Package 'dipsaus'

November 17, 2025

Type Package

Title A Dipping Sauce for Data Analysis and Visualizations

Version 0.3.2

Description Works as an ``add-on'' to packages like 'shiny', 'future', as well as 'rlang', and provides utility functions. Just like dipping sauce adding flavors to potato chips or pita bread, 'dipsaus' for data analysis and visualizations adds handy functions and enhancements to popular packages. The goal is to provide simple solutions that are frequently asked for online, such as how to synchronize 'shiny' inputs without freezing the app, or how to get memory size on 'Linux' or 'MacOS' system. The enhancements roughly fall into these four categories: 1. 'shiny' input widgets; 2. high-performance computing using the 'future' package; 3. modify R calls and convert among numbers, strings, and other objects. 4. utility functions to get system information such like CPU chip-set, memory limit, etc.

URL https://github.com/dipterix/dipsaus, https://dipterix.org/dipsaus/

BugReports https://github.com/dipterix/dipsaus/issues

License GPL-3

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

Imports utils, stats, graphics, grDevices, parallel, Rcpp, R6, shiny, cli, stringr, jsonlite (>= 1.6), future, future.apply, progressr, fastmap (>= 1.1.0), base64enc, digest, rlang (>= 0.4.0), rstudioapi (>= 0.11)

Suggests knitr, promises, later, rmarkdown, testthat, microbenchmark, yaml, future.callr

RoxygenNote 7.3.3

LinkingTo Rcpp

VignetteBuilder knitr

NeedsCompilation yes

Author Zhengjia Wang [aut, cre],
John Magnotti [ctb] (Contributed to `rutabaga.R`),
Xiang Zhang [ctb] (Contributed to `rutabaga.R`)

Maintainer Zhengjia Wang <dipterix.wang@gmail.com>

Repository CRAN

Date/Publication 2025-11-17 06:10:17 UTC

Contents

`AbstractMap` *Abstract Map to store key-value pairs*

Description

Abstract Map to store key-value pairs

`AbstractQueue` *Defines abstract queue class*

Description

This class is inspired by https://cran.r-project.org/package=txtq. The difference is `AbstractQueue` introduce an abstract class that can be extended and can queue not only text messages, but also arbitrary R objects, including expressions and environments. All the queue types in this package inherit this class.

Abstract Public Methods

Methods start with `@...` are not thread-safe. Most of them are not used directly by users. However, you might want to override them if you inherit this abstract class. Methods marked as "(override)" are not implemented, meaning you are supposed to implement the details. Methods marked as "(optional)" usually have default alternatives.

`initialize(...)` **(override)** The constructor. Usually three things to do during the process: 1. set `get_locker` `free_locker` if you don't want to use the default lockers. 2. set lock file (if using default lockers). 3. call `self$connect(...)`

`get_locker()`, `free_locker()` **(optional)** Default is `NULL` for each methods, and queue uses an internal `private$default_get_locker` and `private$default_free_locker`. These two methods are for customized locker, please implement these two methods as functions during `self$initialization` `get_locker` obtains and lock access (exclusive), and `free_locker` frees the locker. Once implemented, `private$exclusive` will take care the rest. Type: function; parameters: none; return: none

`@get_head()`, `@set_head(v)` **(override)** Get head so that we know where we are in the queue `self$@get_head()` should return a integer indicating where we are at the queue `self$@set_head(v)` stores that integer. Parameter `v` is always non-negative, this is guaranteed. Users are not supposed to call these methods directly, use `self$head` and `self$head<-` instead. However, if you inherit this class, you are supposed to override the methods.

`@get_total()`, `@set_total(v)` **(override)** Similar to `@get_head` and `@set_head`, defines the total items ever stored in the queue. total-head equals current items in the queue.

`@inc_total(n=1)` **(optional)** Increase total, usually this doesn't need to be override, unless you are using files to store total and want to decrease number of file connections

`@append_header(msg, ...)` **(override)** `msg` will be vector of strings, separated by "|", containing encoded headers: 'time', 'key', 'hash', and 'message'. to decode what's inside, you can use `self$print_items(stringr::str_split_fixed(msg, '\|', 4))`. **Make sure** to return a number, indicating number of items stored. Unless handled elsewhere, usually `return(length(msg))`.

`@store_value(value, key)` **(override)** Defines how to store value. 'key' is unique identifier generated from time, queue ID, and value. Usually I use it as file name or key ID in database. value is an arbitrary R object to store. you need to store value somewhere and return a string that will be passed as 'hash' in `self$restore_value`.

`restore_value(hash, key, preserve = FALSE)` **(override)** Method to restore value from given combination of 'hash' and 'key'. 'hash' is the string returned by `@store_value`, and 'key' is the same as key in `@store_value`. preserve is a indicator of whether to preserve the value for future use. If set to `FALSE`, then you are supposed to free up the resource related to the value. (such as free memory or disk space)

`@log(n = -1, all = FALSE)` `(override)` get `n` items from what you saved to during `@append_header`. `n` less equal than 0 means listing all possible items. If `all=TRUE`, return all items (number of rows should equals to `self$total`), including popped items. If `all=FALSE`, only return items in the queue (number of rows is `self$count`). The returned value should be a `n x 4` matrix. Usually I use `stringr::str_split_fixed(..., '\|', 4)`. Please see all other types implemented for example.

`@reset(...)` **(override)** Reset queue, remove all items and reset head, total to be 0.

`@clean()` **(override)** Clean the queue, remove all the popped items.

`@validate()` **(override)** Validate the queue. Stop if the queue is broken.

`@connect(con, ...)` **(override)** Set up connection. Usually should be called at the end of `self$initialization` to connect to a database, a folder, or an existing queue you should do checks whether the connection is new or it's an existing queue.

`connect(con, ...)` **(optional)** Thread-safe version. sometimes you need to override this function instead of `@connect`, because `private$exclusive` requires `lockfile` to exist and to be

locked. If you don't have lockers ready, or need to set lockers during the connection, override this one.

`destroy()` **(optional)** Destroy a queue, free up space and call `delayedAssign('.lockfile', {stop(...)}, assign.env=private)` to raise error if a destroyed queue is called again later.

Public Methods

Usually don't need to override unless you know what you are doing.

`push(value, message="",...)` Function to push an arbitrary R object to queue. `message` is a string giving notes to the pushed item. Usually message is stored with header, separated from values. The goal is to describe the value. `...` is passed to `@append_header`

`pop(n = 1, preserve = FALSE)` Pop `n` items from the queue. `preserve` indicates whether not to free up the resources, though not always guaranteed.

`print_item(item)`, `print_items(items)` To decode matrix returned by `log()`, returning named list or data frame with four heads: 'time', 'key', 'hash', and 'message'.

`list(n=-1)` List items in the queue, decoded. If `n` is less equal than 0, then list all results. The result is equivalent to `self$print_items(self$log(n))`

`log(n=-1,all=FALSE)` List items in the queue, encoded. This is used with `self$print_items`. When `all=TRUE`, result will list the records ever pushed to the queue since the last time queue is cleaned. When `all=FALSE`, results will be items in the queue. `n` is the number of items.

Public Active Bindings

`id` Read-only property. Returns unique ID of current queue.

`lockfile` The lock file.

`head` Integer, total number of items popped, i.e. inactive items.

`total` Total number of items ever pushed to the queue since last cleaned, integer.

`count` Integer, read-only, equals to total - head, number of active items in the queue

Private Methods or properties

`.id` Don't use directly. Used to store queue ID.

`.lockfile` Location of lock file.

`lock` Preserve the file lock.

`exclusive(expr,...)` Function to make sure the methods are thread-safe

`default_get_locker()` Default method to lock a queue

`default_free_locker` Default method to free a queue

`actionButtonStyled` *Action Button but with customized styles*

Description

Action Button but with customized styles

Usage

```
actionButtonStyled(
  inputId,
  label,
  icon = NULL,
  width = NULL,
  type = "primary",
  btn_type = "button",
  class = "",
  ...
)
```

Arguments

`inputId`, `label`, `icon`, `width`, `...`	passed to `shiny::actionButton`
`type`	button type, choices are 'default', 'primary', 'info', 'success', 'warning', and 'danger'
`btn_type`	HTML tag type, either `"button"` or `"a"`
`class`	additional classes to be added to the button

Value

'HTML' tags

See Also

`updateActionButtonStyled` for how to update the button.

Examples

```
# demo('example-actionButtonStyled', package='dipsaus')

library(shiny)
library(dipsaus)

ui <- fluidPage(
  actionButtonStyled('btn', label = 'Click me', type = 'default'),
  actionButtonStyled('btn2', label = 'Click me2', type = 'primary')
```

```
)


server <- function(input, output, session) {
  btn_types = c('default', 'primary', 'info', 'success', 'warning', 'danger')
  observeEvent(input$btn, {
    btype = btn_types[((input$btn-1) %% (length(btn_types)-1)) + 1]
    updateActionButtonStyled(session, 'btn2', type = btype)
  })
  observeEvent(input$btn2, {
    updateActionButtonStyled(session, 'btn',
                             disabled = c(FALSE,TRUE)[(input$btn2 %% 2) + 1])
  })
}


if( interactive() ){
  shinyApp(ui, server, options = list(launch.browser=TRUE))
}
```

`add_to_session` *Store/Get key-value pairs in 'shiny' session*

Description

If key is missing, it'll be created, otherwise ignored or overwritten.

Usage

```
add_to_session(
  session,
  key = "rave_id",
  val = paste(sample(c(letters, LETTERS, 0:9), 20), collapse = ""),
  override = FALSE
)
```

Arguments

`session`	'Shiny' session
`key`	character, key to store
`val`	value to store
`override`	if key exists, whether to overwrite its value

Value

If session is shiny session, returns current value stored in session, otherwise returns `NULL`

ask_or_default *Read a Line from the Terminal, but with Default Values*

Description

Ask a question and read from the terminal in interactive scenario

Usage

```
ask_or_default(..., default = "", end = "", level = "INFO")
```

Arguments

..., end, level	passed to cat2
default	default value to return in case of blank input

Details

The prompt string will ask a question, providing defaults. Users need to enter the answer. If the answer is blank (no space), then returns the default, otherwise returns the user input.

This can only be used in an interactive session.

Value

A character from the user's input, or the default value. See details.

See Also

cat2, readline, ask_yesno

Examples

```
if(interactive()){
ask_or_default('What is the best programming language?',
               default = 'PHP')
}
```

ask_yesno *Ask and Return True or False from the Terminal*

Description

Ask a question and read from the terminal in interactive scenario

Usage

```
ask_yesno(
  ...,
  end = "",
  level = "INFO",
  error_if_canceled = TRUE,
  use_rs = TRUE,
  ok = "Yes",
  cancel = "No",
  rs_title = "Yes or No:"
)
```

Arguments

`...`, `end`, `level`	passed to `cat2`
`error_if_canceled`	raise error if canceled
`use_rs`	whether to use `rstudioapi` if possible
`ok`	button label for yes
`cancel`	button label for no
`rs_title`	message title if 'RStudio' question box pops up.

Details

The prompt string will ask for an yes or no question. Users need to enter "y", "yes" for yes, "n", "no" or no, and "c" for cancel (case-insensitive).

This can only be used in an `interactive` session.

Value

logical or `NULL` or raise an error. If "yes" is entered, returns `TRUE`; if "no" is entered, returns `FALSE`; if "c" is entered, `error_if_canceled=TRUE` will result in an error, otherwise return `NULL`

See Also

`cat2`, `readline`, `ask_or_default`

Examples

```
if(interactive()){
ask_yesno('Do you know how hard it is to submit an R package and ',
          'pass the CRAN checks?')
ask_yesno('Can I pass the CRAN check this time?')
}
```

---

async *Evaluate expression in* async_expr

Description

Evaluate expression in async_expr

Usage

```
async(expr)
```

Arguments

expr	R expression

See Also

async_expr

---

async_expr *Apply R expressions in a parallel way*

Description

Apply R expressions in a parallel way

Usage

```
async_expr(
  .X,
  .expr,
  .varname = "x",
  envir = parent.frame(),
  .pre_run = NULL,
  .ncore = future::availableCores(),
  ...
)
```

Arguments

.X	a vector or a list to apply evaluation on
.expr	R expression, unquoted
.varname	variable name representing element of each .X
envir	environment to evaluate expressions
.pre_run	expressions to be evaluated before looping.
.ncore	number of CPU cores
...	passed to future::future

Details

async_expr uses lapply and future::future internally. Within each loop, an item in ".X" will be assigned to variable "x" (defined by ".varname") and enter the evaluation. During the evaluation, function async is provided. Expressions within async will be evaluated in another session, otherwise will be evaluated in current session. Below is the workflow:

- Run .pre_run
- For i in seq_along(.X):
  - 1. Assign x with .X[[i]], variable name x is defined by .varname
  - 2. Evaluate expr in current session.
    - a. If async is not called, return evaluated expr
    - b. If async(aync_expr) is called, evaluate aync_expr in another session, and return the evaluation results if aync_expr

Value

a list whose length equals to .X. The value of each item returned depends on whether async is called. See details for workflow.

---

async_flapply *Wrapper for* future.apply::future_lapply

Description

Wrapper for future.apply::future_lapply

Usage

```
async_flapply(X, FUN, ...)
```

Arguments

X, FUN, ...	passing to future.apply::future_lapply

See Also

future_lapply

async_works *Run jobs in other R sessions without waiting*

Description

This function has been deprecated. Please use `lapply_callr` instead.

Usage

```
async_works(
  X,
  FUN,
  ...,
  .globals = NULL,
  .name = "Untitled",
  .rs = FALSE,
  .wait = TRUE,
  .chunk_size = Inf,
  .nworkers = future::availableCores(),
  .simplify = FALSE,
  .quiet = FALSE,
  .log
)
```

Arguments

`X`	vector or list to be applied
`FUN`	function with the first argument to be each element of `X`
`...`	further arguments to be passed to `FUN`
`.globals`	global variables to be evaluated in `FUN`
`.name`	job names, used if backed by `rstudioapi` jobs
`.rs`	whether to use `rstudioapi` jobs
`.wait`	whether to wait for the results
`.chunk_size`	used only when `.wait=FALSE`, chunk size for each workers at a time. Only useful for printing progress messages, but might slow down the process when `.chunk_size` is too small
`.nworkers`	number of workers at a time
`.simplify`	whether to simplify the results, i.e. merge list of results to vectors or arrays
`.quiet`	whether to suppress the printing messages
`.log`	internally used

Details

Unlike `future` package, where the global variables can be automatically detected, `async_works` require users to specify global variables explicitly via `.globals`

`async_works` is almost surely slower than `future.apply` packages. However, it provides a functionality that `future.apply` can hardly achieve: being non-block. When setting `.wait=FALSE`, the process will run in the background, and one may run as many of these tasks as they want. This is especially useful when large data generating process occurs (such as read in from a file, process, generate summarizing reports).

