

Package: GiottoUtils (via r-universe)

October 22, 2026

Title Giotto Suite Utilities

Version 0.2.9

Description Utility functions and imports for Giotto Suite.

License MIT + file LICENSE

Encoding UTF-8

URL <https://giotto-suite.github.io/GiottoUtils/>

BugReports <https://github.com/giotto-suite/Giotto/issues>

Roxygen list(markdown = TRUE)

Depends R (>= 4.5.0)

Imports checkmate, data.table, gtools, jsonlite, lifecycle, magrittr,
methods, progressr, rlang, R.utils, stats, tools

Suggests arrow, BiocParallel, BiocStyle, dplyr, future, future.apply,
Matrix, knitr, parallel, remotes, RColorBrewer, reticulate,
testthat (>= 3.0.0), covr, rmarkdown

Collate 'DT_helpers.R' 'assert.R' 'color_pals.R' 'depth.R'
'eval_handle.R' 'file_read.R' 'symbols.R' 'format.R'
'globals.R' 'gmatch.R' 'infix.R' 'install.R' 'json.R'
'lapply_flex.R' 'lifecycle.R' 'logging.R' 'mixedsort.R'
'options.R' 'pipe.R' 'pkg_check.R' 'prev_call.R' 'progress.R'
'py_utils.R' 'reshape.R' 'seed.R' 'string.R' 'system_info.R'
'test.R' 'with.R' 'zzz.R'

VignetteBuilder knitr

Config/testthat/edition 3

biocViews Software, Technology, Spatial

Config/roxygen2/version 8.0.0

Repository <https://giotto-suite.r-universe.dev>

Date/Publication 2026-10-22 16:05:44 UTC

RemoteUrl <https://github.com/giotto-suite/GiottoUtils>

RemoteRef v0.2.9

RemoteSha ca3350c1e4a694736b1bc072cf9912f28046da84

Contents

degrees	3
deprecate_param	3
dir_manifest	5
dt_dcast_string	6
dt_remove_na	7
dt_set_row_order	7
dt_sort_combine_two_columns	8
dt_to_matrix	9
file_extension	9
getDistinctColors	10
getMonochromeColors	11
getRainbowColors	12
giotto_bpparam	12
giottoNewLog	13
giottoReadLog	13
gstop	14
handle_errors	15
handle_warnings	16
init_option	17
keep_awake	17
lifecycle_badge	18
local_seed	19
log_write	20
melt_matrix	21
new_github_ver_avail	21
pbar	22
prev_call	24
print_list	25
py_active_env	26
random_seed	26
require_depth	27
scipy_sparse	27
set_if	28
str_abbreviate	29
str_convenience	30
str_locate2	31
suite_install	32
suite_packages	34
time_format	34
vmsg	35
with	36
with_pbar	38
wrap_msg	39
wrap_txt	40

Index

42

degrees	<i>Radian/degree conversions</i>
---------	----------------------------------

Description

Convert radians to degrees and vice versa

Usage

radians(deg)

degrees(rad)

Arguments

deg	degrees
rad	radians

Value

numeric

Functions

- radians(): Degrees to radians
- degrees(): Radians to degrees

Examples

radians(180)
degrees(pi)

deprecate_param	<i>Deprecate a parameter</i>
-----------------	------------------------------

Description

Accepts the directly passed deprecated and superceding params and Sends a deprecation message if the deprecated param was used. This function wraps lifecycle::deprecate_warn() to make a standard param deprecation message. It then outputs the final value to be used.

Usage

```
deprecate_param(
  x,
  y,
  fun,
  when,
  check = c("deprecated", "null"),
  always = FALSE
)
```

Arguments

<code>x</code>	name. Deprecated param
<code>y</code>	name. Superceding param
<code>fun</code>	character. Name of function
<code>when</code>	character. Version number in which the deprecation happened.
<code>check</code>	character. Method to check if deprecated param was used
<code>always</code>	If FALSE, the default, will warn every 8 hours. If TRUE, will always warn in direct usages. Indirect usages keep warning every 8 hours to avoid disrupting users who can't fix the issue. Only use <code>always = TRUE</code> after at least one release with the default.

Value

final value to be used

Examples

```
foo <- function(dep = deprecated(), sup = 10) {
  sup <- deprecate_param(
    dep, sup,
    fun = "foo", when = "0.0.1"
  )
  return(sup)
}

foo() # following defaults, no deprecation message
foo(sup = 3) # no deprecation message triggered
foo(dep = 3) # deprecation message triggered

# convenient nested function when deprecating multiple params
bar <- function(dep1 = deprecated(),
  dep2 = deprecated(),
  sup1 = 10,
  sup2 = 20) {
  # internally defined function that streamlines downstream deprecations
  .dep <- function(...) {
    deprecate_param(..., fun = "bar", when = "0.0.2")
  }
}
```

```

    sup1 <- .dep(dep1, sup1)
    sup2 <- .dep(dep2, sup2)

    return(list(sup1, sup2))
}

bar(sup1 = 100)
bar(dep1 = 100, dep2 = "hello")

```

dir_manifest

*Generate file manifest list from a directory***Description**

Create a list of full filepaths (character) that are named by with the respective `[basename()]`. Allows easy \$ exploration and indexing of items.

All params are directly passed to `[list.files()]` except for `full.names`. `[list.files()]` also normally returns both actual files and directories when `recursive = FALSE`, but this function specifically tests if items are existing files and not directories with `file_test(op = -f)` and fully obeys that flag in all cases.

