env = FALSE,
assign_output = TRUE,
seed = 42,
test_id = NULL,
add_wrapper_comments = TRUE,
add_test_comments = TRUE,
start_with_newline = TRUE,
end_with_newline = TRUE,
out = "insert"
)

Arguments

selection  String of code. (Character)
  E.g. "stop('This gives an expect_error test')".
indentation  Indentation of the selection. (Numeric)
tolerance  The tolerance for numeric tests as a string, like "1e-4". (Character)
round_to_tolerance  Whether to round numeric elements to the specified tolerance. (Logical)
  This is currently applied to numeric columns and vectors (excluding some lists).
strip  Whether to insert strip_msg() and strip() in tests of side effects. (Logical)
  Sometimes testthat tests have differences in punctuation and newlines on different systems. By stripping both the error message and the expected message of non-alphanumeric symbols, we can avoid such failed tests.
sample_n  The number of elements/rows to sample. Set to NULL to avoid sampling.
  Inserts smpl() in the generated tests when sampling was used. A seed is set internally, setting sample.kind as "Rounding" to ensure compatibility with R versions < 3.6.0.
  The order of the elements/rows is kept intact. No replacement is used, why no oversampling will take place.
  When testing a big data.frame, sampling the rows can help keep the test files somewhat readable.
envir  Environment to evaluate in. Defaults to parent.frame().
copy_env  Whether to work in a deep copy of the environment. (Logical)
  Side effects will be captured in copies of the copy, why two copies of the environment will exist at the same time.
  Disabled by default to save memory but is often preferable to enable, e.g. when the function changes non-local variables.
assign_output  Whether to assign the output of a function call or long selection to a variable.
  This will avoid recalling the function and decrease cluttering. (Logical)
  Heuristic: when the `selection` isn't of a string and contains a parenthesis, it is considered a function call. A selection with more than 30 characters will be assigned as well.
The tests themselves can be more difficult to interpret, as you will have to look at the assignment to see the object that is being tested.

**seed**
Seed to set. (Whole number)

**test_id**
Number to append to assignment names. (Whole number)
For instance used to create the "output_" name: output_<test_id>.

**add_wrapper_comments**
Whether to add intro and outro comments. (Logical)

**add_test_comments**
Whether to add comments for each test. (Logical)

**start_with_newline**, **end_with_newline**
Whether to have a newline in the beginning/end. (Logical)

**out**
Either "insert" or "return".

"**insert**" (Default): Inserts the expectations via `r studioapi::insertText()`.

"**return**": Returns the expectations in a list.
These can be prepared for insertion with `prepare_insertion()`.

Details

The following "types" are currently supported or intended to be supported in the future. Please suggest more types and tests in a GitHub issue!

Note: A set of fallback tests will be generated for unsupported objects.

<table>
<thead>
<tr>
<th>Type</th>
<th>Supported</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Side effects</td>
<td>Yes</td>
<td>Errors, warnings, and messages.</td>
</tr>
<tr>
<td>Vector</td>
<td>Yes</td>
<td>Lists are treated differently, depending on their structure.</td>
</tr>
<tr>
<td>Factor</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Data Frame</td>
<td>Yes</td>
<td>List columns (like nested tibbles) are currently skipped.</td>
</tr>
<tr>
<td>Matrix</td>
<td>Yes</td>
<td>Supported but could be improved.</td>
</tr>
<tr>
<td>Formula</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Array</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Dates</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ggplot2</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

Value

Either NULL or the unprepared expectations as a character vector.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also

Other expectation generators: `gxs_function()`, `initializeGXSFunctionAddin()`, `insertExpectationsAddin()`

Examples

# Attach packages
library(xpectr)

## Not run:
df <- data.frame(
  a = c(1, 2, 3),
  b = c('t', 'y', 'u'),
  stringsAsFactors = FALSE)

gxs_selection("stop('This gives an expect_error test!')")
gxs_selection("warning('This gives a set of side effect tests!')")
gxs_selection("message('This also gives a set of side effect tests!')")
gxs_selection("stop('This: tests the -> punctuation!'), strip = FALSE")
gxs_selection("sum(1, 2, 3, 4)")

tests <- gxs_selection("df", out = "return")
for_insertion <- prepare_insertion(tests)
rstudioapi::insertText(for_insertion)

## End(Not run)

initializeGXSFunctionAddin

**Initialize gxs_function() call**

### Description

**Experimental**

Initializes the gxs_function() call with the arguments and default values of the selected function. See `Details` for how to set a key command.