Value

If `.wait=TRUE`, returns the applied results of `FUN` on each of `X`. The result types depend on `.simplify` (compare the difference between `lapply` and `sapply`). If `.wait=FALSE`, then returns a function that can check the result. The function takes `timeout` argument that blocks the session at most `timeout` seconds waiting for the results. See examples.

Examples

```
## Not run:
# requires a sub-process to run the code

# Basic usage
a <- 1
async_works(1:10, function(ii){
  ii + a # sub-process don't know a, hence must pass a as globals
}, .globals = list(a = a))

# non-blocking case
system.time({
  check <- async_works(1:10, function(ii){
    # simulating process, run run run
    Sys.sleep(ii)
    Sys.getpid()
  }, .wait = FALSE)
})

# check the results
res <- check(timeout = 0.1)
attr(res, 'resolved') # whether it's resolved

# block the session waiting for the results
res <- check(timeout = Inf)
attr(res, 'resolved')



## End(Not run)
```

as_pipe *Convert functions to pipe-friendly functions*

Description

Convert functions to pipe-friendly functions

Usage

```
as_pipe(
  x,
  ...,
  call,
  arg_name,
  .name = arg_name,
  .env = parent.frame(),
  .quoted = FALSE
)
```

Arguments

x	R object as input
...	default arguments explicitly display in the returned function
call	a function call, or the function itself
arg_name	argument name to be varied. This argument will be the first argument in the new function so it's pipe-friendly.
.name	new argument name; default is the same as arg_name
.env	executing environment
.quoted	whether call has been quoted

Value

If x is missing, returns a function that takes one argument, otherwise run the function with given x

Examples

```
# modify a function call
vary_title <- as_pipe(call = plot(1:10, 1:10),
                      pch = 16,
                      arg_name = 'main',
                      .name = 'title')
vary_title

# vary_title is pipe-friendly with `pch` default 16
vary_title(title = 'My Title')
```

```
# `pch` is explicit
vary_title(title = 'My Title', pch = 1)

# other variables are implicit
vary_title(title = 'My Title', type = 'l')


# modify a function

f <- function(b = 1, x){ b + x }
f_pipable <- as_pipe(call = f, arg_name = 'x')
f_pipable

f_pipable(2)

# Advanced use

# Set option dipsaus.debug.as_pipe=TRUE to debug
options("dipsaus.debug.as_pipe" = TRUE)

# Both `.(z)` and `z` work

image2 <- as_pipe(call = image(
  x = seq(0, 1, length.out = nrow(z)),
  y = 1:ncol(z),
  z = matrix(1:16, 4),
  xlab = "Time", ylab = "Freq",
  main = "Debug"
), arg_name = 'z')

# main can be overwritten
image2(matrix(1:50, 5), main = "Production")


# reset debug option
options("dipsaus.debug.as_pipe" = FALSE)
```

`attached_packages` *Get attached package names in current session (Internally used)*

Description

Get attached package names in current session (Internally used)

Usage

```
attached_packages(include_base = FALSE)
```

Arguments

include_base	whether to include base packages

Value

characters, package names that are attached in current session

---

base64-url *Encode or decode 'base64'*

Description

Compatible with results from package 'base64url', but implemented with package 'base64enc'. I simply do not like it when I have to depend on two packages that can achieve the same goal. This implementation is slower. If you have 'base64url' installed, please use that version.

Usage

```
base64_urlencode(x)

base64_urldecode(x)
```

Arguments

x	character vector to encode or decode

Value

character vector of the same length as x

Examples

```
x = "plain text"
encoded = base64_urlencode(x)
decoded = base64_urldecode(encoded)
print(encoded)
print(decoded)
```

base64_to_image *Save "Base64" Data to Images*

Description

Save "Base64" Data to Images

Usage

```
base64_to_image(data, path)
```

Arguments

data	characters, encoded "Base64" data for images
path	file path to save to

Value

Absolute path of the saved file

base64_to_string *Convert "Base64" Data to String*

Description

Decode "Base64" data to its generating characters

Usage

```
base64_to_string(what)
```

Arguments

what	characters, encoded "Base64" data

Value

String

Examples

```
input <- "The quick brown fox jumps over the lazy dog"

# Base64 encode
what <- base64enc::base64encode(what = charToRaw(input))

# Base64 decode
base64_to_string(what)
```

baseline_array *Calculate Contrasts of Arrays in Different Methods*

Description

Provides five methods to baseline an array and calculate contrast.

Usage

```
baseline_array(
  x,
  along_dim,
  baseline_indexpoints,
  unit_dims = seq_along(dim(x))[-along_dim],
 method = c("percentage", "sqrt_percentage", "decibel", "zscore", "sqrt_zscore",
    "subtract_mean")
)
```

Arguments

x	array (tensor) to calculate contrast
along_dim	integer range from 1 to the maximum dimension of x. baseline along this dimension, this is usually the time dimension.
baseline_indexpoints	integer vector, which index points are counted into baseline window? Each index ranges from 1 to dim(x)[[along_dim]]. See Details.
unit_dims	integer vector, baseline unit: see Details.
method	character, baseline method options are: "percentage", "sqrt_percentage", "decibel", "zscore", and "sqrt_zscore"

Details

Consider a scenario where we want to baseline a bunch of signals recorded from different locations. For each location, we record n sessions. For each session, the signal is further decomposed into frequency-time domain. In this case, we have the input x in the following form:

$$session x frequency x time x location$$

Now we want to calibrate signals for each session, frequency and location using the first 100 time points as baseline points, then the code will be

$$baseline_array(x, along_dim = 3, 1:100, unit_dims = c(1, 2, 4))$$

along_dim=3 is dimension of time, in this case, it's the third dimension of x. baseline_indexpoints=1:100, meaning the first 100 time points are used to calculate baseline. unit_dims defines the unit signal. Its value c(1,2,4) means the unit signal is per session (first dimension), per frequency (second) and per location (fourth).

In some other cases, we might want to calculate baseline across frequencies then the unit signal is $frequencyxtime$, i.e. signals that share the same session and location also share the same baseline. In this case, we assign `unit_dims=c(1,4)`.

There are five baseline methods. They fit for different types of data. Denote z is an unit signal, z_0 is its baseline slice. Then these baseline methods are:

`"percentage"`

$$\frac{z - \bar{z_0}}{\bar{z_0}} \times 100\%$$

`"sqrt_percentage"`

$$\frac{\sqrt{z} - \bar{\sqrt{z_0}}}{\bar{\sqrt{z_0}}} \times 100\%$$

`"decibel"`

$$10 \times (\log_{10}(z) - \bar{\log_{10}}(z_0))$$

`"zscore"`

$$\frac{z - \bar{z_0}}{sd(z_0)}$$

`"sqrt_zscore"`

$$\frac{\sqrt{z} - \bar{\sqrt{z_0}}}{sd(\sqrt{z_0})}$$

Value

Contrast array with the same dimension as `x`.

Examples

```
library(dipsaus)
set.seed(1)

# Generate sample data
dims = c(10,20,30,2)
x = array(rnorm(prod(dims))^2, dims)

# Set baseline window to be arbitrary 10 timepoints
baseline_window = sample(30, 10)

# ----- baseline percentage change ------

# Using base functions
re1 <- aperm(apply(x, c(1,2,4), function(y){
  m <- mean(y[baseline_window])
  (y/m - 1) * 100
}), c(2,3,1,4))

# Using dipsaus
re2 <- baseline_array(x, 3, baseline_window, c(1,2,4),
                      method = 'percentage')
```

```
# Check different, should be very tiny (double precisions)
range(re2 - re1)

# Check speed for large dataset
if(interactive()){
  dims = c(200,20,300,2)
  x = array(rnorm(prod(dims))^2, dims)
  # Set baseline window to be arbitrary 10 timepoints
  baseline_window = seq_len(100)
  f1 <- function(){
    aperm(apply(x, c(1,2,4), function(y){
      m <- mean(y[baseline_window])
      (y/m - 1) * 100
    }), c(2,3,1,4))
  }
  f2 <- function(){
    # equivalent as bl = x[,,baseline_window, ]
    #
    baseline_array(x, along_dim = 3,
                   baseline_indexpoints = baseline_window,
                   unit_dims = c(1,2,4), method = 'sqrt_percentage')
  }
  microbenchmark::microbenchmark(f1(), f2(), times = 3L)
}
```

`capture_expr` *Captures Evaluation Output of Expressions as One Single String*

Description

Evaluate expression and captures output as characters, then concatenate as one single string.

Usage

```
capture_expr(expr, collapse = "\n", type = c("output", "message"), ...)
```

Arguments

`expr`	R expression
`collapse`	character to concatenate outputs
`type, ...`	passed to `capture.output`

Value

Character of length 1: output captured by `capture.output`

Examples

```
x <- data.frame(a=1:10)
x_str <- capture_expr({
  print(x)
})

x_str

cat(x_str)
```

---

cat2 *Color Output*

Description

Color Output

Usage

```
cat2(
  ...,
  level = "DEBUG",
  print_level = FALSE,
  file = "",
  sep = " ",
  fill = FALSE,
  labels = NULL,
  append = FALSE,
  end = "\n",
 pal = list(DEBUG = "grey60", INFO = "#1d9f34", WARNING = "#ec942c", ERROR = "#f02c2c",
    FATAL = "#763053", DEFAULT = "grey60"),
  use_cli = TRUE,
  bullet = "auto"
)
```

Arguments

...	to be printed
level	'DEBUG', 'INFO', 'WARNING', 'ERROR', or 'FATAL' (total 5 levels)
print_level	if true, prepend levels before messages
file, sep, fill, labels, append	pass to base::cat
end	character to append to the string
pal	a named list defining colors see details
use_cli	logical, whether to use package 'cli'
bullet	character, if use 'cli', which symbol to show. see symbol

Details

There are five levels of colors by default: 'DEBUG', 'INFO', 'WARNING', 'ERROR', or FATAL. Default colors are: 'DEBUG' (`grey60`), 'INFO' (`#1d9f34`), 'WARNING' (`#ec942c`), 'ERROR' (`#f02c2c`), 'FATAL' (`#763053`) and 'DEFAULT' (`#000000`, black). If level is not in preset five levels, the color will be "default"-black color.

Value

none.

`check_installed_packages` *Check If Packages Are Installed, Returns Missing Packages*

Description

Check If Packages Are Installed, Returns Missing Packages

Usage

```
check_installed_packages(
  pkgs,
  libs = base::.libPaths(),
  auto_install = FALSE,
  ...
)
```

Arguments

`pkgs`	vector of packages to install
`libs`	paths of libraries
`auto_install`	automatically install packages if missing
`...`	other parameters for `install.packages`

Value

package names that are not installed

clear_env *Function to clear all elements within environment*

Description

Function to clear all elements within environment

Usage

```
clear_env(env, ...)
```

Arguments

env	environment to clean, can be an R environment, or a fastmap2 instance
...	ignored

Examples

```
env = new.env()
env$a = 1
print(as.list(env))

clear_env(env)
print(as.list(env))
```

col2hexStr *Convert color to Hex string*

Description

Convert color to Hex string

Usage

```
col2hexStr(col, alpha = NULL, prefix = "#", ...)
```

Arguments

col	character or integer indicating color
alpha	NULL or numeric, transparency. See grDevices::rgb
prefix	character, default is "#"
...	passing to adjustcolor

Details

col2hexStr converts colors such as 1, 2, 3, "red", "blue", ... into hex strings that can be easily recognized by 'HTML', 'CSS' and 'JavaScript'. Internally this function uses adjustcolor with two differences:

1. the returned hex string does not contain alpha value if alpha is NULL;
2. the leading prefix "#" can be customized

Value

characters containing the hex value of each color. See details

See Also

adjustcolor

Examples

```
col2hexStr(1, prefix = '0x')      # "0x000000"
col2hexStr('blue')                # "#0000FF"

# Change default palette, see "grDevices::colors()"
grDevices::palette(c('orange3', 'skyblue1'))
col2hexStr(1)                     # Instead of #000000, #CD8500
```

---

collapse *Collapse Sensors And Calculate Summations/Mean*

Description

Collapse Sensors And Calculate Summations/Mean

Usage

```
collapse(x, keep, average = FALSE)
```

Arguments

x	A numeric multi-mode tensor (array), without NA
keep	Which dimension to keep
average	collapse to sum or mean

Value

a collapsed array with values to be mean or summation along collapsing dimensions

Examples

```
# Example 1
x = matrix(1:16, 4)

# Keep the first dimension and calculate sums along the rest
collapse(x, keep = 1)
rowSums(x)  # Should yield the same result

# Example 2
x = array(1:120, dim = c(2,3,4,5))
result = collapse(x, keep = c(3,2))
compare = apply(x, c(3,2), sum)
sum(abs(result - compare)) # The same, yield 0 or very small number (1e-10)

# Example 3 (performance)
# Small data, no big difference, even slower
x = array(rnorm(240), dim = c(4,5,6,2))
microbenchmark::microbenchmark(
  result = collapse(x, keep = c(3,2)),
  compare = apply(x, c(3,2), sum),
  times = 1L, check = function(v){
    max(abs(range(do.call('-', v)))) < 1e-10
  }
)

# large data big difference
x = array(rnorm(prod(300,200,105)), c(300,200,105,1))
microbenchmark::microbenchmark(
  result = collapse(x, keep = c(3,2)),
  compare = apply(x, c(3,2), sum),
  times = 1L , check = function(v){
    max(abs(range(do.call('-', v)))) < 1e-10
  })
```

`compoundInput2` *Compound input that combines and extends shiny inputs*

Description

Compound input that combines and extends shiny inputs

Usage

```
compoundInput2(
  inputId,
  label = "Group",
  components = shiny::tagList(),
  initial_ncomp = 1,
```

```
  min_ncomp = 0,
  max_ncomp = 10,
  value = NULL,
  label_color = NA,
  max_height = NULL,
  ...
)
```