Usage

```

dir_manifest(
  path = ".",
  pattern = NULL,
  all.files = FALSE,
  recursive = FALSE,
  ignore.case = FALSE,
  include.dirs = FALSE,
  no.. = FALSE,
  as.list = TRUE
)

```

Arguments

- | | |
|-----------|--|
| path | a character vector of full path names; the default corresponds to the working directory, <code>[getwd()]</code> . Tilde expansion (see path.expand) and normalizePath() are performed. Missing values will be ignored. Elements with a marked encoding will be converted to the native encoding (and if that fails, considered non-existent). |
| pattern | an optional regular expression. Only file names which match the regular expression will be returned. |
| all.files | a logical value. If <code>FALSE</code> , only the names of visible files are returned (following Unix-style visibility, that is files whose name does not start with a dot). If <code>TRUE</code> , all file names will be returned. |

- `recursive` logical. Should the listing recurse into directories?
- `ignore.case` logical. Should pattern-matching be case-insensitive?
- `include.dirs` logical. Should subdirectory names be included in recursive listings?
- `no..` logical. Should both "." and ".." be excluded also from non-recursive listings?
- `as.list` logical. Should output be a list or a named character vector

Value

full and normalized filepaths named by the file basename as either a list (default) or if `as.list = FALSE`, a character vector.

Examples

```
dir_manifest()
```

<code>dt_dcast_string</code>	<i>dt_dcast_string</i>
------------------------------	------------------------

Description

Data.table dcast using character inputs for formula

Usage

```
dt_dcast_string(data, col_name1, col_name2, value.var)
```

Arguments

- `data` a `data.table`
- `col_name1` character. LHS of cast formula
- `col_name2` character. RHS of cast formula
- `value.var` character. Name of the column whose values will be filled to cast.

Value

A keyed `data.table` that has been cast

See Also

```
data.table::dcast.data.table\(\)
```

Examples

```
x <- data.table::data.table(
  col1 = c(rep("a", 3), rep("b", 3)),
  col2 = rep(LETTERS[1:3], 2),
  value = c(1:6)
)
force(x)
y <- dt_dcast_string(x, "col1", "col2", "value")
force(y)
```

dt_remove_na	<i>dt_remove_na</i>
--------------	---------------------

Description

set NA values to 0 in a data.table object

Usage

```
dt_remove_na(DT)
```

Arguments

DT datatable

Value

A data.table

Examples

```
x <- data.table::data.table(x = 1:3, y = c(1, NA, 2))
dt_remove_na(x)
x
```

dt_set_row_order	<i>Set specific data.table row order</i>
------------------	--

Description

Set specific data.table row order

Usage

```
dt_set_row_order(x, neworder)
```

Arguments

x	data.table
neworder	numerical vector to reorder rows

Value

A data.table

Examples

```
x <- data.table::data.table(x = c("a", "b", "c"), y = 1:3, z = 5:7)
dt_set_row_order(x, neworder = c(1, 3, 2))
x
```

dt_sort_combine_two_columns
<i>dt_sort_combine_two_columns</i>

Description

fast sorting and pasting of 2 character columns in a data.table

Usage

```
dt_sort_combine_two_columns(DT, column1, column2, myname = "unif_gene_gene")
```

Arguments

DT	datatable
column1	column1 to use
column2	column2 to use
myname	name of combined column to generate

Value

A data.table

Examples

```
x <- data.table::data.table(x = 1:3, y = 5:7)
dt_sort_combine_two_columns(x, column1 = "x", column2 = "y")
x
```

dt_to_matrix	<i>dt_to_matrix</i>
--------------	---------------------

Description

converts data.table to Matrix

Usage

```
dt_to_matrix(x, chunked = FALSE)
```

Arguments

- x data.table object
- chunked logical. Whether to chunk the ingestion to Matrix

Details

When matrices are very large, Matrix::Matrix(x) and as(x, "Matrix") may throw Error: vector memory exhausted (To get around this, we chunk the conversion to Matrix by chunks of up to roughly 2e+08 matrix cells.

Value

A Matrix

Examples

```
x <- data.table::data.table(x = c("a", "b", "c"), y = 1:3, z = 5:7)
dt_to_matrix(x)
dt_to_matrix(x, chunked = TRUE)
```

file_extension	<i>Get file extension</i>
----------------	---------------------------

Description

Get the file extension from a provided filepath

Usage

```
file_extension(file)
```

Arguments

- file character. Filepath

Value

character

Examples

```
file_extension("my_file.txt")
```

getDistinctColors	<i>getDistinctColors</i>
-------------------	--------------------------

Description

Returns a number of distinct colors based on the RGB scale. When fewer than 444 colors are requested, colors are returned in an order that maximizes differences. When more than 444 colors are requested, a recycled sampling of the available colors is returned.

Usage

```
getDistinctColors(n, seed = 1234)
```

Arguments

n	number of colors wanted
seed	seed to use when randomizing color order. Default is 1234

Value

character vector of hexadecimal distinct colors

See Also

Other basic color palette functions: [getMonochromeColors\(\)](#), [getRainbowColors\(\)](#)

Examples

```
getDistinctColors(500)
getDistinctColors(500, seed = 1)
```

getMonochromeColors	Create color scaling for a single color starting from black
---------------------	---

Description

Create color scaling for a single color starting from black

Usage

```
getMonochromeColors(col, n = 256L, ...)
```

Arguments

col	hexadecimal color to scale scale towards
n	number of colors to request in monochrome palette
...	Arguments passed on to grDevices::colorRampPalette

bias a positive number. Higher values give more widely spaced colors at the high end.

space a character string; interpolation in RGB or CIE Lab color spaces.

interpolate use spline or linear interpolation.

alpha logical: should alpha channel (opacity) values be returned? It is an error to give a true value if space is specified.

Value

character vector

See Also

Other basic color palette functions: [getDistinctColors\(\)](#), [getRainbowColors\(\)](#)

Examples

```
getMonochromeColors("green", n = 100)
```

getRainbowColors	<i>getRainbowColors</i>
------------------	-------------------------

Description

Returns a number of rainbow colors spaced around the spectrum. Only 100 unique colors will be supplied after which they are recycled.

Usage

```
getRainbowColors(n, slim = 1, vlim = 1, seed = 1234)
```

Arguments

n	numeric. Number of colors wanted
slim	numeric. Saturation. If two values are provided, a random uniform distribution with the two values as min and max will be used.
vlim	numeric. Value. If two values are provided, a random uniform distribution with the two values as min and max will be used.
seed	integer. seed to use when randomizing saturation and value. Default is 1234.

