Package 'teal.code'

September 12, 2023

Type Package

Title Code Storage and Execution Class for 'teal' Applications

Version 0.4.1

Date 2023-09-06

Description Introduction of 'qenv' S4 class, that facilitates code execution and reproducibility in 'teal' applications.

License Apache License 2.0

URL https://insightsengineering.github.io/teal.code/latest-tag/, https://github.com/insightsengineering/teal.code

BugReports https://github.com/insightsengineering/teal.code/issues

Depends R (>= 4.0)

Imports checkmate (>= 2.1.0), grDevices, lifecycle (>= 0.2.0), methods, rlang (>= 1.1.0), shiny (>= 1.6.0), styler (>= 1.2.0)

Suggests cli (>= 3.4.0), knitr (>= 1.42), magrittr (>= 1.5), rmarkdown (>= 2.19), testthat (>= 3.1.5)

VignetteBuilder knitr

RdMacros lifecycle

Config/Needs/verdepcheck mllg/checkmate, r-lib/lifecycle, r-lib/rlang, rstudio/shiny, r-lib/styler, r-lib/cli, yihui/knitr, tidyverse/magrittr, rstudio/rmarkdown, r-lib/testthat

Config/Needs/website insightsengineering/nesttemplate

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

Collate 'include_css_js.R' 'qenv-class.R' 'qenv-errors.R' 'qenv-concat.R' 'qenv-constructor.R' 'qenv-eval_code.R' 'qenv-get_code.R' 'qenv-get_var.R' 'qenv-get_warnings.R' 'qenv-join.R' 'qenv-show.R' 'teal.code-package.R' 'utils.R'

NeedsCompilation no

Author Dawid Kaledkowski [aut, cre],
Pawel Rucki [aut],
Nikolas Burkoff [aut],
Mahmoud Hallal [aut],
Maciej Nasinski [aut],
Konrad Pagacz [aut],
Junlue Zhao [aut],
F. Hoffmann-La Roche AG [cph, fnd]

Maintainer Dawid Kaledkowski <dawid.kaledkowski@roche.com>

Repository CRAN

Date/Publication 2023-09-12 06:10:03 UTC

R topics documented:

`concat` *Concatenate two* `qenv` *objects*

Description

Combine two `qenv` objects by simple concatenate their environments and the code. We recommend to use the `join` method to have a stricter control in case `x` and `y` contain duplicated bindings and code. RHS argument content has priority over the LHS one.

Usage

```
concat(x, y)

## S4 method for signature 'qenv,qenv'
concat(x, y)

## S4 method for signature 'qenv.error,ANY'
concat(x, y)

## S4 method for signature 'qenv,qenv.error'
concat(x, y)
```

Arguments

x	(qenv)
y	(qenv)

Value

qenv object.

Examples

```
q1 <- new_qenv(
  code = c(iris1 = "iris1 <- iris", mtcars1 = "mtcars1 <- mtcars"),
  env = list2env(list(
    iris1 = iris,
    mtcars1 = mtcars
  ))
)
q2 <- q1
q1 <- eval_code(q1, "iris2 <- iris")
q2 <- eval_code(q2, "mtcars2 <- mtcars")
qq <- concat(q1, q2)
get_code(qq)
```

---

dev_suppress *Suppresses plot display in the IDE by opening a PDF graphics device*

Description

This function opens a PDF graphics device using pdf to suppress the plot display in the IDE. The purpose of this function is to avoid opening graphic devices directly in the IDE.

Usage

```
dev_suppress(x)
```

Arguments

x	lazy binding which generates the plot(s)

Details

The function uses on.exit to ensure that the PDF graphics device is closed (using dev.off) when the function exits, regardless of whether it exits normally or due to an error. This is necessary to clean up the graphics device properly and avoid any potential issues.

Value

No return value, called for side effects.

Examples

```
dev_suppress(plot(1:10))
```

`eval_code` *Evaluate the code in the* qenv *environment*

Description

Given code is evaluated in the qenv environment and appended to the code slot. This means that state of the environment is always a result of the stored code (if qenv was initialized) with reproducible code.

Usage

```
eval_code(object, code)

## S4 method for signature 'qenv,expression'
eval_code(object, code)

## S4 method for signature 'qenv,language'
eval_code(object, code)

## S4 method for signature 'qenv,character'
eval_code(object, code)

## S4 method for signature 'qenv.error,ANY'
eval_code(object, code)
```

Arguments

object	(qenv)
code	(character or language) code to evaluate. Also accepts and stores comments

Value

qenv object.