### Usage

```
initializeGXSFunctionAddin(selection = NULL, insert = TRUE, indentation = 0)
```

### Arguments

- **selection**
  
  Name of function to test with gxs_function(). (Character)

  **N.B.** Mainly intended for testing the addin programmatically.

- **insert**
  
  Whether to insert the code via `rstudioapi::insertText()` or return them. (Logical)

  **N.B.** Mainly intended for testing the addin programmatically.

- **indentation**
  
  Indentation of the selection. (Numeric)

  **N.B.** Mainly intended for testing the addin programmatically.

### Details

**How:** Parses and evaluates the selected code string within the parent environment. When the output is a function, it extracts the formals (arguments and default values) and creates the initial `args_values` for gxs_function(). When the output is not a function, it throws an error.
**initializeTestthatAddin**

**Description**

Experimental

Inserts code for calling `testthat::test_that()`.

See `Details` for how to set a key command.

**Usage**

`initializeTestthatAddin(insert = TRUE, indentation = NULL)`

**Arguments**

- `insert` Whether to insert the code via `rstudioapi::insertText()` or return it. (Logical)
  
  **N.B.** Mainly intended for testing the addin programmatically.

- `indentation` Indentation of the code. (Numeric)
  
  **N.B.** Mainly intended for testing the addin programmatically.

---

**How to set up a key command in RStudio:**

After installing the package. Go to:

Tools >> Addins >> Browse Addins >> Keyboard Shortcuts.

Find “Initialize gxs_function()” and press its field under Shortcut.

Press desired key command, e.g. Alt+F.

Press Apply.

Press Execute.

**Value**

Inserts `gxs_function()` call for the selected function.

Returns NULL invisibly.

**Author(s)**

Ludvig Renbo Olsen, &lt;r-pkgs@ludvigolsen.dk&gt;

**See Also**

Other expectation generators: `gxs_function()`, `gxs_selection()`, `insertExpectationsAddin()`

Other addins: `assertCollectionAddin()`, `dputSelectedAddin()`, `initializeTestthatAddin()`, `insertExpectationsAddin()`, `navigateTestFileAddin()`, `wrapStringAddin()`

**initializeTestthatAddin**

*Initializes* `test_that()` *call*
How to set up a key command in RStudio:
After installing the package. Go to:
Tools >> Addins >> Browse Addins >> Keyboard Shortcuts.
Find “Initialize test_that()” and press its field under Shortcut.
Press desired key command, e.g. Alt+T.
Press Apply.
Press Execute.

Value
Inserts code for calling testthat::test_that().
Returns NULL invisibly.

Author(s)
Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also
Other addins: assertCollectionAddin(), dputSelectedAddin(), initializeGXSFunctionAddin(),
insertExpectationsAddin(), navigateTestFileAddin(), wrapStringAddin()

insertExpectationsAddin
Creates testthat tests for selected code

Description
Experimental
Inserts relevant expect_* tests based on the evaluation of the selected code.
Example: If the selected code is the name of a `data.frame` object, it will create an expect_equal
test for each column, along with a test of the column names.
Currently supports side effects (error, warnings, messages), `data.frames`, and vectors.
List columns in `data.frames` (like nested tibbles) are currently skipped.
See `Details` for how to set a key command.