Value

character vector of hexadecimal rainbow colors

See Also

Other basic color palette functions: [getDistinctColors\(\)](#), [getMonochromeColors\(\)](#)

Examples

```
getRainbowColors(100)
getRainbowColors(10, slim = c(0.5, 1), vlim = c(0.3, 1))
getRainbowColors(10, slim = c(0.5, 1), vlim = c(0.3, 1), seed = 11)
```

giotto_bpparam	<i>Giotto Default BiocParallel Param</i>
----------------	--

Description

Get and set a cached BiocParallel BPPARAM setting. Cached settings can either be retrieved using `giotto_bpparam()` or directly with `getOption("giotto.bpparam")`

Usage

```
giotto_bpparam(BPPARAM = NULL)
```

Arguments

BPPARAM set a BiocParallel parameter class deciding how to perform parallelized or (or sequential) evaluation

Value

If a BiocParallelParam is passed to BPPARAM, it will be cached and invisibly returned. If BPPARAM is NULL then the cached param will be invisibly returned (SerialParam) is default.

giottoNewLog	<i>Create a new Giotto log</i>
--------------	--------------------------------

Description

Create a new Giotto log

Usage

```
giottoNewLog(logdir)
```

Arguments

logdir (optional) specific directory in which to generate logfiles. If not provided, will choose a directory based on `getOption("giotto.logdir", tempdir())`

Value

a log file

Examples

```
giottoNewLog("my_log_directory")
```

giottoReadLog	<i>Read from the last generated log file</i>
---------------	--

Description

Read from the last generated log file

Usage

```
giottoReadLog(filepath = getOption("giotto.last_logpath", NULL))
```

Arguments

filepath character. filepath to log file. If omitted, tries to find the last created log (will not work after a crash)

Value

character

Examples

```
log_write(x = "my content")
giottoReadLog()
```

gstop	<i>Module-specific error message</i>
-------	--------------------------------------

Description

Send an error message formatted with `wrap_txt()`. Prepends the Giotto module from which the error was triggered. This function should not be used directly, but instead an internal `.gstop()` function should be created using this framework for each module using it. The `.n` param should be incremented to 2L for this wrapper function

Usage

```
gstop(
  ...,
  sep = " ",
  strWidth = 100,
  errWidth = FALSE,
  .module,
  .prefix = " ",
  .initial = "",
  .n = 1L,
  .call = TRUE,
  .warn_nstack = getOption("giotto.warn_gstop_nstack", FALSE)
)
```

Arguments

... additional strings and/or elements to pass to `wrap_msg`

sep how to join elements of string (default is one space)

strWidth externally set wrapping width. (default value of 100 is not effected)

errWidth default = FALSE. Set strWidth to be compatible with error printout

.module character. Giotto module to send the error from

<code>.prefix</code>	character. prefix for later lines
<code>.initial</code>	character. prefix for first line
<code>.n</code>	stack frames back where the error happened
<code>.call</code>	logical, whether to include the call selected through <code>.n</code> as the location where the error was. Default is TRUE
<code>.warn_nstack</code>	logical. whether to warn when there are insufficient stackframes for requested <code>.n</code> (default = FALSE)

Value

character message

Examples

```
try(
  gstop("My stop message", .module = "GiottoUtils"),
  silent = TRUE
)
```

handle_errors	<i>Error handling</i>
---------------	-----------------------

Description

Some errors may lock up the console for much longer than they should. This can often be because R has difficulty figuring out the callstack to generate the error message. This function is a simple wrapper to set up a tryCatch evaluation that will return all enclosed errors with the `stop()` arg `call. = FALSE` setting.

Usage

```
handle_errors(expr, prefix = "")
```

Arguments

<code>expr</code>	expression to evaluate
<code>prefix</code>	string. An optional prefix to add before the error to help report where it was from.

Value

Throws an error without processing stop calls.

Examples

```
x <- function() {  
  handle_errors(stop("this is an error"))  
}  
if (FALSE) {  
  x()  
}  
  
y <- function() {  
  handle_errors(  
    stop("this is an error"),  
    prefix = "error location:"  
  )  
}  
if (FALSE) {  
  y()  
}
```

handle_warnings

<i>Warning handling</i>

Description

Utility to gracefully handle expected warnings

Usage

```
handle_warnings(expr)
```

Arguments

expr	expression to evaluate
------	------------------------

Value

list of "result" which is the output of expr and "warnings" which are any warnings that were generated during eval.

Examples

```
f <- function() {  
  warning("this is a warning")  
  return(TRUE)  
}  
handle_warnings(f())
```

init_option	<i>Initialize an option</i>
-------------	-----------------------------

Description

Set an option only if it does not exist yet.

Usage

```
init_option(x, value)
```

Arguments

x	character. Option to initialize
value	value to set

Value

NULL invisibly

Examples

```
init_option("dummy_option", 2)
init_option("dummy_option", 5)
getOption("dummy_option") # still 2
```

keep_aware	<i>keep_aware</i>
------------	-------------------

Description

Prevent the machine from going to sleep during a long computation, and allow it again afterwards.

This exists because a system sleep is invisible to every in-process measurement. While the machine is asleep wall clock advances and CPU time does not, so a step that spans a sleep appears many times slower while doing identical work: recorded pipeline timings have been inflated 5-14x this way, with CPU-seconds, memory and I/O counters all looking entirely normal.

Nothing in Giotto holds a sleep assertion unless you call `keep_aware()` or `gwith_aware()`. Attaching the package changes no power settings.

Usage

```
keep_aware(on = TRUE)
```

Arguments

on	logical. TRUE holds the assertion, FALSE releases it.
----	---

Details

Uses caffeinate on macOS and systemd-inhibit on Linux; on other platforms, or when neither is available, it reports that it cannot help and does nothing. No sleep assertion can outlive the R session: on both platforms the helper is bound to this process and exits with it. For the same reason, sessions running side by side on one machine hold and release independently.

Idempotent within a session — calling it repeatedly with `on = TRUE` leaves a single helper. A session only ever tracks and releases the helper it started itself, so it cannot report another session's hold as its own.