Arguments

inputId	character, shiny input ID
label	character, will show on each groups
components	'HTML' tags that defines and combines HTML components within groups
initial_ncomp	numeric initial number of groups to show, non-negative
min_ncomp	minimum number of groups, default is 0, non-negative
max_ncomp	maximum number of groups, default is 10, greater or equal than min_ncomp
value	list of lists, initial values of each inputs, see examples.
label_color	integer or characters, length of 1 or max_ncomp, assigning colors to each group labels; default is NA, and try to get color from foreground par("fg")
max_height	maximum height of the widget
...	will be ignored

Value

'HTML' tags

See Also

updateCompoundInput2 for how to update inputs

Examples

```
library(shiny); library(dipsaus)
compoundInput2(
  'input_id', 'Group',
    div(
    textInput('text', 'Text Label'),
    sliderInput('sli', 'Slider Selector', value = 0, min = 1, max = 1)
  ),
  label_color = 1:10,
  value = list(
    list(text = '1'),  # Set text first group to be "1"
    list(),               # no settings for second group
    list(sli = 0.2)    # sli = 0.2 for the third group
  ))

# Source - system.file('demo/example-compountInput2.R', package='dipsaus')
```

```
# demo('example-compountInput2', package='dipsaus')

library(shiny)
library(dipsaus)
ui <- fluidPage(
  fluidRow(
    column(
      width = 4,
      compoundInput2(
        'compound', 'Group Label', label_color = c(NA,1:9),
        components = div(
          textInput('txt', 'Text'),
          selectInput('sel', 'Select', choices = 1:10, multiple = TRUE),
          sliderInput('sli', 'Slider', max=1, min=0, val=0.5)
        ),
        value = list(
          list(txt = '1'),  # Set text first group to be "1"
          '',               # no settings for second group
          list(sli = 0.2)    # sli = 0.2 for the third group
        )
      ),
      hr(),
      actionButton('action', 'Update compound input')
    )
  )
)

server <- function(input, output, session) {
  observe({
    print(input$compound)
  })
  observe({
    # Getting specific input at group 1
    print(input$compound_txt_1)
  })
  observeEvent(input$action, {
    updateCompoundInput2(
      session, 'compound',
      # Update values for each components
      value = lapply(1:5, function(ii){
        list(
          txt = sample(LETTERS, 1),
          sel = sample(1:10, 3),
          sli = runif(1)
        )
      }), ncomp = NULL, txt = list(label = as.character(Sys.time())))
  })
}

if( interactive() ){
  shinyApp(ui, server, options = list(launch.browser = TRUE))
}
```

`decorate_function` *Python-style decorator*

Description

Python-style decorator

Usage

```
decorate_function(orig, decor, ...)

lhs %D% rhs
```

Arguments

`orig`, `lhs`	any function
`decor`, `rhs`	decorator function that takes `orig` as its first argument
`...`	passed to `decor`

Examples

```
# Example 1: basic usage
# Decorator that prints summary of results and return results itself
verbose_summary <- function(...){
  summary_args <- list(...)
  function(f){
    function(...){
      results <- f(...)


      print(do.call(
        summary,
        c(list(results), summary_args)
      ))
      results

    }
  }

}

# runs as.list, but through verbose_summary
as_list2 <- decorate_function(as.list, verbose_summary)

# run test
res <- as_list2(1:3)  # will verbose summary
identical(res, as.list(1:3))
```

```
# Example 2
x <- 1:20
y <- x + rnorm(20)

# decorator, add a line with slope 1 with given intercept
abline_xy <- function(b){
  function(f){
    function(...){
      f(...)
      intercept <- get_dots('intercept', 0, ...)
      abline(a = intercept, b = b)
    }
  }
}

# orig, plot whatever x vs jittered+intercept
plot_xy <- function(x, intercept = rnorm(1)){
  plot(x, jitter(x, amount = 3) + intercept)
}

# new function that decorate plot_xy with abline_xy, and
# returns the intercept
plot_xy2 <- decorate_function(plot_xy, abline_xy, b = 1)

# alternatively, you might also want to try
plot_xy2 <- plot_xy %D% abline_xy(b = 1)

plot_xy2(x = 1:20)
```

`deparse_svec` *Convert Integer Vectors To String*

Description

Convert Integer Vectors To String

Usage

```
deparse_svec(
  nums,
  connect = "-",
  concatenate = TRUE,
  collapse = ",",
  max_lag = 1
)
```

Arguments

nums	integer vector
connect	character used to connect consecutive numbers
concatenate	connect strings if there are multiples
collapse	if concatenate, character used to connect strings
max_lag	defines "consecutive", min = 1

Value

strings representing the input vector. For example, `c(1, 2, 3)` returns "1-3".

See Also

parse_svec

Examples

```
deparse_svec(c(1:10, 15:18))
```

---

digest2 *Digest R object with source reference removed*

Description

Digest R object with source reference removed

Usage

```
digest2(object, ..., keep_source = FALSE)
```

Arguments

object, ...	passed to digest
keep_source	whether to keep the code that generates the object; default is false

See Also

removeSource

`dipsaus-rstudio-shortcuts` *Register customized R code to 'RStudio' shortcuts*

Description

'RStudio' keyboard shortcuts is handy, however, it is non-trivial to set shortcuts that run customized code. The proposing functions allow 10 customized R expressions to be registered. The first five (1 to 5) are interactive shortcuts, the rest five (6 to 10) are non-interactive.

Usage

```
rs_add_insertion_shortcut(which, txt, force = FALSE)

rs_add_shortcut(which, expr, force = FALSE, quoted = FALSE)

rs_remove_shortcut(which)

rs_show_shortcut(which)

rs_quick_debug(env = globalenv())
```

Arguments

`which`	integer from 1 to 10, which keyboard shortcut to edit
`txt`	an insertion/replacement shortcut to add
`force`	whether to remove existing shortcut if the hot-key has been registered
`expr`	expression to run if shortcut is pressed
`quoted`	whether `expr` is quoted, default is false
`env`	environment to debug code; default is global environment

Details

There are two steps to register an 'RStudio' keyboard shortcut.

1. Please enable the shortcuts by opening `'Tools' > 'Modify Keyboard Shortcuts'` in 'RStudio' menu bar; search and locate add-in items starting with 'Dipsaus'; register hot-keys of your choices, and then save. It is recommended that these keys are `'Alt' + 1` to `'Alt' + 0`. On Apple, 'Alt' is equivalent to 'option' key.

2. run `rs_add_insertion_shortcut` or `rs_add_shortcut` to customize the behaviors of each shortcuts; see Examples.

Function `rs_quick_debug` provides quick way to debug a script or function without messing up the code. The script only works in 'RStudio'. When executing the quick-debug function, the cursor context will be automatically resolved and nearest debugging code blocks will be searched and executed. To enable this feature, add a line with `"# DIPSAUS: DEBUG START"` in your code,

followed by debugging code blocks in comments. The script will figure it out. Since the 'RStudio' context will be obtained when executing the function, it is recommended to add this function to your shortcuts. By default, if the shortcut-1 is unset, this function will be executed.

Examples

```
## Not run:

# Need to run in RStudio
# Please read the Section 'Details' carefully

# ------------------------------------------

# I assume the shortcuts are Alt+1,2,...,9,0,
# corresponding to shortcuts 1 - 10

# Adds an insertion to Alt+9
rs_add_insertion_shortcut(9, " %?<-% ", force = TRUE)
# restart RStudio and try `Alt+9`


# Adds an expression to Alt+2
rs_add_shortcut(2, {
  expr <- sprintf("system.time({\n%s\n})\n",
                  rstudioapi::selectionGet()$value)
  cat(expr)
  eval(parse(text = expr))
}, force = TRUE)

# Select any valid R code and press Alt+1

# ------------------------------------------

# run this to set your shortcut (one-time setup)
rs_add_shortcut(1, { dipsaus::rs_quick_debug() })

# Add debug feature: insert the following comment anywhere in your code
# You may open a new script in the RStudio

# DIPSAUS: DEBUG START
# message("Debugging...")
# a <- 1
# print(a)
# message("Finished")


# Place your cursor here, press the shortcut key


## End(Not run)
```

do_aggregate *Make aggregate pipe-friendly*

Description

A pipe-friendly wrapper of `aggregate` when using formula as input.

Usage

```
do_aggregate(x, ...)
```

Arguments

x	an R object
...	other parameters passed to `aggregate`

Value

Results from `aggregate`

See Also

`aggregate`

Examples

```
data(ToothGrowth)

ToothGrowth |>
  do_aggregate(len ~ ., mean)
```

do_nothing *A dummy function that literally does nothing*

Description

A dummy function that literally does nothing

Usage

```
do_nothing(...)
```

Arguments

...	ignored

Value

Nothing

`drop_nulls` *Drop* `NULL` *values from list or vectors*

Description

Drop `NULL` values from list or vectors

Usage

```
drop_nulls(x, .invalids = list("is.null"))
```

Arguments

`x`	list to check
`.invalids`	a list of functions, or function name. Default is 'is.null'.

Value

list or vector containing no invalid values

Examples

```
x <- list(NULL,NULL,1,2)
drop_nulls(x)  # length of 2
```

`eval_dirty` *Evaluate expressions*

Description

Evaluate expressions

Usage

```
eval_dirty(expr, env = parent.frame(), data = NULL, quoted = TRUE)
```

Arguments

`expr`	R expression or 'rlang' quo
`env`	environment to evaluate
`data`	dataframe or list
`quoted`	Is the expression quoted? By default, this is `TRUE`. This is useful when you don't want to use an expression that is stored in a variable; see examples

Details

`eval_dirty` uses `base::eval()` function to evaluate expressions. Compare to `rlang::eval_tidy`, which won't affect original environment, `eval_dirty` causes changes to the environment. Therefore if `expr` contains assignment, environment will be changed in this case.

Value

the executed results of `expr` evaluated with side effects.

Examples

```
env = new.env(); env$a = 1
rlang::eval_tidy(quote({a <- 111}), env = env)
print(env$a)  # Will be 1. This is because eval_tidy has no side effect

eval_dirty(quote({a <- 111}), env)
print(env$a)  # 111, a is changed

# Unquoted case
eval_dirty({a <- 222}, env, quoted = FALSE)
print(env$a)
```

`fancyFileInput` *Shiny drag-and-drop file input*

Description

Fancy drag and drop file upload for `shiny` apps.

Usage

```
fancyFileInput(
  inputId,
  label,
  width = NULL,
  after_content = "Drag & drop, or button",
  size = c("s", "m", "l", "xl"),
  ...
)
```

Arguments

`inputId`	the input slot that will be used to access the value
`label`	display label for the control, or NULL for no label.
`width`	the width of the input
`after_content`	tiny content that is to be displayed below the input box
`size`	height of the widget, choices are `'s'`, `'m'`, `'l'`, and `'xl'`
`...`	passed to `fileInput`

Value

See fileInput

Examples

```
library(shiny)
library(dipsaus)

ui <- basicPage(
  fancyFileInput('file_input', "Please upload")
)

if(interactive()) {
  shinyApp(
    ui, server = function(input, output, session){},
    options = list(launch.browser = TRUE)
  )
}
```

---

fastcov2 *Calculate Covariance Matrix in Parallel*

Description

Speed up covariance calculation for large matrices. The default behavior is similar cov. Please remove any NA prior to calculation.

Usage

```
fastcov2(x, y = NULL, col1, col2, df)
```

Arguments

x	a numeric vector, matrix or data frame; a matrix is highly recommended to maximize the performance
y	NULL (default) or a vector, matrix or data frame with compatible dimensions to x; the default is equivalent to y = x
col1	integers indicating the subset (columns) of x to calculate the covariance; default is all the columns
col2	integers indicating the subset (columns) of y to calculate the covariance; default is all the columns
df	a scalar indicating the degrees of freedom; default is nrow(x)-1

Value

A covariance matrix of x and y. Note that there is no NA handling. Any missing values will lead to NA in the resulting covariance matrices.

Examples

```
x <- matrix(rnorm(400), nrow = 100)

# Call `cov(x)` to compare
fastcov2(x)

# Calculate covariance of subsets
fastcov2(x, col1 = 1, col2 = 1:2)

# Speed comparison
x <- matrix(rnorm(100000), nrow = 1000)
microbenchmark::microbenchmark(
  fastcov2 = {
    fastcov2(x, col1 = 1:50, col2 = 51:100)
  },
  cov = {
    cov(x[,1:50], x[,51:100])
  },
  unit = 'ms', times = 10
)
```

---

fastmap2 *A Wrapper for* fastmap::fastmap

Description

fastmap provides a key-value store where the keys are strings and the values are any R objects. It differs from normal environment that fastmap avoids memory leak. fastmap2 is a wrapper for fastmap, which provides several generic functions such that it has similar behaviors to lists or environments

Usage

```
fastmap2(missing_default = NULL)

## S3 method for class 'fastmap2'
x[[name]]

## S3 method for class 'fastmap2'
x$name
```

```
## S3 replacement method for class 'fastmap2'
x[[name]] <- value

## S3 replacement method for class 'fastmap2'
x$name <- value

## S3 method for class 'fastmap2'
x[i, j = NULL, ...]

## S3 replacement method for class 'fastmap2'
x[i, j = NULL, ...] <- value

## S3 method for class 'fastmap2'
names(x)

## S3 method for class 'fastmap2'
print(x, ...)

## S3 method for class 'fastmap2'
length(x)

## S3 method for class 'fastmap2'
as.list(x, recursive = FALSE, sorted = FALSE, ...)
```

Arguments

missing_default	passed to fastmap::fastmap
x	a 'fastmap2' object
name	name, or key of the value
value	any R object
i, j	vector of names
...	passed to other methods
recursive	whether to recursively apply as.list
sorted	whether to sort names; default is false

Value

A list of 'fastmap2' instance

Examples

```
## -------------------------- Basic Usage -------------------------
map <- fastmap2()
map$a = 1
map$b = 2
print(map)
```

```
map[c('a', 'b')]
# Alternative way
map['a', 'b']

map[c('c', 'd')] <- 3:4
# or
map['e', 'f'] <- 5:6

# The order is not guaranteed, unless sort=TRUE
as.list(map)
as.list(map, sort=TRUE)

names(map)
length(map)

## ----------------------- NULL value handles -----------------------
map$b <- NULL
names(map)   # 'b' still exists!
as.list(map) # 'b' is NULL, but still there

# to remove 'b', you have to use `@remove` method
map$`@remove`('b')

## ---------------- Native fastmap::fastmap methods -----------------

# whether map has 'a'
map$`@has`('a')

# Remove a name from list
map$`@remove`('a')

# remove all from list
map$`@reset`()
print(map)
```

`fastquantile` *Calculate single quantile for numerical values*

Description

Slightly faster than `quantile` with `na.rm=TRUE`. The internal implementation uses the 'C++' function `std::nth_element`, which is significantly faster than base R implementation when the length of input `x` is less than `1e7`.