Usage

```r
insertExpectationsAddin(
  selection = NULL,
  insert = TRUE,
  indentation = 0,
  copy_env = FALSE
)
```

```r
insertExpectationsCopyEnvAddin(
  selection = NULL,
)```
Arguments

- **insert** (Logical): Whether to insert the expectations via `rstudioapi::insertText()` or return them. 
  
  **N.B.** Mainly intended for testing the addin programmatically.

- **indentation** (Numeric): Indentation of the selection. 
  
  **N.B.** Mainly intended for testing the addin programmatically.

- **copy_env** (Logical): Whether to work in a deep copy of the environment. 
  
  Side effects will be captured in copies of the copy, why two copies of the environment will exist at the same time. 
  
  Disabled by default to save memory but is often preferable to enable, e.g. when the function changes non-local variables.

Details

**How:** Parses and evaluates the selected code string within the parent environment (or a deep copy thereof). Depending on the output, it creates a set of unit tests (like `expect_equal(data[['column']], c(1, 2, 3)))`, and inserts them instead of the selection.

**How to set up a key command in RStudio:**

After installing the package, go to:

*Tools >> Addins >> Browse Addins >> Keyboard Shortcuts.*

Find "Insert Expectations" and press its field under Shortcut. 

Press desired key command, e.g. Alt+E. 

Press Apply. 

Press Execute.

Value

Inserts `testthat::expect_` unit tests for the selected code.

Returns NULL invisibly.

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

See Also

Other expectation generators: `gxs_function()`, `gxs_selection()`, `initializeGXSFunctionAddin()`

Other addins: `assertCollectionAddin()`, `dputSelectedAddin()`, `initializeGXSFunctionAddin()`, `initializeTestthatAddin()`, `navigateTestFileAddin()`, `wrapStringAddin()`
navigateTestFileAddin  Navigates to test file

Description

Experimental

RStudio Addin: Extracts file name and (possibly) line number of a test file from a selection or from clipboard content. Navigates to the file and places the cursor at the line number.

Supported types of strings: "test_x.R:3", "test_x.R#3", "test_x.R".

The string must start with "test_" and contain ".R". It is split at either ":" or "#", with the second element (here "3") being interpreted as the line number.

See `Details` for how to set a key command.

Usage

navigateTestFileAddin(selection = NULL, navigate = TRUE, abs_path = TRUE)

Arguments

- **selection**: String with file name and line number. (Character)
  E.g. "test_x.R:3: ", which navigates to the third line of "/tests/testthat/test_x.R".
  N.B. Mainly intended for testing the addin programmatically.

- **navigate**: Whether to navigate to the file or return the extracted file name and line number. (Logical)
  N.B. Mainly intended for testing the addin programmatically.

- **abs_path**: Whether to return the full path or only the file name when `navigate` is FALSE. N.B. Mainly intended for testing the addin programmatically.

Details

**How to set up a key command in RStudio:**

After installing the package. Go to:

Tools >> Addins >> Browse Addins >> Keyboard Shortcuts.

Find "Go To Test File" and press its field under Shortcut.

Press desired key command, e.g. Alt+N.

Press Apply.

Press Execute.

Value

Navigates to file and line number.

Does not return anything.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>
num_total_elements

See Also

Other addins: assertCollectionAddin(), dputSelectedAddin(), initializeGXSFunctionAddin(), initializeTestthatAddin(), insertExpectationsAddin(), wrapStringAddin()

num_total_elements        Total number of elements

Description

Experimental
Unlists ‘x’ recursively and finds the total number of elements.

Usage

num_total_elements(x, deduplicated = FALSE)

Arguments

x    List with elements.
deduplicated    Whether to only count the unique elements. (Logical)

Details

Simple wrapper for length(unlist(x, recursive = TRUE, use.names = FALSE)).

Value

The total number of elements in ‘x’.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also

Other element descriptors: element_classes(), element_lengths(), element_types()

Examples

# Attach packages
library(xpectr)

l <- list(list(list(1, 2, 3), list(2, list(3, 2))),
          list(1, list(list(2, 4), list(7, 1, list(3, 8)))),
          list(list(2, 7, 8), list(10, 2, list(18, 1, 4))))

num_total_elements(l)
num_total_elements(l, deduplicated = TRUE)
prepare_insertion  Prepare expectations for insertion

Description

Experimental
Collapses a list/vector of expectation strings and adds the specified indentation.

Usage