Value

TRUE if an assertion is held after the call, otherwise FALSE, invisibly

Options

`giotto.prevent_sleep` when FALSE, `keep_awake(TRUE)` and `gwith_awake()` become no-ops (default TRUE). This is a kill switch for shared machines, cluster nodes and CI, where holding a power assertion is unwanted; it does **not** cause anything to hold one on its own. It gates taking an assertion only — `keep_awake(FALSE)` still releases one that is already held, so setting the option partway through cannot strand it.

See Also

`gwith_awake()` to scope it to a block of code

Examples

```
## Not run:
keep_awake(TRUE)
# ... long pipeline ...
keep_awake(FALSE)

## End(Not run)
```

lifecycle_badge

lifecycle badge

Description

Generate markdown needed for lifecycle badges that help communicate stability of functions in documentation when added in backticks as shown below.

```
#' `r lifecycle::badge("experimental")`
#' `r lifecycle::badge("deprecated")`
#' `r lifecycle::badge("superseded")`
```

Usage

```
lifecycle_badge(stage = "stable")
```

Arguments

stage character. 'stable', 'experimental', 'deprecated', or 'superseded'

Value

markdown

See Also

Other lifecycle: [reexports](#)

Examples

```
lifecycle_badge()
```

local_seed	<i>Set a seed local to a call</i>
------------	-----------------------------------

Description

When used inside a call, `local_seed()` first records the existing seed then transiently sets a specific one. When the call exits, the recorded seed is set again, so that the transient seed setting leaves no effects.

Based on discussion from <https://support.bioconductor.org/p/110439/>

Usage

```
local_seed(seed)
```

Arguments

seed seed value to set

Value

set up seed passed to environment

Examples

```
f <- function() {
  local_seed(1234)
  r_val <- rnorm(1)
  return(r_val)
}

rnorm(1) # make sure a seed exists (not needed but handy for this example)

seed1 <- .Random.seed
x <- f()
seed2 <- .Random.seed
y <- rnorm(1)
seed3 <- .Random.seed
z <- f()

identical(seed1, seed2)
!identical(seed1, seed3)

identical(x, z)
!identical(x, y)
```

log_write

Write to log file

Description

Write to log file

Usage

```
log_write(file_conn = .log_conn(), x = "", collapse = " ", main = NULL)
```

Arguments

file_conn	a file connection (Uses last created or generates a new one if previous does not exist.)
x	character vector. Content to write
collapse	character. Collapse to use with x
main	character. Title to assign log entry

Value

character

Examples

```
log_write(x = "my content")
```

melt_matrix	<i>Melt a matrix</i>
-------------	----------------------

Description

Simple implementation of melt for matrices to produces results similar to that of reshape2's now that it is deprecated.

The param `na.rm` is not implemented.

Usage

```
melt_matrix(data, varnames = NULL, ..., as.is = FALSE, value.name = "value")
```

Arguments

<code>data</code>	matrix to melt
<code>varnames</code>	variable names to use in molten <code>data.table</code>
<code>...</code>	further arguments passed to or from other methods
<code>as.is</code>	if FALSE, dimnames will be converted to factor or integer etc as appropriate. If TRUE, they will be left as strings.
<code>value.name</code>	name of variable used to store values (default = "value")

Value

`data.table` in long format

Examples

```
set.seed(9)
Data <- matrix(round(rnorm(12, 10, 4)), nrow = 4, ncol = 3)
melt_matrix(Data)
```

<code>new_github_ver_avail</code>	<i>Check if a package has newer github version</i>
-----------------------------------	--

Description

This function works by downloading the DESCRIPTION file from a github repo and then comparing the version number with a provided value. If the download fails or the github version is not newer than what is installed, then it silently returns NULL. If the github version is newer, the new version number will be returned

Usage

```
new_github_ver_avail(url, current_ver = NULL)
```

Arguments

`url` character. url to the package to check's DESCRIPTION file.
`current_ver` character. Current version to check against

Value

character. Version number

Examples

```
url <- "https://raw.githubusercontent.com/giotto-suite/GiottoData/master/DESCRIPTION"
new_github_ver_avail(url, 0.2)
```

`pbar` *Create a **progressr** progress tracker.*

Description

Create a **progressr** progress tracker. This is a wrapper around `progressr::progressor()`.

Usage

```
pbar(
  steps = length(along),
  along = NULL,
  offset = 0L,
  scale = 1L,
  transform = function(steps) scale * steps + offset,
  message = character(0L),
  label = NA_character_,
  trace = FALSE,
  initiate = TRUE,
  finalize = FALSE,
  auto_finish = TRUE,
  on_exit = !identical(envir, globalenv()),
  enable = getOption("progressr.enable", TRUE),
  envir = parent.frame()
)
```

Arguments

`steps` (integer) Number of progressing steps.
`along` (vector; alternative) Alternative that sets `steps = length(along)`.
`offset, scale` (integer; optional) scale and offset applying the transform `steps <- scale * steps + offset`.

transform	(function; optional) A function that takes the effective number of steps as input and returns another finite and non-negative number of steps.
message	(character vector or a function) If a character vector, then it is pasted together into a single string using an empty separator. If a function, then the message is constructed by <code>conditionMessage(p)</code> calling this function with the progression condition <code>p</code> itself as the first argument.
label	(character) A label.
trace	(logical) If TRUE, then the call stack is recorded, otherwise not.
initiate	(logical) If TRUE, the progressor will signal a progression 'initiate' condition when created.
finalize	(logical) If TRUE, the progressor signals a progression 'shutdown' condition when finalized by the garbage collector.
auto_finish	(logical) If TRUE, then the progressor will signal a progression 'finish' condition as soon as the last step has been reached.
on_exit, envir	(logical) If TRUE, then the created progressor will signal a progression 'finish' condition when the calling frame exits. This is ignored if the calling frame (<code>envir</code>) is the global environment.
enable	(logical) If TRUE, progression conditions are signaled when calling the progressor function created by this function. If FALSE, no progression conditions is signaled because the progressor function is an empty function that does nothing.