Usage

```
fastquantile(x, q)
```

Arguments

x	numerical vector (integers or double)
q	number from 0 to 1

Value

Identical to `quantile(x, q, na.rm=TRUE)`

Examples

```
# create input x with NAs
x <- rnorm(10000)
x[sample(10000, 10)] <- NA

# compute median
res <- fastquantile(x, 0.5)
res

# base method
res == quantile(x, 0.5, na.rm = TRUE)
res == median(x, na.rm = TRUE)

# Comparison
microbenchmark::microbenchmark(
  {
    fastquantile(x, 0.5)
  },{
    quantile(x, 0.5, na.rm = TRUE)
  },{
    median(x, na.rm = TRUE)
  }
)
```

`fastqueue2` *A Wrapper for* `fastmap::fastqueue`

Description

A Wrapper for `fastmap::fastqueue`

Usage

```
fastqueue2(init = 20L, missing_default = NULL)

## S3 method for class 'fastqueue2'
x[[i]]
```

```
## S3 method for class 'fastqueue2'
x[i, j = NULL, ...]

## S3 method for class 'fastqueue2'
print(x, ...)

## S3 method for class 'fastqueue2'
length(x)

## S3 method for class 'fastqueue2'
as.list(x, ...)
```

Arguments

init, missing_default	passed to fastmap::fastqueue
x	a 'fastqueue2' object
i, j	integer index
...	integer indices or passed to other methods

Value

A list of 'fastqueue2' instance

Examples

```
x <- fastqueue2()

# add elements
x$madd(1, "b", function(){ "c" }, 4, "5")

# print information
print(x)

# get the second element without changing the queue
x[[2]]

# remove and get the first element
x$remove()

# the second item
x[[2]]

# first two items in a list
x[c(1,2)]

print(x)
as.list(x)
```

flex_div *Generate Shiny element with arrangement automatically*

Description

Generate Shiny element with arrangement automatically

Usage

```
flex_div(..., ncols = "auto")
```

Arguments

...	shiny UI elements
ncols	number of columns, either "auto" or vector of positive integers

Details

If multiple numbers of columns are specified, flex_div will guess the best size that will be applied. For button UI, flex_div automatically add "20px" on the top margin.

Value

HTML objects

Examples

```
ui <- flex_div(
  shiny::selectInput('sel', label = 'Select input',
                     choices = '', width = '100%'),
  shiny::textInput('id2', label = html_asis(' '), width = '100%',
                   value = 'Heights aligned'),
  actionButtonStyled('ok2', 'Button', width = '100%',),
  shiny::sliderInput('sl', 'Item 4', min = 1, max = 2,
                     value = 1.5, width = '100%'),
  shiny::fileInput('aa', 'item 5', width = '100%'),
  ncols = c(2,3) # Try to assign 2 or 3 items per column
)
if(interactive()){
  shiny::shinyApp(ui = shiny::fluidPage(shiny::fluidRow(ui)),
                  server = function(input, output, session){})
}
```

forelse *Python-style "for-else" function*

Description

Provide Python-style "for-else" that works as follows: for each element, execute "for" block, if there is break while executing "for" block, then just stop and ignore the "else" statement, otherwise run "else" block.

Usage

```
forelse(x, FUN, ALT_FUN = NULL)
```

Arguments

x	iterative R objects such as list, vector, etc.
FUN	function that applies to each x
ALT_FUN	function that takes no argument or other types of R object

Value

If any FUN returns anything other than NULL, then the function returns the first none NULL object. If all x fed to FUN return NULL, then this function returns ALT_FUN (if ALT_FUN is not a function) or the result of ALT_FUN().

Examples

```
# --------------------------- Basic Usage -----------------------------

# 1. ALT_FUN get executed because FUN returns NULL for all items in x
forelse(
  1:10,
  function(x){
    cat('The input is ', x, end = '\n')
    if( x > 10) return(x) else return(NULL)
  },
  function(){
    cat('ALT_FUN is executed!\n')
    'wow'
  }
)

# 2. FUN returns non-NULL object
forelse(
  1:10,
  function(x){
    cat('The input is ', x, end = '\n')
    if( x %% 2 == 0 ) return(x) else return(NULL)
```

```
  },
  'wow'
)

# --------------------------- Performance ------------------------------
FUN <- function(x){
  Sys.sleep(0.01)
  if( x %% 2 == 0 ) return(x) else return(NULL)
}

microbenchmark::microbenchmark({
  forelse(1:10, FUN, 'wow')
}, {
  y <- unlist(lapply(1:10, FUN))
  if(length(y)){
    y <- y[[1]]
  }else{
    y <- 'wow'
  }
}, {
  y <- NULL
  for(x in 1:10){ y <- FUN(x) }
  if(is.null(y)){ y <- 'wow' }
}, times = 3)
```

`getInputBinding` *Obtain registered input bindings*

Description

Obtain registered input bindings

Usage

```
getInputBinding(fname, pkg = NULL, envir = parent.frame())
```

Arguments

`fname`	input function name, character or quoted expression such as `'shiny::textInput'` or `numericInput`.
`pkg`	(optional), name of package
`envir`	environment to evaluate `fname` if `pkg` is not provided

Value

a list containing: 1. 'JavaScript' input binding name; 2. 'R' updating function name

Examples

```
library(dipsaus)

# Most recommended usage
getInputBinding('compoundInput2', pkg = 'dipsaus')

# Other usages
getInputBinding('shiny::textInput')


getInputBinding(shiny::textInput)

getInputBinding(compoundInput2, pkg = 'dipsaus')

# Bad usage, raise errors in some cases
## Not run:
## You need to library(shiny), or set envir=asNamespace('shiny'), or pkg='shiny'
getInputBinding('textInput')
getInputBinding(textInput) # also fails

## Always fails
getInputBinding('dipsaus::compoundInput2', pkg = 'dipsaus')

## End(Not run)
```

`get_cpu` *Defunct Functions in Package* **dipsaus** *The functions or variables listed here are no longer part of the package.*

Description

Defunct Functions in Package **dipsaus** The functions or variables listed here are no longer part of the package.

Usage

```
get_cpu()
```

`get_credential` *Generate a random password*

Description

Please note that this function is not meant to be used in production. It is not meant to be used for highly secured cryptographic purposes.

Usage

```
get_credential(
  master_password,
  method = c("get_or_create", "replace", "query"),
  service = NULL,
  special_chr = "~`! @#$%^&*()_-+={[}]|:;'<,>.?/",
  tokenfile = NULL,
  verbose = FALSE
)
```

Arguments

master_password	a master password that only you know, should have at least 8 characters
method	whether to query token map, or to create the password, choices are 'get_or_create' (default), 'replace', 'query'; see 'Details'
service	service name, must only contains letters, digits, equal sign, underscore, comma, dot, dash
special_chr	special characters allowed in the password
tokenfile	a file containing all the tokens. Warning: if you lose the token book, it is hard (not impossible, but impractical) to restore the passwords
verbose	whether to print out service names; default is false

Details

Please note that this function is not meant to be used in production or anything that requires high security level. This is most likely for my personal use since I am tired of storing the passwords on the cloud or having to buy the services.

The encryption adopts 'sha256' algorithm provided by digest function. To restore a password, you will need twp components: master_password, a token book (tokenfile). If any of them is missing, then the password is lost. Please store the token book properly (for example, in 'Dropbox' vault).

The token book could be shared. Anyone who do not have master password will be unlikely to restore the service password. Do not share the master password with anyone other than yourself.

By default, method='get_or_create' will try to retrieve existing tokens to generate password. If the token is missing, then a new token will be generated. The method='replace' will ignore existing tokens and directly create a new one.

Value

If method is 'query', returns token map; otherwise returns the password itself

See Also

digest

Examples

```
tokenfile <- tempfile()

# ---------- Create a password and store the tokens to token book ------
pass1 <- get_credential(
  master_password = "my password",
  service = "google.com:my_username",
  special_chr = "@#$%^&*",
  tokenfile = tokenfile
)
print(pass1)

# ---------- Query existing tokens ------
token_params <- get_credential(
  method = "query",
  tokenfile = tokenfile,
  verbose = TRUE
)

print(token_params)

# ---------- retrieve stored password ----------
pass2 <- get_credential(
  master_password = "my password",
  service = "google.com",
  tokenfile = tokenfile
)
identical(pass1, pass2)

# Using wrong master password
pass3 <- get_credential(
  master_password = "wrong password",
  service = "google.com",
  tokenfile = tokenfile
)
identical(pass1, pass3)

# ---------- Replace token ----------
# Existing token will be replaced with a new token
pass4 <- get_credential(
  master_password = "my password",
  method = "replace",
  service = "google.com",
  special_chr = "@#$%^&*",
  tokenfile = tokenfile
)
print(pass4)
identical(pass1, pass4)
```

get_dots *Get or check elements from dots '...'*

Description

Get information from '...' without evaluating the arguments.

Usage

```
get_dots(..name, ..default = NULL, ...)

missing_dots(envir = parent.frame())
```

Arguments

..name	character name of the argument
..default	R object to return if argument not found
...	dots that contains argument
envir	R environment

Value

missing_dots returns logical vector with lengths matching with dot lengths. get_dots returns value corresponding to the name.

Examples

```
# ------------------------ Basic Usage ---------------------------

# missing_dots(environment()) is a fixed usage

my_function <- function(...){
  missing_dots(environment())
}
my_function(,)

# get_dots
plot2 <- function(...){
  title = get_dots('main', 'There is no title', ...)
  plot(...)
  title
}

plot2(1:10)
plot2(1:10, main = 'Scatter Plot of 1:10')

# ------------------------ Comparisons ---------------------------
```

```
f1 <- function(...){ get_dots('x', ...) }
f2 <- function(...){ list(...)[['x']] }
delayedAssign('y', { cat('y is evaluated!') })

# y will not evaluate
f1(x = 1, y = y)

# y gets evaluated
f2(x = 1, y = y)

# -------------------- Decorator example --------------------------
ret_range <- function(which_range = 'y'){
  function(f){
    function(...){
      f(...)
      y_range <- range(get_dots(which_range, 0, ...))
      y_range
    }
  }
}
plot_ret_yrange <- plot %D% ret_range('y')
plot_ret_yrange(x = 1:10, y = rnorm(10))
```

`get_ip` *Get 'IP' address*

Description

Get 'IP' address

Usage

```
get_ip(get_public = NA)
```

Arguments

`get_public`	whether to get public 'IP'

Value

a list of 'IP' addresses

get_os *Detect the type of operating system*

Description

Detect the type of operating system

Usage

```
get_os()
```

Value

The type of current operating system: `'windows'`, `'darwin'`, `'linux'`, `'solaris'`, or otherwise `'unknown'`.

Examples

```
get_os()
```

get_ram *Get Memory Size*

Description

Get Memory Size

Usage

```
get_ram()
```

Details

The function `get_ram` only supports 'MacOS', 'Windows', and 'Linux'. 'Solaris' or other platforms will return `NA`. Here are the system commands used to detect memory limits:

'Windows' Uses command `'wmic.exe'` in the 'Windows' system folder. Notice this command-line tool might not exist on all 'Windows' machines. `get_ram` will return `NA` if it cannot locate the command-line tool.

'MacOS' Uses command `'sysctl'` located at `'/usr/sbin/'` or `'/sbin/'`. Alternatively, you can edit the environment variable `'PATH'` to include the command-line tools if `'sysctl'` is missing. `get_ram` will return `NA` if it cannot locate `'sysctl'`.

'Linux' Uses the file `'/proc/meminfo'`, possibly the first entry `'MemTotal'`. If the file is missing or entry `'MemTotal'` cannot be located, `get_ram` will return `NA`.

Value

System RAM in bytes, or NA if not supported.

Examples

```
get_ram()
```

---

graphic-devices *Create a group of named graphic devices*

Description

Create a group of named graphic devices

Usage

```
dev_create(..., env = parent.frame(), attributes = list())

get_dev_attr(which, dev = grDevices::dev.cur(), ifnotfound = NULL)
```

Arguments

...	named expressions to launch devices
env	environment to evaluate expressions
attributes	named list; names correspond to device names and values are attributes to set to the devices
which	which attribute to obtain
dev	which device to search for attributes
ifnotfound	value to return if attribute is not found

Value

A list of functions to query, control, and switch between devices

Examples

```
## Not run:  ## Unix-specific example

# Create multiple named devices, setting attributes to the second graph
devs <- dev_create(
  line = X11(), points = x11(),
  attributes = list(points = list(pch = 16))
)

# switch to device named "points"
```

```
devs$dev_which('points')

# Plot points, with pch given as preset
plot(1:10, pch = get_dev_attr(which = 'pch', ifnotfound = 1))

# switch to "line" device
devs$dev_switch('line')
plot(1:100, type='l')

# Create another group with conflict name
dev_another <- dev_create(line = X11())

# Query device name with 'line'
dev_another$dev_which('line')  # 4
devs$dev_which('line')  # 2, doesn't conflict with the new groups

dev.list()
# close one or more device
dev_another$dev_off('line')
dev.list()

# close all devices
devs$dev_off()
dev.list()


## End(Not run)
```

`handler_dipsaus_progress` *Progress-bar Handler*

Description

Handler for `progress2` to support `progressr::handlers`. See examples for detailed use case

Usage

```
handler_dipsaus_progress(
  title = getOption("dipsaus.progressr.title", "Progress"),
  intrusiveness = getOption("progressr.intrusiveness.gui", 1),
  target = if (is.null(shiny::getDefaultReactiveDomain())) "terminal" else "gui",
  enable = interactive() || shiny_is_running(),
  ...
)
```

Arguments

`title`	default title of `progress2`

intrusiveness	A non-negative scalar on how intrusive (disruptive) the reporter to the user
target	where progression updates are rendered
enable	whether the progress should be reported
...	passed to make_progression_handler

Examples

```
library(progressr)
library(shiny)
library(future)

## ------------------------------ Setup! --------------------------------
handlers(handler_dipsaus_progress())

# ------------------------------ A simple usage ------------------------
xs <- 1:5
handlers(handler_dipsaus_progress())
with_progress({
  p <- progressor(along = xs)
  y <- lapply(xs, function(x) {
    p(sprintf("x=%g", x))
    Sys.sleep(0.1)
    sqrt(x)
  })
})

# ------------------------ A future.apply case -------------------------
plan(sequential)
# test it yourself with plan(multisession)

handlers(handler_dipsaus_progress())
with_progress({
  p <- progressor(along = xs)
  y <- future.apply::future_lapply(xs, function(x) {
    p(sprintf("x=%g", x))
    Sys.sleep(0.1)
    sqrt(x)
  })
})

# ------------------------ A shiny case --------------------------------

ui <- fluidPage(
  actionButton('ok', 'Run Demo')
)

server <- function(input, output, session) {
  handlers(handler_dipsaus_progress())
  make_forked_clusters()

  observeEvent(input$ok, {
```

```
    with_progress({
      p <- progressor(along = 1:100)
      y <- future.apply::future_lapply(1:100, function(x) {
        p(sprintf("Input %d|Result %d", x, x+1))
        Sys.sleep(1)
        x+1
      })
    })
  })
}

if(interactive()){
  shinyApp(ui, server)
}
```

`html_asis` *Escape HTML strings*

Description

Escape HTML strings so that they will be displayed 'as-is' in websites.