Examples

```
q1 <- new_qenv(env = list2env(list(a = 1)), code = quote(a <- 1))
q2 <- eval_code(q1, quote(library(checkmate)))
q3 <- eval_code(q2, quote(assert_number(a)))
```

get_code *Get code from* qenv

Description

Get code from qenv

Usage

```
get_code(object, deparse = TRUE)

## S4 method for signature 'qenv'
get_code(object, deparse = TRUE)

## S4 method for signature 'qenv.error'
get_code(object)
```

Arguments

object	(qenv)
deparse	(logical(1)) if the returned code should be converted to character.

Value

named character with the reproducible code.

Examples

```
q1 <- new_qenv(env = list2env(list(a = 1)), code = quote(a <- 1))
q2 <- eval_code(q1, code = quote(b <- a))
q3 <- eval_code(q2, code = quote(d <- 2))
get_code(q3)
get_code(q3, deparse = FALSE)
```

get_var *Get object from the* qenv *environment*

Description

Get object from the qenv environment.

Usage

```
get_var(object, var)

## S4 method for signature 'qenv,character'
get_var(object, var)

## S4 method for signature 'qenv.error,ANY'
get_var(object, var)

## S4 method for signature 'qenv,ANY,missing'
x[[i, j, ...]]

## S3 method for class 'qenv.error'
x[[i, j, ...]]
```

Arguments

object	(qenv)
var	(character(1)) name of the variable to pull from the environment.
x	(qenv)
i	(character) name of the binding in environment (name of the variable)
j	not used
...	not used

Value

The value of required variable (var) within qenv object.

Examples

```
q1 <- new_qenv(env = list2env(list(a = 1)), code = quote(a <- 1))
q2 <- eval_code(q1, code = "b <- a")
get_var(q2, "b")
q2[["b"]]
```

---

get_warnings *Get the warnings of* qenv *object*

Description

Get the warnings of qenv object

Usage

```
get_warnings(object)

## S4 method for signature 'qenv.error'
get_warnings(object)

## S4 method for signature 'qenv'
get_warnings(object)

## S4 method for signature '`NULL`'
get_warnings(object)
```

Arguments

object	(qenv)

Value

character containing warning information or NULL if no warnings

Examples

```
data_q <- new_qenv()
data_q <- eval_code(new_qenv(), "iris_data <- iris")
warning_qenv <- eval_code(
  data_q,
  bquote(p <- hist(iris_data[, .("Sepal.Length")], ff = ""))
)
cat(get_warnings(warning_qenv))
```

---

join *Join two* qenv *objects*

Description

join() perform checks and merges two qenv objects into one qenv object. Any common code at the start of the qenvs is only placed once at the start of the joined qenv. This allows consistent behavior when joining qenvs which share a common ancestor. See below for an example.

Usage

```
join(x, y)

## S4 method for signature 'qenv,qenv'
join(x, y)

## S4 method for signature 'qenv.error,ANY'
join(x, y)
```

```
## S4 method for signature 'qenv,qenv.error'
join(x, y)
```

Arguments

x	(qenv)
y	(qenv)

Details

There are some situations where `join()` cannot be properly performed, such as these three scenarios:

1. Both qenv objects contain an object of the same name but are not identical.

   Example:

   ```
   x <- new_qenv(
     code = c(mtcars1 = "mtcars1 <- mtcars"),
     env = list2env(list(mtcars1 = mtcars))
   )
   y <- new_qenv(
     code = c(mtcars1 = "mtcars1 <- mtcars['wt']"),
     env = list2env(list(mtcars1 = mtcars['wt']))
   )
   z <- join(x, y)
   # Error message will occur
   ```

   In this example, `mtcars1` object exists in both `x` and `y` objects but the content are not identical. `mtcars1` in the `x` qenv object has more columns than `mtcars1` in the `y` qenv object (only has one column).

2. `join()` will look for identical `@id` values in both qenv objects. The index position of these `@id`s must be the same to determine the evaluation order. Otherwise, `join()` will throw an error message.