Value

a progressr progressor function

See Also

[with_pbar\(\)](#)

Examples

```
a <- function(x = 10) {
  pb <- pbar(steps = x)
}

a <- function(x = seq_len(10)) {
  pb <- pbar(along = x)
}
```

prev_call	<i>Previous calls</i>
-----------	-----------------------

Description

Functions to check the call stack and get aspects of previous calls. Used in putting together object histories and error handling.

Usage

```
get_prev_call(toplevel = 1L)

get_prev_fname(toplevel = 3L)

get_args(toplevel = 2L, verbose = FALSE)

get_args_list(toplevel = 1L, keep = NULL, ...)
```

Arguments

toplevel	integer. Relative stack where the function call was made.
verbose	be verbose
keep	character. When NULL, all params are captured. If not NULL, specifies which params to capture.
...	additional params to capture

Value

character

Functions

- `get_prev_call()`: Get previous call
- `get_prev_fname()`: Get previous call function name
- `get_args()`: Get previous call args as named character vector
- `get_args_list()`: Get call args as named list

Examples

```
get_prev_call()

get_prev_fname(1)

# preserve params
foo <- function(a, b) get_args(toplevel = 1)
foo(a = 1, b = 2)
```

```

bar <- function() get_args() # default toplevel = 2
baz <- function(x, y) bar()
baz("a", "b")
a <- function(x = 1, y = 2, ...) {
  get_args_list(...)
}

a(z = 3, keep = "y")

```

print_list

Pretty print formatting for lists and vectors

Description

Pretty print formatting for lists and vectors

Usage

```
print_list(x, pre = "")
```

Arguments

x	list of items to print. All entries must be named and have <code>as.character()</code> methods
pre	character. Optional characters to place at the head of each line

Value

lists and vectors

Examples

```

print_list(list())
print_list(c())

testvec <- seq(3)
names(testvec) <- LETTERS[seq(3)]
print_list(testvec)

test <- list(
  name1 = "1",
  longername2 = "test_char",
  thirdname = factor("this will be converted with as.character()"),
  df_test = data.frame(a = "a", b = "b"),
  list_test = list(a = 1, b = 2, c = 3),
  formula_test = ~a
)
print_list(test)
print_list(test, pre = "* ")

```

py_active_env	<i>Active python environment</i>
---------------	----------------------------------

Description

Check for an active python environment without initialization. If none initialized, FALSE is returned. If an initialized environment is found, the env name based on `reticulate::conda_list()` will be returned. The following options will also be updated with information about the active environment:

- "giotto.py_active_env" (*envname*)
- "giotto.py_active_ver" (*python version*)
- "giotto.py_active_path" (*path to python*)

Usage

```
py_active_env()
```

Value

boolean

Examples

```
py_active_env()
```

random_seed	<i>Set a random seed</i>
-------------	--------------------------

Description

From reproducible package, `set.randomseed()`
 Convenient function for applying a random seed. Usually used with `on.exit()` when a specified seed is applied so that non-random operations do not interfere with other aspects of a user's work. May also be called with `set.seed = TRUE` to simply return a random seed to use, without actually having applied it.

Usage

```
random_seed(set.seed = TRUE)
```

Arguments

<code>set.seed</code>	Whether to apply the random seed. Defaults to TRUE
-----------------------	--

Value

The seed value is returned invisibly

Examples

```
random_seed()
```

require_depth	<i>Coerce to AT LEAST specified nesting depth</i>
---------------	---

Description

Coerce to AT LEAST specified nesting depth

Usage

```
require_depth(x, dnames = NULL, min_depth = 1L, count = 1L)
```

Arguments

x	object to evaluate
dnames	character. vector of names to apply per depth level if not already named the name entries should match the depth they are intended for. NULL values
min_depth	required minimum nesting depth are ignored.
count	do not use

Value

list

Examples

```
x <- list(a = 1)
require_depth(x, min_depth = 2L)
```

scipy_sparse	<i>Convert between Matrix and scipy sparse representations</i>
--------------	--

Description

Convert between **R Matrix** sparse matrix representation and python scipy sparse formats. Inspired by the implementation in **nmslibR**. Currently compatible classes:

- `to_scipy_sparse()`: `matrix`, `dgCMatrix`, `dgRMatrix`, `dgTMatrix`
- `from_scipy_sparse()`: `scipy.sparse._csr.csr_matrix`, `scipy.sparse._csc.csc_matrix`

Usage

```
to_scipy_sparse(x, format = c("C", "R"), transpose = FALSE, ...)
```

```
from_scipy_sparse(x, format = c("C", "R"), transpose = FALSE, ...)
```

Arguments

<code>x</code>	matrix-like object to convert
<code>format</code>	character. Either "C" for Compressed Sparse Column or "R" for Compressed Sparse Row matrix representations
<code>transpose</code>	whether to transpose the matrix. default is FALSE
<code>...</code>	additional params to pass to <code>scipy.sparse.cs*_matrix()</code> or <code>Matrix::sparseMatrix</code> depending on which output is desired.

Value

scipy or Matrix sparse representation depending on 'to' or 'from' respectively

Examples

```
# example data
m <- matrix(data = 0L, nrow = 400, ncol = 300)
m[sample(400, 100), sample(300, 100)] <- runif(100, max = 10)
dgc <- Matrix::rsparsematrix(nrow = 400L, ncol = 300L, density = 0.1)

# some conversions
py_m_c <- to_scipy_sparse(m, format = "C")
py_m_r_t <- to_scipy_sparse(m, format = "R", transpose = TRUE)

py_dgc_c <- to_scipy_sparse(dgc, format = "C")
py_dgc_r <- to_scipy_sparse(dgc, format = "R")

dgc_revert <- from_scipy_sparse(py_dgc_c, format = "C")
dgr_revert <- from_scipy_sparse(py_dgc_r, format = "R")
identical(dgc, dgc_revert)
```

set_if

Set if

Description

Set a default value if a test is true.