Usage

```
html_asis(s, space = TRUE)
```

Arguments

s	characters
space	whether to also escape white space, default is true.

Value

An R string

Examples

```
ui <- flex_div(
  shiny::textInput('id', ' ', width = '100%',
                   value = 'Height not aligned'),
  actionButtonStyled('ok', 'Button1', width = '100%',),
  shiny::textInput('id2', html_asis(' '), width = '100%',
                   value = 'Heights aligned'),
  actionButtonStyled('ok2', 'Button2', width = '100%',),
  ncols = 2
)
```

```
if(interactive()){
  shiny::shinyApp(ui = shiny::fluidPage(shiny::fluidRow(ui)),
                  server = function(input, output, session){})
}
```

`html_class` *Combine, add, or remove 'HTML' classes*

Description

Combine 'HTML' classes to produce nice, clean 'HTML' class string via `combine_html_class`, or to remove a class via `remove_html_class`

Usage

```
combine_html_class(...)

remove_html_class(target, class)
```

Arguments

`...`	one or more characters, classes to combine; duplicated classes will be removed
`target`	characters, class list
`class`	one or more characters, classes to be removed from `target`

Value

A character string of new 'HTML' class

Examples

```
# Combine classes "a b c d e"
combine_html_class("a", "b  a", c("c", " d", "b"), list("e ", "a"))

# Remove class
remove_html_class("a b   c  e", c("b", "c "))
```

iapply *Apply each elements with index as second input*

Description

Apply function with an index variable as the second input.

Usage

```
iapply(X, FUN, ..., .method = c("sapply", "lapply", "vapply"))
```

Arguments

X	a vector (atomic or list)
FUN	the function to be applied to each element of X: see 'Details'.
...	passed to apply methods
.method	method to use, default is sapply

Details

FUN will be further passed to the apply methods. Unlike lapply, FUN is expected to have at least two arguments. The first argument is each element of X, the second argument is the index number of the element.

Value

a list or matrix depends on .method. See lapply

is_from_namespace *Check whether a function, environment comes from a namespace*

Description

A coarse way to find if a function comes from a package.

Usage

```
is_from_namespace(x, recursive = TRUE)
```

Arguments

x	function, environment, language (with environment attached)
recursive	whether to recursively search parent environments

Value

logical true if x or its environment is defined in a namespace; returns false if the object is atomic, or defined in/from global environment, or an empty environment.

Examples

```
is_from_namespace(baseenv())        # TRUE
is_from_namespace(utils::read.csv)  # TRUE

x <- function(){}
is_from_namespace(NULL)             # FALSE
is_from_namespace(x)                # FALSE
is_from_namespace(emptyenv())       # FALSE

# Let environment of `x` be base environment
# (exception case)
environment(x) <- baseenv()
is_from_namespace(x)        # TRUE
```

---

lapply_async2 *Apply, but in parallel*

Description

Apply, but in parallel

Usage

```
lapply_async2(
  x,
  FUN,
  FUN.args = list(),
  callback = NULL,
  plan = TRUE,
  future.chunk.size = NULL,
  future.seed = sample.int(1, n = 1e+05 - 1),
  ...
)
```

Arguments

x	vector, list
FUN	function to apply on each element of x
FUN.args	more arguments to feed into FUN

callback	function to run after each iteration
plan	logical, or character or future plan; see Details.
future.chunk.size, future.seed	see also future_lapply. If you want the callbacks to be called immediately after each loop, then set it to 1, which is not optimal but the only way right now.
...	passed to plan

Details

When plan is logical, FALSE means use current plan. If plan=TRUE, then it equals to plan='multicore'. For characters, plan can be 'multicore', 'callr', 'sequential', 'multisession', 'multiprocess', etc. Alternatively, you could pass future plan objects.

Value

same as with(FUN.args, lapply(x, function(el){eval(body(FUN))}))

See Also

make_forked_clusters

Examples

```
library(future)
plan(sequential)

# Use sequential plan
# 1. Change `plan` to 'multicore', 'multisession', or TRUE to enable
# multi-core, but still with progress information
# 2. Change plan=FALSE will use current future plan
res <- lapply_async2(100:200, function(x){
  return(x+1)
}, callback = function(e){
  sprintf('Input=%d', e)
}, plan = 'sequential')

# Disable callback message, then the function reduce to
# normal `future.apply::future_lapply`
res <- lapply_async2(100:200, function(x){
  return(x+1)
}, callback = NULL, plan = FALSE)

if(interactive()) {

  # PID are different, meaning executing in different sessions
  lapply_async2(1:4, function(x){
    Sys.getpid()
  })
}
```

`lapply_callr` *Apply function with* `rs_exec`

Description

Apply function with `rs_exec`

Usage

```
lapply_callr(
  x,
  fun,
  ...,
  .callback = NULL,
  .globals = list(),
  .ncores = future::availableCores(),
  .packages = attached_packages(),
  .focus_on_console = TRUE,
  .rs = FALSE,
  .quiet = FALSE,
  .name = "",
  .wait = TRUE
)
```

Arguments

`x`	vector or list
`fun`	function
`...`	passed to function, see `lapply`
`.callback`	a function takes zero, one, or two arguments and should return a string to show in the progress
`.globals`	a named list that `fun` relies on
`.ncores`	number of cores to use; only used when `.wait=TRUE`
`.packages`	packages to load
`.focus_on_console`	whether to focus on console once finished; is only used when `.rs` is true
`.rs`	whether to create 'RStudio' jobs; default is false
`.quiet`	whether to suppress progress message
`.name`	the name of progress and jobs
`.wait`	whether to wait for the results; default is true, which blocks the main session waiting for results.

Value

When .wait=TRUE, returns a list that should be, in most of the cases, identical to lapply; when .wait=FALSE, returns a function that collects results.

See Also

rs_exec

Examples

```
if(interactive()){

  lapply_callr(1:3, function(x, a){
    c(Sys.getpid(), a, x)
  }, a = 1)

  lapply_callr(1:30, function(x)
    {
      Sys.sleep(0.1)
      sprintf("a + x = %d", a + x)
    }, .globals = list(a = 1),
    .callback = I, .name = "Test")

}
```

---

list_to_fastmap2 *Copy elements to* fastmap2

Description

Copy elements to fastmap2

Usage

```
list_to_fastmap2(li, map = NULL)
```

Arguments

li	a list or an environment
map	NULL or a fastmap2 instance

Value

If map is not NULL, elements will be added to map and return map, otherwise create a new instance.

list_to_fastqueue2 *Copy elements to* fastqueue2

Description

Copy elements to fastqueue2

Usage

```
list_to_fastqueue2(li, queue = NULL)
```

Arguments

li	a list or an environment
queue	NULL or a fastqueue2 instance

Value

If map is not NULL, elements will be added to map and return map, otherwise create a new instance.

lock *Create or Unlock a Lock*

Description

A wrapper for 'synchronicity' package, but user can interrupt the lock procedure anytime, and don't have to worry about whether the lock exists or not.

Usage

```
dipsaus_lock(name, timeout = 10, exclusive = TRUE)

dipsaus_unlock(name, timeout = 10, exclusive = TRUE)

dipsaus_resetlocks(name)
```

Arguments

name	character, the locker's name, must be only letters and digits
timeout	numeric, seconds to wait for the locker to lock or unlock
exclusive	ignored

Value

Logical, whether the operation succeed.

Examples

```
# Clear existing locks
dipsaus::dipsaus_resetlocks()

# unlock to prepare for the example
dipsaus_unlock('testlocker', timeout = 0.01)

# Create a locker, return TRUE
lock_success = dipsaus_lock('testlocker')
if(lock_success){
  cat2('testlocker has been locked')
}

# test whether locker has been locked
lock_success = dipsaus_lock('testlocker', timeout = 0.01)
if(!lock_success){
  cat2('attempt to lock testlocker failed')
}

# unlock
dipsaus_unlock('testlocker', timeout = 0.01)

# clean up
dipsaus::dipsaus_resetlocks()
```

---

make_forked_clusters *Create forked clusters, but more than that*

Description

Creates forked clusters. If fails, then switch to alternative plan (default is "multisession").

Usage

```
make_forked_clusters(
  workers = future::availableCores(),
  on_failure = getOption("dipsaus.cluster.backup", "sequential"),
  clean = FALSE,
  ...
)
```

Arguments

workers	positive integer, number of cores to use
on_failure	alternative plan to use if failed. This is useful when forked process is not supported (like 'windows'); default is options("dipsaus.cluster.backup") or 'sequential'

clean	whether to reverse the plan on exit. This is useful when use make_forked_clusters inside of a function. See details and examples.
...	passing to future::plan

Details

This was original designed as a wrapper for future::plan(future::multicore, ...). Forked clusters are discouraged when running in 'RStudio' because some pointers in 'RStudio' might be incorrectly handled, causing fork-bombs. However, forked process also has big advantages over other parallel methods: there is no data transfer needed, hence its speed is very fast. Many external pointers can also be shared using forked process. Since version 1.14.0, unfortunately, forked 'multicore' is banned by future package by default, and you usually need to enable it manually. This function provides a simple way of enable it and plan the future at the same time.

On windows, forked process is not supported, under this situation, the plan fall back to sequential, which might not be what you want. In such case, this function provides an alternative strategy that allows you to plan. You could also always enable the alternative strategy by setting dipsaus.no.fork option to true.

The parameter clean allows you to automatically clean the plan. This function allows you to reverse back to previous plan automatically once your function exits. For example, users might have already set up their own plans, clean=TRUE allows you to set the plan back to those original plans once function exit. To use this feature, please make sure this function is called within another function, and you must collect results before exiting the outer function.

Value

Current future plan

See Also

lapply_async2

Examples

```
if(interactive()){

  # ------ Basic example
  library(future)
  library(dipsaus)

  # sequential
  plan("sequential")

  make_forked_clusters()
  plan()  # multicore, or multisession (on windows)

  Sys.getpid()  # current main session PID
  value(future({Sys.getpid()}))  # sub-process PID, evaluated as multicore
```

```
  # ------ When fork is not supported

  # reset to default single core strategy
  plan("sequential")

  # Disable forked process
  options("dipsaus.no.fork" = TRUE)
  options("dipsaus.cluster.backup" = "multisession")

  # Not fall back to multisession
  make_forked_clusters()
  plan()

  # ------ Auto-clean

  # reset plan
  plan("sequential")
  options("dipsaus.no.fork" = FALSE)
  options("dipsaus.cluster.backup" = "multisession")

  # simple case:
  my_func <- function(){
    make_forked_clusters(clean = TRUE)

    fs <- lapply(1:4, function(i){
      future({Sys.getpid()})
    })

    unlist(value(fs))
  }

  my_func()    # The PIDs are different, meaning they ran in other sessions
  plan()       # The plan is sequential, auto reversed strategy

  # ------ Auto-clean with lapply_async2
  my_plan <- plan()

  # lapply_async2 version of the previous task
  lapply_async2(1:4, function(i){
    Sys.getpid()
  })

  identical(plan(), my_plan)

}
```

---

map *Create R object map.*

Description

Provides five types of map that fit in different use cases.

Usage

```
session_map(map = fastmap::fastmap())

rds_map(path = tempfile())

text_map(path = tempfile())
```

Arguments

map	a fastmap::fastmap() list
path	directory path where map data should be stored

Details

There are five types of map implemented. They all inherit class AbstractMap. There are several differences in use case scenarios and they backend implementations.

session_map A session map takes a fastmap object. All objects are stored in current R session. This means you cannot access the map from other process nor parent process. The goal of this map is to share the data across different environments and to store global variables, as long as they share the same map object. If you are looking for maps that can be shared by different processes, check the rest map types. The closest map type is rds_map.

rds_map An 'RDS' map uses file system to store values. The values are stored separately in '.rds' files. Compared to session maps, 'RDS' map can be shared across different R process. It's recommended to store large files in rds_map. If the value is not large in RAM, text_map is recommended.

text_map A 'text' map uses file system to store values. Similar to rds_map, it can be stored across multiple processes as long as the maps share the same file directory. However, unlike rds_map, text_map the text_map can only store basic data values, namely atom data types. The supported types are: numeric, character, vector, list, matrix It's highly recommended to convert factors to characters. Do NOT use if the values are functions or environments. The recommended use case scenario is when the speed is not the major concern, and you want to preserve data with backward compatibility. Otherwise it's highly recommended to use rds_map.