```r
prepare_insertion(
  strings,
  indentation = 0,
  trim_left = FALSE,
  trim_right = FALSE
)
```

Arguments

- `strings`: Expectation strings. (List or Character) As returned with `gxs_*` functions with `out = "return"`.
- `indentation`: Indentation to add. (Numeric)
- `trim_left`: Whether to trim whitespaces from the beginning of the collapsed string. (Logical)
- `trim_right`: Whether to trim whitespaces from the end of the collapsed string. (Logical)

Value

A string for insertion with `rstudioapi::insertText()`.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

Examples

```r
# Attach packages
library(xpectr)

## Not run:
df <- data.frame(
  a = c(1, 2, 3),
  b = c('t', 'y', 'u'),
  stringsAsFactors = FALSE)

tests <- gxs_selection("df", out = "return")
for_insertion <- prepare_insertion(tests)
for_insertion
rstudioapi::insertText(for_insertion)

## End(Not run)
```
**set_test_seed**  
*Set random seed for unit tests*

**Description**

**Experimental**

In order for tests to be compatible with R versions < 3.6.0, we set the `sample.kind` argument in `set.seed()` to "Rounding" when using R versions $\geq$ 3.6.0.

**Usage**

```r
set_test_seed(seed = 42, ...)  
```

**Arguments**

- **seed**  
  Random seed.

- **...**  
  Named arguments to `set.seed()`.

**Details**

Initially contributed by R. Mark Sharp (github: @rmsharp).

**Value**

NULL.

**Author(s)**

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

R. Mark Sharp

---

**simplified_formals**  
*Extract and simplify a function's formal arguments*

**Description**

**Experimental**

Extracts `formals` and formats them as an easily testable character vector.

**Usage**

```r
simplified_formals(fn)  
```

**Arguments**

- **fn**  
  Function.

**Value**

A character vector with the simplified formals.
Examples

# Attach packages
library(xpectr)

fn1 <- function(a = "x", b = NULL, c = NA, d){
  paste0(a, b, c, d)
}

simplified_formals(fn1)

smpl(data, n, keep_order = TRUE, seed = 42)

Arguments

data vector or data.frame. (Logical)
n Number of elements/rows to sample.
N.B. No replacement is used, why n > the number of elements/rows in `data`
won’t perform oversampling.
keep_order Whether to keep the order of the elements. (Logical)
seed seed to use.
The seed is set with sample.kind = "Rounding" for compatibility with R
versions < 3.6.0.

Value

When `data` has <= `n` elements, `data` is returned. Otherwise, `data` is sampled and returned.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>
Examples

# Attach packages
library(xpectr)

smpl(c(1,2,3,4,5), n = 3)
smpl(data.frame("a" = c(1,2,3,4,5), "b" = c(2,3,4,5,6), stringsAsFactors = FALSE), n = 3)

---

stop_if **Simple side effect functions**

Description

**Experimental**

If the `condition` is TRUE, generate error/warning/message with the supplied message.

Usage

```r
stop_if(condition, message = NULL, sys.parent.n = 0L)
warn_if(condition, message = NULL, sys.parent.n = 0L)
mmessage_if(condition, message = NULL, sys.parent.n = 0L)
```

Arguments

- **condition** The condition to check. (Logical)
- **message** Message. (Character)
  
  Note: If NULL, the `condition` will be used as message.
- **sys.parent.n** The number of generations to go back when calling the message function.

Details

When `condition` is FALSE, they return NULL invisibly.

When `condition` is TRUE:

- **stop_if()**: Throws error with the supplied message.
- **warn_if()**: Throws warning with the supplied message.
- **message_if()**: Generates message with the supplied message.

Value

Returns NULL invisibly.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>
Examples

```r
# Attach packages
library(xpectr)
## Not run:
  a <- 0
  stop_if(a == 0, "'a' cannot be 0.")
  warn_if(a == 0, "'a' was 0.")
  message_if(a == 0, "'a' was so kind to be 0.")