Usage

```
x %null% y
```

```
x %na% y
```

```
x %none% y
```

Arguments

x	object to test
y	default value to set

Value

if x tests TRUE, y is returned if x tests FALSE x is returned

Examples

```
NULL %null% 1
2 %null% 1

"test" %na% 10
NA_character_ %na% "missing"

c() %none% 20
list() %none% "a"
logical(5) %none% 10
```

str_abbreviate	<i>Abbreviate a string</i>
----------------	----------------------------

Description

Abbreviates a string in the format of head[...]tail when it exceeds the length specified by width. Useful for shortening how filepaths are displayed.

Usage

```
str_abbreviate(string, width = 60L, head = 15L, tail = 35L)
```

Arguments

string	character. Input string
width	numeric. Strings longer than this many characters will be abbreviated
head	numeric. Number of characters to include before abbreviated section
tail	numeric. Number of characters to include after abbreviated section

Value

character

Examples

```
a <- "/short/file/path/"
b <- "/much/longer/fooooooooooooooooooooo/baaaaaaaaaaaaaaaaaaaaar/file/path/"
str_abbreviate(c(a, b))
str_abbreviate(c(a, b), width = 10, head = 3, tail = 3)
```

str_convenience	<i>String convenience functions</i>
-----------------	-------------------------------------

Description

String convenience functions

Usage

```
str_vector(x, qchar = c("single", "double"))

str_bracket(x)

str_parenth(x)

str_double_quote(x)

str_quote(x)

str_reformat(x, ...)
```

Arguments

x	string item(s) to format
qchar	quote character to use. Either 'single' or "double"
...	additional params to pass to strwrap()

Value

character

Functions

- `str_vector()`: Format a vector of values into 'a', 'b', 'c' or "a", "b", "c" depending on qchar via [toString\(\)](#)
- `str_reformat()`: capture and reformat a print with [strwrap\(\)](#)

Examples

```

x <- "test"
cat(str_bracket(x), "\n")
cat(str_parenth(x), "\n")
cat(str_double_quote(x), "\n")
cat(str_quote(x), "\n")
# format a set of character values with str_vector()
vec <- c("item1", "item2", "item3")
cat(str_vector(vec)) # single quote (default)
cat(str_vector(vec, qchar = "double")) # double quote
# print post-processing with str_reformat()
txt <- "hello
      world"
cat(str_reformat(cat(txt), indent = 3))

```

str_locate2

*Find location of match in string***Description**

Implementation of **stringr**'s str_locate with base R.

Usage

```
str_locate2(string, pattern)
```

Arguments

string	Input vector. Either a character vector, or something coercible to one.
pattern	Pattern to look for.

Value

integer matrix with two columns and one row for each element of string. The first column, start, gives the position at the start of the match, and the second column, end, gives the position of the end.

Examples

```

fruit <- c("apple", "banana", "pear", "pineapple")
str_locate2(fruit, "$")
str_locate2(fruit, "a")
str_locate2(fruit, "e")
str_locate2(fruit, c("a", "b", "p", "p"))

```

 suite_install

Giotto Suite GitHub Installation

Description

[Experimental]

Installation convenience function. Helps with Giotto Suite installation from the GitHub website by ensuring modules are installed in the correct order. Also automatically switches to the R4.4.0 version when user R version is lower than 4.4.1.

This utility will do a full reinstall of the specified Giotto Suite modules but non-Suite dependencies will never prompt to be updated.

Usage

```
suite_install(
  modules = suite_packages(),
  suite_deps = TRUE,
  ref = "main",
  dry_run = FALSE,
  install_arrow = NULL,
  ...
)
```

Arguments

modules	character. Which modules to install. Defaults to the core packages needed for Giotto to run.
suite_deps	logical. Whether to install any potential Giotto Suite dependency modules
ref	character. Currently one of "main", "R4.4.0", "R4.1.0", "dev", or "disk". These determine which branches to install. See details.
dry_run	logical. When TRUE, only print the install commands instead of actually running them.
install_arrow	logical or NULL. Whether to (re)install <i>arrow</i> with <i>zstd</i> support when "arrow" is in modules and the installed version is missing or lacks <i>zstd</i> . NULL (default) errors with the install command, TRUE runs it, FALSE skips it. See the <i>Non-suite packages</i> section.
...	additional params to pass to <code>remotes::install_github()</code>

Value

TRUE if install succeeds

ref "main"

Installs the main Giotto version. This version is expected to chase the highest R version.

ref "R4.4.0", r440, or 440

Mirrors the main branch, but locks the R version at 4.4.0 (lowest version supported by all dependencies)

ref "R4.1.0", r410, or 410

Mirrors the main branch, but locks the R version at 4.1.0. This version requires installation of {Matrix} 1.6-5

ref "dev"

This version is ahead of the main version and also chases the latest R version.

ref "disk", giottodisk, or gsource

The on-disk build set. Installs the gsource branches that *GiottoDisk* builds against, and *GiottoDisk* itself from its dev integration branch. This is the only ref *GiottoDisk* is compatible with, so requesting *GiottoDisk* with any other ref switches to this one.

GiottoData and *GiottoDB* have no gsource branch and install from their default branch. Neither is part of the *GiottoDisk* dependency chain, so they are only installed when asked for by name.

Non-suite packages

"tilework" and "arrow" can be requested as modules under any ref. With suite_deps = TRUE they are only added automatically for *GiottoDisk*.

tilework installs from GitHub. *arrow* installs from source through r-universe with zstd enabled, since zstd is needed for 10x parquet files. This build can take a long time, so it only runs when *arrow* is missing or lacks zstd and install_arrow = TRUE.

Examples

```
if (FALSE) {
  # install core packages
  suite_install()
  suite_install("GiottoClass", ref = "dev")

  # install the on-disk build set
  suite_install("GiottoDisk", ref = "disk", install_arrow = TRUE)

  # install ONLY Giotto, ignoring module dependencies
  # (i.e. GiottoVisuals, GiottoClass, etc)
  suite_install("Giotto", suite_deps = FALSE)
}
```

suite_packages	<i>Giotto Suite Packages</i>
----------------	------------------------------

Description

Returns character vector of Giotto Suite's packages. Only the core packages are returned by default.