Value

An R6 instance that inherits AbstractMap

Examples

```
# ----------------------Basic Usage ----------------------

# Define a path to your map.
path = tempfile()
```

```
map <- rds_map(path)

# Reset
map$reset()

# Check if the map is corrupted.
map$validate()

# You have not set any key-value pairs yet.
# Let's say two parallel processes (A and B) are sharing this map.
# Process A set values
map$keys()

# Start push
# set a normal message
map$set(key = 'a', value = 1)

# set a large object
map$set(key = 'b', value = rnorm(100000))

# set an object with hash of another object
map$set(key = 'c', value = 2, signature = list(
  parameter1 = 123,
  parameter2 = 124
))

# Check what's in the map from process B
mapB <- rds_map(path)
mapB$keys()
mapB$keys(include_signatures = TRUE)

# Number of key-values pairs in the map.
mapB$size()

# Check if key exists
mapB$has(c('1','a', 'c'))

# Check if key exists and signature also matches
mapB$has('c', signature = list(
  parameter1 = 123,
  parameter2 = 124
))

# Signature changed, then return FALSE. This is especially useful when
# value is really large and reading the value takes tons of time
mapB$has('c', signature = list(
  parameter1 = 1244444,
  parameter2 = 124
))

# Destroy the map's files altogether.
mapB$destroy()
```

```
## Not run:
  # Once destroyed, validate will raise error
  mapB$validate()

## End(Not run)
```

`mask_function2` *Mask a function with given variables*

Description

Modifies the default behavior of the function by adding one environment layer on top of input function. The masked variables are assigned directly to the environment.

Usage

```
mask_function2(f, ..., .list = list())
```

Arguments

`f`	any function
`...`, `.list`	name-value pairs to mask the function

Value

a masked function

Examples

```
a <- 123
f1 <- function(){
  a + 1
}
f1()   # 124

f2 <- mask_function2(f1, a = 1)
f2()   # a is masked with value 1, return 2

environment(f1)  # global env
environment(f2)  # masked env

env <- environment(f2)
identical(parent.env(env), environment(f1))  # true
env$a  # masked variables: a=1
```

match_calls *Recursively match calls and modify arguments*

Description

Recursively match calls and modify arguments

Usage

```
match_calls(
  call,
  recursive = TRUE,
  replace_args = list(),
  quoted = FALSE,
  envir = parent.frame(),
  ...
)
```

Arguments

call	an R expression
recursive	logical, recursively match calls, default is true
replace_args	named list of functions, see examples
quoted	logical, is call quoted
envir	which environment should call be evaluated
...	other parameters passing to match.call

Value

A nested call with all arguments matched

Examples

```
library(dipsaus); library(shiny)

# In shiny modules, we might want to add ns() to inputIds
# In this example, textInput(id) will become textInput(ns(id))
match_calls(lapply(1:20, function(i){
  textInput(paste('id_', i), paste('Label ', i))
}), replace_args = list(
  inputId = function(arg, call){ as.call(list(quote(ns), arg)) }
))
```

mean_se *Calculates mean and standard error of mean*

Description

Calculates mean and standard error of mean

Usage

```
mean_se(x, na.rm = FALSE, se_na_as_zero = na.rm)
```

Arguments

x	R numerical object
na.rm	whether to remove NA; default is false
se_na_as_zero	see na_as_zero in ste_mean

Value

A named vector containing the mean and standard error of mean (ste_mean).

See Also

ste_mean

Examples

```
# Mean should be near 0 (mean of standard normal)
# standard error of mean should be near 0.01
mean_se(rnorm(10000))
```

mem_limit2 *Get max RAM size This is an experimental function that is designed for non-windows systems*

Description

Get max RAM size This is an experimental function that is designed for non-windows systems

Usage

```
mem_limit2()
```

Value

a list of total free memory.

new_function2 *Create new function that supports 'quasi-quosure' syntax*

Description

Create new function that supports 'quasi-quosure' syntax

Usage

```
new_function2(
  args = alist(),
  body = {
 },
  env = parent.frame(),
  quote_type = c("unquoted", "quote", "quo"),
  quasi_env = parent.frame()
)
```

Arguments

args	named list of function formals
body	function body expression, supports 'quasi-quosure' syntax
env	declare environment of the function
quote_type	character, whether body is unquoted, quoted, or a 'quo' object (from 'rlang' package)
quasi_env	where the 'quasi-quosure' should be evaluated, default is parent environment

Details

An unquoted body expression will be quoted, all the expressions with 'quasi-quosure' like `!!var` will be evaluated and substituted with the value of `var`. For a 'quosure', `quo_squash` will be applied. A quoted expression will not be substitute, but will be expanded if any 'quasi-quosure' detected

`args` must be a `list` object, see `formals`. For arguments with no default values, or quoted defaults, use `alist`. An `arg=alist(a=)` will result in a function like `function(a){...}`. See examples for more details.

Value

a function

See Also

`new_function`

Examples

```
# ------------ standard usage ------------
x <- 1:10
f1 <- new_function2(alist(a=), { print(a + x) }, env = environment())
f1(0)

x <- 20:23
f1(0)  # result changed as x changed

# ------------ 'quasi-quosure' syntax ------------
x <- 1:10
f2 <- new_function2(alist(a=), { print(a + !!x) })
print(f2)

f2(0)

x <- 20:23
f2(0)  # result doesn't change as f2 doesn't depend on x anymore

# ------------ argument settings ------------

default <- 123

# default with values pre-specified
new_function2(list(a = default))   # function (a = 123){}

# default with values unevaluated
new_function2(list(a = quote(default)))   # function (a = default){}
new_function2(alist(a = default))

# missing default
new_function2(alist(a = ))    # function (a){}
```

---

no_op *Pipe-friendly no-operation function*

Description

returns the first input with side effects

Usage

```
no_op(.x, .expr, ..., .check_fun = TRUE)
```

Arguments

.x	any R object

.expr	R expression that produces side effects
..., .check_fun	see ‘details‘

Details

no_op is a pipe-friendly function that takes any values in, evaluate expressions but still returns input. This is very useful when you have the same input across multiple functions and you want to use pipes.

.expr is evaluated with a special object '.', you can use '.' to represent .x in .expr. For example, if .x=1:100, then plot(x=seq(0,1,length.out = 100), y=.) is equivalent to plot(x=seq(0,1,length.out = 100), y=1:100).

.check_fun checks whether .expr returns a function, if yes, then the function is called with argument .x and ...

Value

The value of .x

Examples

```
## 1. Basic usage

# Will print('a') and return 'a'
no_op('a', print)

# Will do nothing and return 'a' because .check_fun is false
no_op('a', print, .check_fun = FALSE)

# Will print('a') and return 'a'
no_op('a', print(.), .check_fun = FALSE)

## 2. Toy example
library(graphics)

par(mfrow = c(2,2))
x <- rnorm(100)

# hist and plot share the same input `rnorm(100)`

y <- x |>
  # .expr is a function, all ... are passed as other arguments
  no_op( hist, nclass = 10 ) |>
  no_op( plot, x = seq(0,1,length.out = 100) ) |>

  # Repeat the previous two plots, but with different syntax
  no_op({ hist(., nclass = 10) }) |>
  no_op({ plot(x = seq(0,1,length.out = 100), y = .) }) |>

  # The return statement is ignored
```

```
  no_op({ return(x + 1)})

# x is returned at the end

identical(x, y)   # TRUE
```

`package_installed` *Check if a package is installed*

Description

Check if a package is installed

Usage

```
package_installed(pkgs, all = FALSE)
```

Arguments

`pkgs`	vector of package names
`all`	only returns TRUE if all packages are installed. Default is FALSE.

Value

logical, if packages are installed or not. If `all=TRUE`, return a logical value of whether all packages a re installed.

Examples

```
# Check if package base and dipsaus are installed
package_installed(c('base', 'dipsaus'))

# Check if all required packages are installed
package_installed(c('base', 'dipsaus'), all = TRUE)
```

parse_svec *Parse Text Into Numeric Vectors*

Description

Parse Text Into Numeric Vectors

Usage

```
parse_svec(text, sep = ",", connect = "-:|", sort = FALSE, unique = TRUE)
```

Arguments

text	string with chunks, e.g. "1-10, 14, 16-20, 18-30" has 4 chunks
sep	default is ",", character used to separate chunks
connect	characters defining connection links for example "1:10" is the same as "1-10"
sort	sort the result
unique	extract unique elements

Value

a numeric vector. For example, "1-3" returns `c(1, 2, 3)`

See Also

deparse_svec

Examples

```
parse_svec('1-10, 13:15,14-20')
```

PersistContainer *Wrapper to cache key-value pairs and persist across sessions*

Description

This class is designed to persist arbitrary R objects locally and share across different sessions. The container consists two-level caches. The first one is session-based, meaning it's only valid under current R session and will be cleared once the session is shut down. The second is the persist-level map, which will persist to hard drive and shared across sessions. See `cache` method in 'details'.

Public Methods

`initialize(..., backend = rds_map)` The constructor. backend must inherit `AbstractMap`, `...` will be passed to `backend$new(...)`. To check available back-ends and their use cases, see `map`.

`reset(all = FALSE)` Reset container. If all is set to be true, then reset session-based and hard-drive-based, otherwise only reset session-based container.

`destroy(all = FALSE)` destroy the container. Only use it when you want to finalize the container in `reg.finalizer`.

`has(key, signature = NULL)` returns a list of true/false (logical) vectors indicating whether keys exist in the container, if signature is used when caching the key-value pairs, then it also checks whether signature matches. This is very important as even if the keys match but signature is wrong, the results will be false.

`remove(keys, all = TRUE)` Remove keys in the container. Default is to remove the keys in both levels. If `all=FALSE`, then only remove the key in current session

`cache(key, value, signature = NULL, replace = FALSE, persist = FALSE)` `key` and `signature` together form the unique identifier for the value. By default `signature` is none, but it's very useful when value if large, or `key` is not a string. `replace` indicates whether to force replace the key-value pairs even if the entry exists. If `persist` is true, then the value is stored in hard-disks, otherwise the value will be deleted once the session is closed.

See Also

`map`

Examples

```
container = PersistContainer$new(tempfile())

# Reset the container so that values are cleared
container$reset(all = TRUE)

# Store `1` to 'a' with signature 111 to a non-persist map
# returns 1
container$cache(key = 'a', value = 1, signature = 111, persist = FALSE)

# Replace 'a' with 3
# returns 3
container$cache(key = 'a', value = 3, signature = 111,
                persist = TRUE, replace = TRUE)

# check if 'a' exists with signature 111
container$has('a', signature = 111)    # TRUE
# When you only have 'a' but no signature
container$has('a')                     # TRUE
# check if 'a' exists with wrong signature 222
container$has('a', signature = 222)    # FALSE


# Store 'a' with 2 with same signature
```

```
# will fail and ignore the value (value will not be evaluated if signatured)
# Return 2 (Important! use cached values)
container$cache(key = 'a', value = {
  print(123)
  return(2)
}, signature = 111, replace = FALSE)

# When no signature is present
# If the key exists (no signature provided), return stored value
# returns 3
container$cache(key = 'a', value = 4)

# replace is TRUE (no signature provided), signature will be some default value
container$cache(key = 'a', value = 2, replace = TRUE)

# destroy the container to free disk space
container$destroy()
```

`print_directory_tree` *Print Directory Tree*

Description

Print Directory Tree

Usage

```
print_directory_tree(
  target,
  root = "~",
  child,
  dir_only = FALSE,
  collapse = NULL,
  ...
)
```

Arguments

`target`	target directory path, relative to `root`
`root`	root directory, default is `'~'`
`child`	child files in target; is missing, then list all files
`dir_only`	whether to display directory children only
`collapse`	whether to concatenate results as one single string
`...`	pass to `list.files` when list all files

Value

Characters, print-friendly directory tree.

progress2 *'Shiny' progress bar, but can run without reactive context*

Description

'Shiny' progress bar, but can run without reactive context

Usage

```
progress2(
  title,
  max = 1,
  ...,
  quiet = FALSE,
  session = shiny::getDefaultReactiveDomain(),
  shiny_auto_close = FALSE,
  log = NULL
)
```

Arguments

title	character, task description
max	maximum number of items in the queue
...	passed to shiny::Progress$new(...)
quiet	suppress console output, ignored in shiny context.
session	'shiny' session, default is current reactive domain
shiny_auto_close	logical, automatically close 'shiny' progress bar once current observer is over. Default is FALSE. If setting to TRUE, then it's equivalent to p <- progress2(...); on.exit({p$close()}, add = TRUE).
log	function when running locally, default is NULL, which redirects to cat2

Value

A list of functions:

`inc(detail, message = NULL, amount = 1, ...)` Increase progress bar by `amount` (default is 1).

`close()` Close the progress

`reset(detail = '', message = '', value = 0)` Reset the progress to `value` (default is 0), and reset information

`get_value()` Get current progress value

`is_closed()` Returns logical value if the progress is closed or not.

Examples

```
progress <- progress2('Task A', max = 2)
progress$inc('Detail 1')
progress$inc('Detail 2')
progress$close()

# Check if progress is closed
progress$is_closed()

# ------------------------------ Shiny Example ------------------------------
library(shiny)
library(dipsaus)

ui <- fluidPage(
  actionButtonStyled('do', 'Click Here', type = 'primary')
)

server <- function(input, output, session) {
  observeEvent(input$do, {
    updateActionButtonStyled(session, 'do', disabled = TRUE)
    progress <- progress2('Task A', max = 10, shiny_auto_close = TRUE)
    lapply(1:10, function(ii){
      progress$inc(sprintf('Detail %d', ii))
      Sys.sleep(0.2)
    })
    updateActionButtonStyled(session, 'do', disabled = FALSE)
  })
}

if(interactive()){
  shinyApp(ui, server)
}
```

---

registerInputBinding *Register customized input to enable support by compound input*

Description

Register customized input to enable support by compound input

Usage

```
registerInputBinding(
  fname,
  pkg,
  shiny_binding,
  update_function = NULL,
  quiet = FALSE
)
```

Arguments

fname	character, function name, such as "textInput"
pkg	character, package name, like "shiny"
shiny_binding	character, 'JavaScript' binding name.See examples
update_function	character, update function such as "shiny::textInput"
quiet	logical, whether to suppress warnings

Value

a list of binding functions, one is 'JavaScript' object key in Shiny.inputBindings, the other is 'shiny' update function in R end.

Examples

```
# register shiny textInput
registerInputBinding('textInput', 'shiny',
                     'shiny.textInput', 'shiny::updateTextInput')

# Register shiny actionLink
# In "Shiny.inputbindings", the binding name is "shiny.actionButtonInput",
# Shiny update function is "shiny::updateActionButton"
registerInputBinding('actionLink', 'shiny',
                     'shiny.actionButtonInput', 'shiny::updateActionButton')
```

---

restart_session *Restart R Session*

Description

Utilize 'RStudio' functions to restart, if running without 'RStudio', use package startup (not included in this library) instead.