## End(Not run)
```

---

**strip**  
Strip strings of non-alphanumeric characters

### Description

**Experimental**

1. Removes any character that is not alphanumeric or a space.
2. (Disabled by default): Remove numbers.
3. Reduces multiple consecutive whitespaces to a single whitespace and trims ends.

Can for instance be used to simplify error messages before checking them.

### Usage

```r
strip(
  strings,
  replacement = "",
  remove_spaces = FALSE,
  remove_numbers = FALSE,
  remove_ansi = TRUE,
  lowercase = FALSE,
  allow_na = TRUE
)
```

### Arguments

- **strings**: vector of strings. (Character)
- **replacement**: What to replace blocks of punctuation with. (Character)
- **remove_spaces**: Whether to remove all whitespaces. (Logical)
- **remove_numbers**: Whether to remove all numbers. (Logical)
- **remove_ansi**: Whether to remove ANSI control sequences. (Logical)
- **lowercase**: Whether to make the strings lowercase. (Logical)
- **allow_na**: Whether to allow strings to contain NAs. (Logical)
strip_msg

Details

1. ANSI control sequences are removed with `fansi::strip_ctl()`.
2. `gsub("[^[:alnum:][:blank:]]", replacement, strings))`
3. `gsub('[0-9]+', '', strings)` (Note: only if specified!)
4. `trimws(gsub("[[[:blank:]]]+"," ",strings))` (Or "" if `remove_spaces` is TRUE)

Value

The stripped strings.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also

Other strippers: `strip_msg()`

Examples

```r
# Attach packages
library(xpectr)

strings <- c("Hello! I am George. \n\rDon't call me Frank! 123",
" \tAs that, is, not, my, name!"
)

strip(strings)
strip(strings, remove_spaces = TRUE)
strip(strings, remove_numbers = TRUE)
```

---

**strip_msg**

Strip side-effect messages of non-alphanumeric characters and rethrow them

Description

**Experimental**

Catches side effects (error, warnings, messages), strips the message strings of non-alphanumeric characters with `strip()` and regenerates them.

When numbers in error messages vary slightly between systems (and this variation isn’t important to catch), we can strip the numbers as well.

Use case: Sometimes `testthat` tests have differences in punctuation and newlines on different systems. By stripping both the error message and the expected message (with `strip()`), we can avoid such failed tests.
Usage

strip_msg(
  x,
  remove_spaces = FALSE,
  remove_numbers = FALSE,
  remove_ansi = TRUE,
  lowercase = FALSE
)

Arguments

  x          Code that potentially throws warnings, messages, or an error.
  remove_spaces  Whether to remove all whitespaces. (Logical)
  remove_numbers Whether to remove all numbers. (Logical)
  remove_ansi   Whether to remove ANSI control sequences. (Logical)
  lowercase     Whether to make the strings lowercase. (Logical)

Value

Returns NULL invisibly.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also

Other strippers: strip()

Examples

# Attach packages
library(xpectr)
library(testthat)

### Not run:
strip_msg(stop("this 'dot' \n is removed! 123"))
strip_msg(warning("this 'dot' \n is removed! 123"))
strip_msg(message("this 'dot' \n is removed! 123"))
strip_msg(message("this 'dot' \n is removed! 123"), remove_numbers = TRUE)
error_fn <- function(){stop("this 'dot' \n is removed! 123")}
strip_msg(error_fn())

# With testthat tests
expect_error(strip_msg(error_fn()),
  strip("this 'dot' \n is removed! 123"))
expect_error(strip_msg(error_fn(), remove_numbers = TRUE),
  strip("this 'dot' \n is removed! 123", remove_numbers = TRUE))

### End(Not run)
**suppress_mw**

*Suppress warnings and messages*

**Description**

**Experimental**

Run expression wrapped in both `suppressMessages()` and `suppressWarnings()`.

**Usage**

```
suppress_mw(expr)
```

**Arguments**

`expr` Any expression to run within `suppressMessages()` and `suppressWarnings()`.

**Details**

`suppressWarnings(suppressMessages(expr))`

**Value**

The output of `expr`.

**Author(s)**

Ludvig Renbo Olsen, <r-pkgs@ludvig Olsen.dk>

**Examples**