Usage

```
suite_packages(type = "core")
```

Arguments

type	character. Type of module to return. Current choices are "core" for the packages needed for Giotto to run, "extensions" for extending packages such as <i>GiottoData</i> , and "all" for all current modules to be returned.
------	--

Value

character

Examples

```
suite_packages()  
suite_packages("core")  
suite_packages("extensions")  
suite_packages("all")
```

time_format	<i>Format time for printing</i>
-------------	---------------------------------

Description

Format time for printing

Usage

```
time_format(secs)
```

Arguments

secs	numeric. seconds
------	------------------

Details

Code from [timetaken](#)

Value

character

Examples

time_format(90)

vmmsg	<i>Verbose message handler</i>
-------	--------------------------------

Description

Verbose message handler

Usage

```
vmmsg(  
    ...,  
    .v = NULL,  
    .is_debug = FALSE,  
    .vopt = getOption("giotto.verbose", TRUE),  
    .log_path = NULL  
)
```

Arguments

- | | |
|-----------|---|
| ... | additional strings and/or elements to pass to wrap_msg |
| .v | verbose flag to pass. Will check options through .vopt if NULL (default). This param is intended for passing function-level verbose flags |
| .is_debug | flag as a debug print (only prints when .v or .vopt is 'debug') |
| .vopt | global verbosity option to pull from |
| .log_path | filepath. Temporarily log to a specific location. Does not disturb the "giotto.last_logpath" option. |

Value

character

Examples

```
# common usage (.v is logical)
vmsg("print me", .v = TRUE)
vmsg("dont print me", .v = FALSE)

# debug messages (.v == "debug")
# flag as a debug message using .is_debug = TRUE
vmsg("I am a debug message", .is_debug = TRUE, .v = TRUE) # no print
vmsg("I am a debug message", .is_debug = TRUE, .v = "debug") # prints

vmsg("print me", .v = "debug") # also prints non-debug messages

# with global option
options("giotto.verbose" = TRUE)
vmsg("Print by default")

options("giotto.verbose" = FALSE)
vmsg("Do not print by default")
vmsg("Do not print by default", .v = TRUE)
# function level input overrides global option
```

with

With utilities

Description

Simple *with* functions. Similar to or from **withr** implementations.

Usage

```
gwith_options(new, code)

gwith_package(package, code, pos = 2L)

gwith_seed(seed = 1234, code, ..., substitute = TRUE, envir = parent.frame())

gwith_awake(code)
```

Arguments

new	new option to set and its value
code	R code to execute with temporary settings
package	character. The name of a package
pos	integer. position in search to attach at. Default = 2
seed	numeric. seed to set
...	additional params to pass. See details.
substitute	If TRUE , argument <code>expr</code> is <code>substitute()</code> :ed, otherwise not.
envir	The environment in which the expression should be evaluated.

Details

`gwith_seed()` : ... passes to `set.seed()`

`gwith_awake()` : holds a sleep assertion for the duration of code, then releases it. A system sleep part-way through a long computation inflates its wall-clock time while leaving CPU time, memory and I/O counters looking normal, so an affected measurement is indistinguishable from a slow one without this. See [keep_awake\(\)](#), which this wraps, for platform support and for the `giotto.prevent_sleep` kill switch.

Value

the evaluated value of code

Functions

- `gwith_options()`: Eval with temporary option setting
- `gwith_package()`: Eval with temporarily attached package
- `gwith_seed()`: Eval with temporary specifiable seed
- `gwith_awake()`: Eval while preventing the machine from sleeping

Examples

```
# options #####
gwith_options(list(gutils.temp = "found"), {
  print(getOption("gutils.temp", default = "not_found"))
})
getOption("gutils.temp", default = "not_found")

# packages #####
# temporarily attach at end of search (right before "base")
gwith_package("data.table", print(search()), pos = length(search()))
search()

# seed #####
start_seed <- .Random.seed

# identical generation
a <- gwith_seed(runif(10), seed = 1234)
b <- gwith_seed(runif(10), seed = 1234)
identical(a, b)

# does not alter pre-existing seed
end_seed <- .Random.seed
identical(start_seed, end_seed)
# sleep prevention #####
## Not run:
gwith_awake({
  g <- runPCA(g, feats_to_use = "hvf", ncp = 30)
  g <- createNearestNetwork(g, dim_reduction_to_use = "pca")
})
```

```
## End(Not run)
```

with_pbar

Track progress while evaluating an R expression

Description

Track progress while evaluating an R expression. This is a wrapper around `progressr::with_progress()`. **progressr** is useful because it handles progress reporting in parallel contexts. However, it is worth noting that Giotto does not adhere to **progressr**'s guideline that `with_progress()` should not be used within packages. The default behavior of displaying progress is more convenient for end users of packages with many functions. This implementation may shift over time.

Usage

```
with_pbar(
  expr,
  handlers = progressr::handlers(),
  cleanup = TRUE,
  delay_terminal = NULL,
  delay_stdout = NULL,
  delay_conditions = NULL,
  interrupts = getOption("progressr.interrupts", TRUE),
  interval = NULL,
  enable = NULL
)
```

Arguments

<code>expr</code>	An R expression to evaluate.
<code>handlers</code>	A progression handler or a list of them. If NULL or an empty list, progress updates are ignored.
<code>cleanup</code>	If TRUE, all progression handlers will be shutdown at the end regardless of whether the progression is complete or not.
<code>delay_terminal</code>	If TRUE, output and conditions that may end up in the terminal will be delayed.
<code>delay_stdout</code>	If TRUE, standard output is captured and relayed at the end just before any captured conditions are relayed.
<code>delay_conditions</code>	A character vector specifying <code>base::condition</code> classes to be captured and relayed at the end after any captured standard output is relayed.
<code>interrupts</code>	Controls whether interrupts should be detected or not. If TRUE and an interrupt is signaled, progress handlers are asked to report on the current amount of progress when the evaluation was terminated by the interrupt, e.g. when a user pressed Ctrl-C in an interactive session, or a batch process was interrupted because it ran out of time. Note that it's optional for a progress handler to support this and only some do.

interval	(numeric) The minimum time (in seconds) between successive progression updates from handlers.
enable	(logical) If FALSE, then progress is not reported. The default is to report progress in interactive mode but not batch mode. See below for more details.