Usage

```
restart_session()
```

rs_active_project *Get 'RStudio' active project*

Description

Get 'RStudio' active project

Usage

```
rs_active_project(...)
```

Arguments

...	passed to rs_avail

Value

If 'RStudio' is running and current project is not none, return project name, otherwise return NA

rs_avail *Verify 'RStudio' version*

Description

Verify 'RStudio' version

Usage

```
rs_avail(version_needed = "1.3", child_ok = FALSE, shiny_ok = FALSE)
```

Arguments

version_needed	minimum version required
child_ok	check if the current R process is a child process of the main RStudio session.
shiny_ok	if set false, then check if 'Shiny' is running, return false if shiny reactive domain is not NULL

Value

whether 'RStudio' is running and its version is above the required

See Also

isAvailable

rs_edit_file *Use 'RStudio' to open and edit files*

Description

Use 'RStudio' to open and edit files

Usage

```
rs_edit_file(path, create = TRUE)
```

Arguments

path	path to file
create	whether to create if path is not found; default is true

Value

Opens the file pointing to path to edit, and returns the path

rs_exec *Schedule a Background Job*

Description

Utilizes 'RStudio' job scheduler if correct environment is detected, otherwise call system command via Rscript

Usage

```
rs_exec(
  expr,
  name = "Untitled",
  quoted = FALSE,
  rs = TRUE,
  as_promise = FALSE,
  wait = FALSE,
  packages = NULL,
  focus_on_console = FALSE,
  ...,
  nested_ok = FALSE
)
```

Arguments

expr	R expression
name	used by 'RStudio' as name of the job
quoted	is expr quoted
rs	whether to use 'RStudio' by default
as_promise	whether to return as a promise object; default is no
wait	whether to wait for the result.
packages	packages to load in the sub-sessions
focus_on_console	whether to return back to console after creating jobs; useful when users want to focus on writing code; default is false. This feature works with 'RStudio' (>=1.4)
...	internally used
nested_ok	whether nested rs_exec is allowed; default is false; Set to true to allow nested parallel code, but use at your own risk.

Details

'RStudio' provides interfaces jobRunScript to schedule background jobs. However, this functionality only applies using 'RStudio' IDE. When launching R from other places such as terminals, the job scheduler usually result in errors. In this case, the alternative is to call system command via Rscript

The expression expr will run a clean environment. Therefore R objects created outside of the context will be inaccessible from within the child environment, and packages except for base packages will not be loaded.

There is a small difference when running within and without 'RStudio'. When running via Rscript, the environment will run under vanilla argument, which means no load, no start-up code. If you have start-up code stored at ~/.Rprofile, the start-up code will be ignored. When running within 'RStudio', the start-up code will be executed. As of rstudioapi version 0.11, there is no 'vanilla' option. This feature is subject to change in the future.

Value

If wait=TRUE, returns evaluation results of expr, otherwise a function that can track the state of job.

Examples

```
if(interactive()){
  h <- rs_exec(
    {
      Sys.sleep(2)
      print(Sys.getpid())
    },
    wait = FALSE, name = 'Test',
    focus_on_console = TRUE
  )
```

```
  code <- h()
  print(code)

  # wait 3 seconds
  Sys.sleep(3)
  code <- h()
  attributes(code)
}
```

`rs_focus_console` *Focus on 'RStudio' Console*

Description

Focus on coding; works with 'RStudio' (>=1.4)

Usage

```
rs_focus_console(wait = 0.5)
```

Arguments

`wait`	wait in seconds before sending command; if too soon, then 'RStudio' might not be able to react.

Value

None

`rs_save_all` *Save all documents in 'RStudio'*

Description

Perform "safe" save-all action with backward compatibility: check whether 'RStudio' is running and whether `rstudioapi` has function `documentSaveAll`.

Usage

```
rs_save_all()
```

rs_select_path *Use 'RStudio' to Select a Path on the Server*

Description

Use 'RStudio' to Select a Path on the Server

Usage

```
rs_select_path(is_directory = TRUE)
```

Arguments

is_directory	whether the path should be a directory

Value

Raise error if `rs_avail` fails, otherwise returns the selected path

rs_set_repos *Add secondary 'CRAN'-like repository to the 'RStudio' settings*

Description

Add self-hosted repository, such as 'drat', 'r-universe' to 'RStudio' preference. Please restart 'RStudio' to take changes into effect.

Usage

```
rs_set_repos(name, url, add = TRUE)
```

Arguments

name	repository name, must be unique and readable
url	the website address of the repository, starting with schemes such as `'https'`.
add	whether to add to existing repository; default is true

Details

'RStudio' allows to add secondary 'CRAN'-like repository to its preference, such that users can add on-going self-hosted developing repositories (such as package `'drat'`, or 'r-universe'). These repositories will be set automatically when running `install.packages`.

Value

a list of settings.

rs_viewer *Get 'RStudio' Viewer, or Return Default*

Description

Get 'RStudio' Viewer, or Return Default

Usage

```
rs_viewer(
  ...,
  default = TRUE,
  version_needed = "1.3",
  child_ok = FALSE,
  shiny_ok = FALSE
)
```

Arguments

`...`	passed to `viewer`
`default`	if `rs_avail` fails, the value to return. Default is `TRUE`
`version_needed, child_ok, shiny_ok`	passed to `rs_avail`

Value

If `viewer` can be called and 'RStudio' is running, then launch 'RStudio' internal viewer. Otherwise if `default` is a function such as `browseURL`, then call the function with given arguments. If `default` is not a function, return `default`

screenshot *Take a screenshot in shiny apps*

Description

Take a screenshot of the whole page and save encoded `DataURI` that can be accessed via `input[[inputId]]`.

Usage

```
screenshot(inputId, session = shiny::getDefaultReactiveDomain())
```

Arguments

`inputId`	the input id where the screenshot should be
`session`	shiny session

Value

None. However, the screenshot results can be accessed from shiny input

Examples

```
library(shiny)
library(dipsaus)
ui <- fluidPage(
  tagList(
    shiny::singleton(shiny::tags$head(
      shiny::tags$link(rel="stylesheet", type="text/css", href="dipsaus/dipsaus.css"),
      shiny::tags$script(src="dipsaus/dipsaus-dipterix-lib.js")
    ))
  ),
  actionButtonStyled('do', 'Take Screenshot'),
  compoundInput2('group', label = 'Group', components = list(
    textInput('txt', 'Enter something here')
  ))
)

server <- function(input, output, session) {
  observeEvent(input$do, {
    screenshot('screeshot_result')
  })
  observeEvent(input$screeshot_result, {
    showModal(modalDialog(
      tags$img(src = input$screeshot_result, width = '100%')
    ))
  })
}

if(interactive()){
  shinyApp(ui, server)
}
```

---

session_uuid *Provides Unique Session ID According to Current R Session*

Description

Provides Unique Session ID According to Current R Session

Usage

```
session_uuid(pid = Sys.getpid(), attributes = FALSE)
```

Arguments

pid	R session process ID, default is Sys.getpid()
attributes	whether to append data used to calculate ID as attributes, default is false

Value

Character string

---

set_shiny_input *Set Shiny Input*

Description

Shiny 'input' object is read-only reactive list. When try to assign values to input, errors usually occur. This method provides several work-around to set values to input. Please use along with use_shiny_dipsaus.

Usage

```
set_shiny_input(
  session = shiny::getDefaultReactiveDomain(),
  inputId,
  value,
  priority = c("event", "deferred", "immediate"),
  method = c("proxy", "serialize", "value", "expression"),
  quoted = TRUE
)
```

Arguments

session	shiny session, see shiny domains
inputId	character, input ID
value	the value to assign
priority	characters, options are "event", "deferred", and "immediate". "event" and "immediate" are similar, they always fire changes. "deferred" fire signals to other reactive/observers only when the input value has been changed
method	characters, options are "proxy", "serialize", "value", "expression". "proxy" is recommended, other methods are experimental.
quoted	is value quoted? Only used when method is "expression"

Examples

```
library(shiny)
library(dipsaus)
ui <- fluidPage(
  # Register widgets
  use_shiny_dipsaus(),
  actionButton('run', 'Set Input'),
  verbatimTextOutput('input_value')
)

server <- function(input, output, session) {
  start = Sys.time()

  output$input_value <- renderPrint({

    now <- input$key
    now %?<-% start
    cat('This app has been opened for ',
        difftime(now, start, units = 'sec'), ' seconds')
  })

  observeEvent(input$run, {
    # setting input$key to Sys.time()
    set_shiny_input(session, 'key', Sys.time())
  })
}

if(interactive()){
  shinyApp(ui, server)
}
```

---

sexp_type2 *Get Internal Storage Type*

Description

Get internal (C) data types; See https://cran.r-project.org/doc/manuals/r-release/R-ints.pdf Page 1 for more different SEXPTYPEs.

Usage

```
sexp_type2(x)

## S3 method for class 'sexp_type2'
as.character(x, ...)

## S3 method for class 'sexp_type2'
print(x, ...)
```

Arguments

x	any **R** object
...	ignored

Value

An integer of class "sexp_type2"

See Also

storage.mode

Examples

```
# 1 vs 1L

# Integer case
sexp_type2(1L)

# double
sexp_type2(1)

# Built-in function
sexp_type2(`+`)

# normal functions
sexp_type2(sexp_type2)

# symbols (quoted names)
sexp_type2(quote(`+`))

# Calls (quoted expressions)
sexp_type2(quote({`+`}))
```

---

shared_finalizer *Create Shared Finalization to Avoid Over Garbage Collection*

Description

Generates a function to be passed to reg.finalizer

Usage

```
shared_finalizer(x, key, fin, onexit = FALSE, ...)

## Default S3 method:
shared_finalizer(x, key, fin, onexit = FALSE, ...)

## S3 method for class 'R6'
shared_finalizer(x, key, fin, onexit = TRUE, ...)

## S3 method for class 'fastmap'
shared_finalizer(x, key, fin, onexit = FALSE, ...)

## S3 method for class 'fastmap2'
shared_finalizer(x, key, fin, onexit = FALSE, ...)
```

Arguments

x	object to finalize
key	characters that should be identical if finalization method is to be shared
fin	Shared finalization: function to call on finalization; see reg.finalizer. See details.
onexit	logical: should the finalization be run if the object is still uncollected at the end of the R session? See reg.finalizer
...	passed to other methods

Details

The main purpose of this function is to allow multiple objects that point to a same source (say a temporary file) to perform clean up when all the objects are garbage collected.

Base function reg.finalizer provides finalization to to garbage collect single R environment. However, when multiple environments share the same file, finalizing one single environment will result in removing the file so that all the other environment lose the reference. (See example "Native reg.finalizer fails example")

The argument of fin varies according to different types of x. For environments, fin contains and only contains one parameter, which is the environment itself. This is the same as reg.finalizer. For R6 classes, fin is ignored if class has "shared_finalize" method defined. For fastmap or fastmap2 instances, fin accepts no argument.

Examples

```
# ------------ Environment example ------------
file_exists <- TRUE
clear_files <- function(e){
  print('Clean some shared files')
  # do something to remove files
  file_exists <<- FALSE
}
```

```
# e1, e2 both require file existence
e1 <- new.env()
e1$valid <- function(){ file_exists }
e2 <- new.env()
e2$valid <- function(){ file_exists }

e1$valid(); e2$valid()

# we don't want to remove files when either e1,e2 gets
# garbage collected, however, we want to run `clear_files`
# when system garbage collecting *both* e1 and e2

# Make sure `key`s are identical
shared_finalizer(e1, 'cleanXXXfiles', clear_files)
shared_finalizer(e2, 'cleanXXXfiles', clear_files)

# Now remove e1, files are not cleaned, and e2 is still valid
rm(e1); invisible(gc(verbose = FALSE))
e2$valid()  # TRUE
file_exists # TRUE

# remove both e1 and e2, and file gets removed
rm(e2); invisible(gc(verbose = FALSE))
file_exists  # FALSE

# ------------ R6 example ------------

cls <- R6::R6Class(
  classname = '...demo...',
  cloneable = TRUE,
  private = list(
    finalize = function(){
      cat('Finalize private resource\n')
    }
  ),
  public = list(
    file_path = character(0),
    shared_finalize = function(){
      cat('Finalize shared resource - ', self$file_path, '\n')
    },
    initialize = function(file_path){
      self$file_path = file_path
      shared_finalizer(self, key = self$file_path)
    }
  )
)
e1 <- cls$new('file1')
rm(e1); invisible(gc(verbose = FALSE))

e1 <- cls$new('file2')

# A copy of e1
```

```
e2 <- e1$clone()
# unfortunately, we have to manually register
shared_finalizer(e2, key = e2$file_path)

# Remove e1, gc only free private resource
rm(e1); invisible(gc(verbose = FALSE))

# remove e1 and e2, run shared finalize
rm(e2); invisible(gc(verbose = FALSE))

# ------------ fastmap/fastmap2 example -----------

# No formals needed for fastmap/fastmap2
fin <- function(){
  cat('Finalizer is called\n')
}
# single reference case
e1 <- dipsaus::fastmap2()
shared_finalizer(e1, 'fin-fastmap2', fin = fin)
invisible(gc(verbose = FALSE)) # Not triggered
rm(e1); invisible(gc(verbose = FALSE)) # triggered

# multiple reference case
e1 <- dipsaus::fastmap2()
e2 <- dipsaus::fastmap2()
shared_finalizer(e1, 'fin-fastmap2', fin = fin)
shared_finalizer(e2, 'fin-fastmap2', fin = fin)

rm(e1); invisible(gc(verbose = FALSE)) # Not triggered
rm(e2); invisible(gc(verbose = FALSE)) # triggered

# ------------ Native reg.finalizer fails example ------------

# This example shows a failure case using base::reg.finalizer

file_exists <- TRUE
clear_files <- function(e){
  print('Clean some shared files')
  # do something to remove files
  file_exists <<- FALSE
}

# e1, e2 both require file existence
e1 <- new.env()
e1$valid <- function(){ file_exists }
e2 <- new.env()
e2$valid <- function(){ file_exists }

reg.finalizer(e1, clear_files)
reg.finalizer(e2, clear_files)
gc()
file_exists
```

```
# removing e1 will invalidate e2
rm(e1); gc()
e2$valid()    # FALSE

# Clean-ups
rm(e2); gc()
```

`shift_array` *Shift Array by Index*

Description

Re-arrange arrays in parallel

Usage

```
shift_array(x, shift_idx, shift_by, shift_amount)
```

Arguments

`x`	array, must have at least matrix
`shift_idx`	which index is to be shifted
`shift_by`	which dimension decides `shift_amount`
`shift_amount`	shift amount along `shift_idx`

Details

A simple use-case for this function is to think of a matrix where each row is a signal and columns stand for time. The objective is to align (time-lock) each signal according to certain events. For each signal, we want to shift the time points by certain amount.