```r
# Attach packages
library(xpectr)

fn <- function(a, b){
  warning("a warning")
  message("a message")
  a + b
}

suppress_mw(fn(1, 5))
```
wrapStringAddin

Wraps the selection with `paste0()` call.

See Details for how to set a key command.

Usage

```r
wrapStringAddin(
  selection = NULL,
  indentation = 0,
  every_n = NULL,
  tolerance = 10,
  insert = TRUE
)
```

Arguments

- `selection` String of code. (Character)
  - **N.B.** Mainly intended for testing the addin programmatically.
- `indentation` Indentation of the selection. (Numeric)
  - **N.B.** Mainly intended for testing the addin programmatically.
- `every_n` Number of characters per split.
  - If `NULL`, the following is used to calculate the string width:
  ```r
  max(min(80 - indentation, 70), 50)
  ```
  - **N.B.** Strings shorter than `every_n + tolerance` will not be wrapped.
- `tolerance` Tolerance. Number of characters.
  - We may prefer not to split a string that's only a few characters too long. Strings shorter than `every_n + tolerance` will not be wrapped.
- `insert` Whether to insert the wrapped text via `rstudioapi::insertText()` or return it. (Logical)
  - **N.B.** Mainly intended for testing the addin programmatically.

Details

**How to set up a key command in RStudio:**

After installing the package. Go to:
Tools >> Addins >> Browse Addins >> Keyboard Shortcuts.
Find "Wrap String with paste0" and press its field under Shortcut.
Press desired key command, e.g. Alt+P.
Press Apply.
Press Execute.
**Value**

Inserts the following (with newlines and correct indentation):

```r
paste0("first n chars","next n chars")
```

Returns NULL invisibly.

**Author(s)**

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

**See Also**

Other addins: `assertCollectionAddin()`, `dputSelectedAddin()`, `initializeGXSFunctionAddin()`, `initializeTestthatAddin()`, `insertExpectationsAddin()`, `navigateTestFileAddin()`
Index

assertCollectionAddin, 2, 6, 17–19, 21, 31
capture_error(), 4
capture_messages(), 4
capture_parse_eval_side_effects, 3, 5
capture_side_effects, 4, 4
capture_side_effects(), 3
capture_warnings(), 4
checkmate assert collection, 2
class(), 7
dput(), 6
dputSelectedAddin, 3, 6, 17–19, 21, 31
element_classes, 7, 8, 9, 21
element_lengths, 7, 8, 9, 21
element_types, 7, 8, 9, 21
expect_equal, 13, 18
fansi::strip_ctl(), 27
forms, 23
gxs_function, 10, 15, 17, 19
gxs_function(), 16, 17
gxs_selection, 12, 13, 17, 19
initializeGXSFunctionAddin, 3, 6, 12, 15, 16, 18, 19, 21, 31
initializeTestthatAddin, 3, 6, 17, 17, 19, 21, 31
insertExpectationsAddin, 3, 6, 12, 15, 17, 18, 19, 21, 31
insertExpectationsAddin(), 13
insertExpectationsCopyEnvAddin (insertExpectationsAddin), 18
length(), 8
message_if (stop_if), 25
navigateTestFileAddin, 3, 6, 17–19, 20, 31
num_total_elements, 7–9, 21
parent.frame(), 4, 5, 11, 14
paste0(), 30
prepare_insertion, 22
prepare_insertion(), 12, 15
rstudioapi::insertText(), 2, 6, 12, 15–17, 19, 21, 22, 30
sample.int(), 24
set.seed(), 23
set_test_seed, 23
simplified_formals, 23
smpl, 24
smpl(), 11, 14
stop_if, 25
strip, 26, 28
strip(), 11, 14, 27
strip_msg, 27, 27
strip_msg(), 11, 14
suppress_mw, 29
suppressMessages(), 29
suppressWarnings(), 29
testthat::expect_*, 19
testthat::test_that(), 17, 18
typeof(), 9
warn_if (stop_if), 25
wrapStringAddin, 3, 6, 17–19, 21, 30
xpectr, 31