Value

the result of the evaluated code

See Also

[pbar\(\)](#)

Examples

```
a <- function(x = 10) {
  with_pbar({
    pb <- pbar(x)
    for (i in seq_len(x)) {
      Sys.sleep(0.1)
      pb()
    }
  })
}

a()
```

wrap_msg

Wrap message

Description

Wrap message

Usage

```
wrap_msg(..., sep = " ")
```

Arguments

...	additional strings and/or elements to pass to wrap_txt
sep	how to join elements of string (default is one space)

Value

character

Examples

```
wrap_msg("A message")
```

wrap_txt	<i>Wrap text</i>
----------	------------------

Description

Wrap text

Usage

```
wrap_txt(  
    ...,  
    sep = " ",  
    strWidth = 100,  
    errWidth = FALSE,  
    .prefix = " ",  
    .initial = ""  
)  
  
wrap_txtf(  
    ...,  
    sep = " ",  
    strWidth = 100,  
    errWidth = FALSE,  
    .prefix = " ",  
    .initial = ""  
)
```

Arguments

...	additional params to pass
sep	how to join elements of string (default is one space)
strWidth	externally set wrapping width. (default value of 100 is not effected)
errWidth	default = FALSE. Set strWidth to be compatible with error printout
.prefix	character. prefix for later lines
.initial	character. prefix for first line

Value

character

Examples

```
cat(wrap_txt("A text"))
cat(wrap_txt(
    "Newlines are obeyed.
    The first line is not indented by default.
    later lines are indented by default.

    The text is also wrapped to either a default max width of 100 char
    or the width of the console, whichever is smaller.

    More than one item passed will be concatenated in the same way
    that cat() does."
))

cat(wrap_txtf(
    "This function works the same way as %s, but instead
    of concatenating all elements in the way that %s usually
    does, it uses %s formatting.",
    "wrap_txt()", "cat()", "sprintf()"
))
```

Index

- * **basic color palette functions**
 - getDistinctColors, [10](#)
 - getMonochromeColors, [11](#)
 - getRainbowColors, [12](#)
- * **data.table**
 - dt_dcast_string, [6](#)
 - dt_remove_na, [7](#)
 - dt_set_row_order, [7](#)
 - dt_sort_combine_two_columns, [8](#)
 - dt_to_matrix, [9](#)
- * **lifecycle**
 - lifecycle_badge, [18](#)
- %na%(set_if), [28](#)
- %none%(set_if), [28](#)
- %null%(set_if), [28](#)
- ‘%na%’ (set_if), [28](#)
- ‘%none%’, (set_if), [28](#)
- ‘%null%’, (set_if), [28](#)
- base::condition, [38](#)
- data.table::dcast.data.table(), [6](#)
- degrees, [3](#)
- deprecate_param, [3](#)
- dir_manifest, [5](#)
- dt_dcast_string, [6](#)
- dt_remove_na, [7](#)
- dt_set_row_order, [7](#)
- dt_sort_combine_two_columns, [8](#)
- dt_to_matrix, [9](#)
- environment, [36](#)
- file_extension, [9](#)
- from_scipy_sparse (scipy_sparse), [27](#)
- get_args (prev_call), [24](#)
- get_args_list (prev_call), [24](#)
- get_prev_call (prev_call), [24](#)
- get_prev_fname (prev_call), [24](#)
- getDistinctColors, [10](#)
- getDistinctColors(), [11](#), [12](#)
- getMonochromeColors, [11](#)
- getMonochromeColors(), [10](#), [12](#)
- getRainbowColors, [12](#)
- getRainbowColors(), [10](#), [11](#)
- giotto_bpparam, [12](#)
- giottoNewLog, [13](#)
- giottoReadLog, [13](#)
- grDevices::colorRampPalette, [11](#)
- gstop, [14](#)
- gwith_aware (with), [36](#)
- gwith_aware(), [17](#), [18](#)
- gwith_options (with), [36](#)
- gwith_package (with), [36](#)
- gwith_seed (with), [36](#)
- handle_errors, [15](#)
- handle_warnings, [16](#)
- init_option, [17](#)
- keep_aware, [17](#)
- keep_aware(), [37](#)
- lifecycle_badge, [18](#)
- local_seed, [19](#)
- log_write, [20](#)
- melt_matrix, [21](#)
- new_github_ver_avail, [21](#)
- normalizePath(), [5](#)
- path.expand, [5](#)
- pbar, [22](#)
- pbar(), [39](#)
- prev_call, [24](#)
- print_list, [25](#)
- progression, [23](#)
- progressr::progressor(), [22](#)
- progressr::with_progress(), [38](#)

py_active_env, [26](#)

radians (degrees), [3](#)

random_seed, [26](#)

reexports, [19](#)

require_depth, [27](#)

reticulate::conda_list(), [26](#)

scipy_sparse, [27](#)

set_if, [28](#)

str_abbreviate, [29](#)

str_bracket (str_convenience), [30](#)

str_convenience, [30](#)

str_double_quote (str_convenience), [30](#)

str_locate2, [31](#)

str_parenth (str_convenience), [30](#)

str_quote (str_convenience), [30](#)

str_reformat (str_convenience), [30](#)

str_vector (str_convenience), [30](#)

strwrap(), [30](#)

substitute, [36](#)

suite_install, [32](#)

suite_packages, [34](#)

time_format, [34](#)

timetaken, [34](#)

to_scipy_sparse (scipy_sparse), [27](#)

toString(), [30](#)

TRUE, [36](#)

vmsg, [35](#)

with, [36](#)

with_pbar, [38](#)

with_pbar(), [23](#)

wrap_msg, [39](#)

wrap_txt, [40](#)

wrap_txtf (wrap_txt), [40](#)