In this case, the shift amount is defined by `shift_amount`, whose length equals to number of signals. `shift_idx=2` as we want to shift time points (column, the second dimension) for each signal. `shift_by=1` because the shift amount is depend on the signal number.

Examples

```
x <- matrix(1:10, nrow = 2, byrow = TRUE)
z <- shift_array(x, 2, 1, c(1,2))

y <- NA * x
y[1,1:4] = x[1,2:5]
y[2,1:3] = x[2,3:5]

# Check if z ang y are the same
z - y

# array case
```

```
# x is Trial x Frequency x Time
x <- array(1:27, c(3,3,3))

# Shift time for each trial, amount is 1, -1, 0
shift_amount <- c(1,-1,0)
z <- shift_array(x, 3, 1, shift_amount)

if(interactive()){

par(mfrow = c(3, 2))
for( ii in 1:3 ){
  image(t(x[ii, ,]), ylab = 'Frequency', xlab = 'Time',
        main = paste('Trial', ii))
  image(t(z[ii, ,]), ylab = 'Frequency', xlab = 'Time',
        main = paste('Shifted amount:', shift_amount[ii]))
}

}
```

`shiny_alert2` *Simple shiny alert that uses 'JavaScript' promises*

Description

Simple shiny alert that uses 'JavaScript' promises

Usage

```
shiny_alert2(
  title = "Alert",
  text = "",
  icon = c("info", "warning", "success", "error"),
  danger_mode = FALSE,
  auto_close = TRUE,
  buttons = NULL,
  on_close = NULL,
  session = shiny::getDefaultReactiveDomain()
)

close_alert2(session = shiny::getDefaultReactiveDomain())
```

Arguments

`title`	title of the alert
`text`	alert body text (pure text)
`icon`	which icon to display, choices are `'info'`, `'success'` `'warning'`, and `'error'`

danger_mode	true or false; if true, then the confirm button turns red and the default focus is set on the cancel button instead. To enable danger mode, buttons must be TRUE as well
auto_close	whether to close automatically when clicking outside of the alert
buttons	logical value or a named list, or characters. If logical, it indicates whether buttons should be displayed or not; for named list, the names will be the button text, see example; for characters, the characters will be the button text and value
on_close	NULL or a function that takes in one argument. If function is passed in, then it will be executed when users close the alert
session	shiny session, see domains

Value

a temporary input ID, currently not useful

Examples

```
library(shiny)
library(dipsaus)
ui <- fluidPage(
  use_shiny_dipsaus(),
  actionButtonStyled('btn', 'btn')
)

server <- function(input, output, session) {
  observeEvent(input$btn, {
    shiny_alert2(
      on_close = function(value) {
        cat("Modal closed!\n")
        print(value)
      },
      title = "Title",
      text = "message",
      icon = "success",
      auto_close = FALSE,
      buttons = list("cancel" = TRUE,
                     "YES!" = list(value = 1))
    )
  })
}

if(interactive()){
  shinyApp(ui, server, options = list(launch.browser = TRUE))
}
```

shiny_is_running *Detect whether 'Shiny' is running*

Description

Detect whether 'Shiny' is running

Usage

```
shiny_is_running()
```

Value

logical, true if current shiny context is active

ste_mean *Standard error of mean*

Description

Ported from 'rutabaga' package, calculates standard error of mean. The sample size is determined by number of none-NA numbers by default

Usage

```
ste_mean(x, na.rm = FALSE, na_as_zero = na.rm, ...)

## Default S3 method:
ste_mean(x, na.rm = FALSE, na_as_zero = na.rm, ...)
```

Arguments

x	R object
na.rm	whether to remove NA; default is false
na_as_zero	whether convert NA to zero
...	passed to other methods

Value

A numerical number that is the standard error of the mean

See Also

mean_se

Examples

```
x <- rnorm(100)

ste_mean(x)

# internal implementation
identical(ste_mean(x), sd(x) / sqrt(100))
```

`sumsquared` *Fast Calculation of Sum-squared for Large Matrices/Vectors*

Description

Calculate `sum(x^2)`, but faster when the number of elements exceeds 1000.

Arguments

`x`	double, integer, or logical vector/matrix

Value

A numerical scalar

Examples

```
x <- rnorm(10000)
sumsquared(x)

# Compare speed
microbenchmark::microbenchmark(
  cpp = {sumsquared(x)},
  r = {sum(x^2)}
)
```

`sync_shiny_inputs` *Synchronize Shiny Inputs*

Description

Synchronize Shiny Inputs

Usage

```
sync_shiny_inputs(
  input,
  session,
  inputIds,
  uniform = rep("I", length(inputIds)),
  updates,
  snap = 250,
  ignoreNULL = TRUE,
  ignoreInit = FALSE
)
```

Arguments

input, session	shiny reactive objects
inputIds	input ids to be synchronized
uniform	functions, equaling to length of inputIds, converting inputs to a uniform values
updates	functions, equaling to length of inputIds, updating input values
snap	numeric, milliseconds to defer the changes
ignoreNULL, ignoreInit	passed to bindEvent

Value

none.

Examples

```
library(shiny)

ui <- fluidPage(
  textInput('a', 'a', value = 'a'),
  sliderInput('b', 'b', value = 1, min = 0, max = 1000)
)

server <- function(input, output, session) {
  sync_shiny_inputs(input, session, inputIds = c('a', 'b'), uniform = list(
    function(a){as.numeric(a)},
    'I'
  ), updates = list(
    function(a){updateTextInput(session, 'a', value = a)},
    function(b){updateSliderInput(session, 'b', value = b)}
  ))

}

if( interactive() ){
  shinyApp(ui, server)
}
```

test_farg *Test whether function has certain arguments*

Description

Test whether function has certain arguments

Usage

```
test_farg(fun, arg, dots = TRUE)
```

Arguments

fun	function
arg	characters of function arguments
dots	whether fun's dots (...) counts

Examples

```
a <- function(n = 1){}

# Test whether `a` has argument called 'b'
test_farg(a, 'b')

# Test whether `a` has argument called 'b' and 'n'
test_farg(a, c('b', 'n'))

# `a` now has dots
a <- function(n = 1, ...){}

# 'b' could goes to dots and a(b=...) is still valid
test_farg(a, 'b')

# strict match, dots doesn't count
test_farg(a, 'b', dots = FALSE)
```

time_delta *Calculate time difference and return a number*

Description

Calculate time difference and return a number

Usage

```
time_delta(t1, t2, units = "secs")
```

Arguments

t1	time start
t2	time end
units	character, choices are 'secs', 'mins', 'hours', and 'days'

Value

numeric difference of time in units specified

Examples

```
a = Sys.time()
Sys.sleep(0.3)
b = Sys.time()

time_delta(a, b) # In seconds, around 0.3
time_delta(a, b, 'mins') # in minutes, around 0.005
```

---

to_datauri *Convert file to 'base64' format*

Description

Convert file to 'base64' format

Usage

```
to_datauri(file, mime = "")
```

Arguments

file	file path
mime	'mime' type, default is blank

Value

a 'base64' data string looks like 'data:;base64,AEF6986...'

to_ram_size *Convert bytes to KB, MB, GB,...*

Description

Convert bytes to KB, MB, GB,...

Usage

```
to_ram_size(s, kb_to_b = 1000)
```

Arguments

s	size
kb_to_b	how many bytes counts one KB, 1000 by default

Value

numeric equaling to s but formatted

updateActionButtonStyled *Update styled action button*

Description

Update styled action button

Usage

```
updateActionButtonStyled(
  session,
  inputId,
  label = NULL,
  icon = NULL,
  type = NULL,
  disabled = NULL,
  ...
)
```

Arguments

session, inputId, label, icon	passed to shiny::updateActionButton
type	button type to update
disabled	whether to disable the button
...	ignored

Value

none

See Also

actionButtonStyled for how to define the button.

---

updateCompoundInput2 *Update compound inputs*

Description

Update compound inputs

Usage

```
updateCompoundInput2(
  session,
  inputId,
  value = NULL,
  ncomp = NULL,
  initialization = NULL,
  ...
)
```

Arguments

session	shiny session or session proxy
inputId	character see compoundInput2
value	list of lists, see compoundInput2 or examples
ncomp	integer, non-negative number of groups to update, NULL to remain unchanged
initialization, ...	named list of other updates

Value

none

See Also

compoundInput2 for how to define components.

Examples

```
## Not run:
library(shiny); library(dipsaus)

## UI side
compoundInput2(
  'input_id', 'Group',
    div(
    textInput('text', 'Text Label'),
    sliderInput('sli', 'Slider Selector', value = 0, min = 1, max = 1)
  ),
  label_color = 1:10,
  value = list(
    list(text = '1'),  # Set text first group to be "1"
    '',                # no settings for second group
    list(sli = 0.2)    # sli = 0.2 for the third group
  ))

## server side:
updateCompoundInput2(session, 'inputid',
                     # Change the first 3 groups
                     value = lapply(1:3, function(ii){
                       list(sli = runif(1))
                     }),
                     # Change text label for all groups
                     initialization = list(
                       text = list(label = as.character(Sys.time()))
                     ))

## End(Not run)
```

`update_fastmap2` *Migrate a* `fastmap2` *object to a new one*

Description

Migrate a `fastmap2` object to a new one

Usage

```
update_fastmap2(from, to, override = TRUE)
```

Arguments

`from`, `to`	`fastmap2` object
`override`	whether to override keys in `to` if they exist

Value

Map to

See Also

fastmap2

---

use_shiny_dipsaus *Set up shiny plugins*

Description

This function must be called from a Shiny app's UI in order for some widgets to work.

Usage

```
use_shiny_dipsaus(x)
```

Arguments

x	'HTML' tags

%OF% *Get an element with condition that it must be from a list or vector*

Description

Get an element with condition that it must be from a list or vector

Usage

```
lhs %OF% rhs
```

Arguments

lhs	the element of candidate
rhs	the constraint

Value

Returns an element of length one that will be from rhs

Examples

```
# C is from LETTERS, therefore returns `C`
"C" %OF% LETTERS


# `lhs` is not from `rhs`, hence return the first element of LETTERS
'9' %OF% LETTERS
NULL %OF% LETTERS

# When there are multiple elements from `lhs`, select the first that
# matches the constraint
c('9', "D", "V") %OF% LETTERS
```

`%=>%` *A JavaScript style of creating functions*

Description

A JavaScript style of creating functions

Usage

```
args %=>% expr
```

Arguments

`args`	function arguments: see `formals`
`expr`	R expression that forms the body of functions: see `body`

Value

A function that takes `args` as parameters and `expr` as the function body

Examples

```
# Formal arguments
c(a) %=>% {
  print(a)
}

# Informal arguments
list(a=) %=>% {
  print(a)
}

# Multiple inputs
c(a, b = 2, ...) %=>% {
  print(c(a, b, ...))
```

```
}

# ----- JavaScript style of forEach -----
# ### Equivalent JavaScript Code:
# LETTERS.forEach((el, ii) => {
#   console.log('The index of letter ' + el + ' in "x" is: ' + ii);
# });

iapply(LETTERS, c(el, ii) %=>% {
  cat2('The index of letter ', el, ' in ', sQuote('x'), ' is: ', ii)
}) -> results
```

`%?<-%` *Left-hand side checked assignment Provides a way to assign default values to variables. If the statement* ‘`lhs`‘ *is invalid or* `NULL`*, this function will try to assign* `value`*, otherwise nothing happens.*

Description

Left-hand side checked assignment Provides a way to assign default values to variables. If the statement ‘`lhs`‘ is invalid or `NULL`, this function will try to assign `value`, otherwise nothing happens.

Usage

```
lhs %?<-% value
```

Arguments

`lhs`	an object to check or assign
`value`	value to be assigned if lhs is NULL

Value

Assign value on the right-hand side to the left-hand side if `lhs` does not exist or is `NULL`

Examples

```
# Prepare, remove aaa if exists
if(exists('aaa', envir = globalenv(), inherits = FALSE)){
  rm(aaa, envir = globalenv())
}

# Assign
aaa %?<-% 1; print(aaa)

# However, if assigned, nothing happens
aaa = 1;
aaa %?<-% 2;
print(aaa)
```

```
# in a list
a = list()
a$e %?<-% 1; print(a$e)
a$e %?<-% 2; print(a$e)
```

`%+-%` *Plus-minus operator*

Description

Plus-minus operator

Usage

```
a %+-% b
```

Arguments

a, b	numeric vectors, matrices or arrays

Value

a +/- b, the dimension depends on a+b. If a+b is a scalar, returns a vector of two; in the case of vector, returns a matrix; all other cases will return an array with the last dimension equal to 2.

Examples

```
# scalar
1 %+-% 2   # -1, 3

# vector input
c(1,2,3) %+-% 2   # matrix

# matrix input
matrix(1:9, 3) %+-% 2   # 3x3x2 array
```

`%<-?%` *Right-hand side checked assignment Provides a way to avoid assignment to the left-hand side. If the statement ‘*`value`*‘ is invalid or* `NULL`*, this function will not assign values and nothing happens.*

Description

Right-hand side checked assignment Provides a way to avoid assignment to the left-hand side. If the statement ‘`value`‘ is invalid or `NULL`, this function will not assign values and nothing happens.

Usage

```
lhs %<-?% value
```

Arguments

`lhs`	an object to be assigned to
`value`	value to be checked

Value

Assign value on the right-hand side to the left-hand side if `value` does exists and is not `NULL`

Examples

```
# Prepare, remove aaa if exists
if(exists('aaa', envir = globalenv(), inherits = FALSE)){
  rm(aaa, envir = globalenv())
}

# aaa will not be assigned. run `print(aaa)` will raise error
aaa %<-?% NULL

# Assign
aaa %<-?% 1
print(aaa)

# in a list
a = list()
a$e %<-?% bbb; print(a$e)
a$e %<-?% 2; print(a$e)
```

Index