Package ‘xpectr’

March 31, 2020

**Title**  Generates Expectations for 'testthat' Unit Testing

**Version**  0.3.0

**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),
            rstudioapi (>= 0.10),
            testthat (>= 2.3.1),
            plyr,
            dplyr,
            tibble,
            rlang,
            utils,
            withr (>= 2.0.0),
            stats,
            checkmate (>= 2.0.0),
            lifecycle

**Suggests**  knitr,
              rmarkdown,
              data.table

**RdMacros**  lifecycle

**Encoding**  UTF-8

**LazyData**  true

**Roxygen**  list(markdown = TRUE)

**RoxygenNote**  7.1.0

**VignetteBuilder**  knitr

\r topics documented:

assertCollectionAddin .................................................. 2
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 \texttt{rstudioapi::insertText()} or return it. (Logical)
\textbf{N.B.} Mainly intended for testing the addin programmatically.

indentation Indentation of the code. (Numeric)
\textbf{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 "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()

capture_parse_eval_side_effects

Capture side effects from parse eval

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
capture_parse_eval_side_effects(string, envir = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>String of code that can be parsed and evaluated in <code>envir</code>.</td>
</tr>
<tr>
<td>envir</td>
<td>Environment to evaluate in. Defaults to <code>parent.frame()</code>.</td>
</tr>
</tbody>
</table>
Value

Named list with the side effects.

Author(s)

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

Examples

```r
# 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

```r
capture_side_effects(expr, envir = NULL, reset_seed = FALSE)
```

Arguments

- `expr` Expression.
- `envir` Environment to evaluate expression in.
- `reset_seed` Whether to reset the random state on exit. (Logical)

Value

Named list with the side effects.

Author(s)

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

```r
# 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! Yeehaa")
  }
  "the output"
}
capture_side_effects(fn())
capture_side_effects(fn(raise = 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

```r
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@ludvig Olsen.dk>

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

See Also

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

Examples

# 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 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()
element_types

## 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)
```

## Description

**Experimental**

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

## Usage

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

## Arguments

- `x`: List with elements.
- `keep_names`: Whether to keep 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.

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,
  assign_output = TRUE,
  seed = 42,
  add_wrapper_comments = TRUE,
  add_test_comments = TRUE,
  start_with_newline = TRUE,
  end_with_newline = TRUE,
  out = "insert"
)
```

**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:
  ```r
  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:
extra_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.

envir
Environment to evaluate in.

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 clumping. (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)
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()}

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 \texttt{NULL} or the unprepared expectations as a character vector.

Author(s)

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

See Also

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

Examples

```r
# Attach packages
library(xpectr)

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
```r
args_values = list(
    "x" = list(2, 4, NA),
    "y" = list(0, -1),
    "z" = list(5, 10))
)
```

# Add additional combinations
```r
gxs_function(fn,
    args_values = list(
        "x" = list(2, 4, NA),
        "y" = list(0, -1),
        "z" = list(5, 10)),
    extra_combinations = list(
        list("x" = 4, "z" = 10),
        list("y" = 1, "z" = 10))
)
```

---

**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.

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

Addin: `insertExpectationsAddin()`

**Usage**

```r
gxs_selection(
    selection,
    indentation = 0,
    tolerance = "1e-4",
    round_to_tolerance = TRUE,
    strip = TRUE,
    sample_n = 30,
    envir = NULL,
    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 `set.seed()` 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.

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 `rstudioapi::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>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, <r-pkgs@ludvigolsen.dk>

See Also

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

Examples

# Attach packages
library(xpectr)

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)")
gxs_selection("df")

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

```r
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.

**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, <r-pkgs@ludvigolsen.dk>
See Also

Other expectation generators: `gxs_function()`, `gxs_selection()`, `insertExpectationsAddin()
Other addins: `assertCollectionAddin()`, `dputSelectedAddin()`, `initializeTestthatAddin()`, 
`insertExpectationsAddin()`, `navigateTestFileAddin()`, `wrapStringAddin()`

initializeTestthatAddin

*Initializes test_that() call*

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.

Details

*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()`
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

insertExpectationsAddin(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 within the parent environment. 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.
navigateTestFileAddin

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.
**num_total_elements**

**Value**

Navigates to file and line number.
Does not return anything.

**Author(s)**

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

**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)))))

num_total_elements(l)
num_total_elements(l, deduplicated = TRUE)

prepare_insertion预备插入

Description

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

Usage

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>
set_test_seed

Examples

# Attach packages
library(xpectr)

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)

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 >= 3.6.0.

Usage

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
simplified_formals(fn)

Arguments
fn
Function.

Value
A character vector with the simplified formals.

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

Examples

```r
# Attach packages
library(xpectr)

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

simplified_formals(fn1)
```

smpl  
Random sampling

Description
Experimental
Samples a vector, factor or data frame. Useful to reduce size of testthat expect_* tests. Not intended for other purposes.

Wraps sample.int(). Data frames are sampled row-wise.
The seed is set within the function with sample.kind as "Rounding" for compatibility with R versions < 3.6.0. On exit, the random state is restored.

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

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

stop_if(condition, message = NULL, sys.parent.n = 0L)
warn_if(condition, message = NULL, sys.parent.n = 0L)
message_if(condition, message = NULL, sys.parent.n = 0L)

Arguments

c 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 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

# Attach packages
library(xpectr)

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.")

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

strip(
  strings,
  replacement = "",
  remove_spaces = FALSE,
  remove_numbers = FALSE,
  allow_na = TRUE
)
strip_msg

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)
- **allow_na**: Whether to allow strings to contain NAs. (Logical)

Details

1. `gsub("^[[:alnum:]]\[:blank:]]\",replacement,strings))`
2. `gsub("0-9","",strings)` (Note: only if specified!
3. `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", "As 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

```r
strip_msg(x, remove_spaces = FALSE, remove_numbers = 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)

Value

Returns NULL invisibly.

Author(s)

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

See Also

Other strippers: `strip()`

Examples

```r
# Attach packages
library(xpectr)
library(testthat)

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))
```
suppress_mw

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@ludvigolsen.dk>

Examples

# Attach packages
library(xpectr)

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

suppress_mw(fn(1, 5))
wrapStringAddin \(\text{Wraps the selection with paste0}\)

**Description**

**Experimental**

Splits the selection every n characters and inserts it in a `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:
  
  \[
  \text{max}(\text{min}(80 - \text{indentation}, 70), 50)
  \]
  
  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.
xpectr

Value

Inserts the following (with newlines and correct indentation):
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()

Description

A set of utilities and RStudio addins for generating tests.

Author(s)

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

assertCollectionAddin, 2, 6, 16, 18, 19, 29

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