   Example:

   ```
   common_q <- new_qenv(code = "v <- 1", env = list2env(list(v = 1)))
   x <- eval_code(
     common_q,
     "x <- v"
   )
   y <- eval_code(
     common_q,
     "y <- v"
   )
   z <- eval_code(
     y,
   ```

```
  "z <- v"
)
q <- join(x, y)
join_q <- join(q, z)
# Error message will occur

# Check the order of evaluation based on the id slot
shared_ids <- intersect(q@id, z@id)
match(shared_ids, q@id) # Output: 1 3
match(shared_ids, z@id) # Output: 1 2
```

The error occurs because the index position of identical `@id` between the two objects is not the same.

3. The usage of temporary variable in the code expression could cause `join()` to fail.

   Example:

```
common_q <- new_qenv()
x <- eval_code(
  common_q,
  "x <- numeric(0)
   for (i in 1:2) {
     x <- c(x, i)
   }"
)
y <- eval_code(
  common_q,
  "y <- numeric(0)
   for (i in 1:3) {
     y <- c(y, i)
   }"
)
q <- join(x,y)
# Error message will occur

# Check the value of temporary variable i in both objects
x@env$i # Output: 2
y@env$i # Output: 3
```

   `join()` fails to provide a proper result because of the temporary variable `i` exists in both objects but has different value.
   To fix this, we can set `i <- NULL` in the code expression for both objects.

```
common_q <- new_qenv()
x <- eval_code(
  common_q,
  "x <- numeric(0)
   for (i in 1:2) {
```

```
      x <- c(x, i)
    }
    # dummy i variable to fix it
    i <- NULL"
)
y <- eval_code(
  common_q,
  "y <- numeric(0)
   for (i in 1:3) {
     y <- c(y, i)
   }
   # dummy i variable to fix it
   i <- NULL"
)
q <- join(x,y)
```

Value

qenv object.

Examples

```
q1 <- new_qenv(
  code = c(iris1 = "iris1 <- iris", mtcars1 = "mtcars1 <- mtcars"),
  env = list2env(list(
    iris1 = iris,
    mtcars1 = mtcars
  ))
)
q2 <- q1
q1 <- eval_code(q1, "iris2 <- iris")
q2 <- eval_code(q2, "mtcars2 <- mtcars")
qq <- join(q1, q2)
get_code(qq)

common_q <- new_qenv(list2env(list(x = 1)), quote(x <- 1))
y_q <- eval_code(common_q, quote(y <- x * 2))
z_q <- eval_code(common_q, quote(z <- x * 3))
join_q <- join(y_q, z_q)
# get_code only has "x <- 1" occurring once
get_code(join_q)
```

---

new_qenv *Initialize* qenv *object*

Description

Initialize qenv object with code and env. In order to have qenv reproducible one needs to initialize with env which can be reproduced by the code. Alternatively, one can create an empty qenv and evaluate the expressions in this object using eval_code.

Usage

```
new_qenv(env = new.env(parent = parent.env(.GlobalEnv)), code = expression())

## S4 method for signature 'environment,expression'
new_qenv(env = new.env(parent = parent.env(.GlobalEnv)), code = expression())

## S4 method for signature 'environment,character'
new_qenv(env = new.env(parent = parent.env(.GlobalEnv)), code = expression())

## S4 method for signature 'environment,language'
new_qenv(env = new.env(parent = parent.env(.GlobalEnv)), code = expression())

## S4 method for signature 'missing,missing'
new_qenv(env = new.env(parent = parent.env(.GlobalEnv)), code = expression())
```

Arguments

env	(environment) Environment being a result of the code evaluation.
code	(character(1) or language) code to evaluate. Accepts and stores comments also.

Value

qenv object.

Examples

```
new_qenv(env = list2env(list(a = 1)), code = quote(a <- 1))
new_qenv(env = list2env(list(a = 1)), code = parse(text = "a <- 1"))
new_qenv(env = list2env(list(a = 1)), code = "a <- 1")
```

---

show,qenv-method *Show the* qenv *object*

Description

Prints the qenv object

Usage

```
## S4 method for signature 'qenv'
show(object)
```

Arguments

object	(qenv)

Value

nothing

Examples

```
q1 <- new_qenv(
  code = "a <- 5
    b <- data.frame(x = 1:10)",
  env = list2env(list(a = 5, b = data.frame(x = 1:10)))
)
q1
```

Index