

Package ‘tinycodet’

September 27, 2026

Title Functions to Help in your Coding Etiquette

Version 0.8.0

Description Adds some functions to help in your coding etiquette.

‘tinycodet’ primarily focuses on 4 aspects.

- 1) Safer decimal (in)equality testing, standard-evaluated alternatives to `with()` and `aes()`, and other functions for safer coding.
- 2) A new package import system, that attempts to combine the benefits of using a package without attaching it, with the benefits of attaching a package.
- 3) Extending the string manipulation capabilities of the ‘stringi’ R package.
- 4) Reducing repetitive code.

Besides linking to ‘Rcpp’, ‘tinycodet’ has only one other dependency, namely ‘stringi’.

License MIT + file LICENSE

Encoding UTF-8

LinkingTo Rcpp

Suggests tinytest, ggplot2, mgcv, nlme, collapse, kit, knitr, rmarkdown, roxygen2, magrittr, dplyr, data.table, tidytable

Depends R (>= 4.3.0)

Imports methods, Rcpp (>= 1.0.11), stringi (>= 1.7.12)

URL <https://github.com/tony-aw/tinycodet/>,
<https://tony-aw.github.io/tinycodet/>

BugReports <https://github.com/tony-aw/tinycodet/issues/>

Language en-gb

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Tony Wilkes [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0001-9498-8379>>)

Maintainer Tony Wilkes <tonywilkes.nl@gmail.com>

Repository CRAN

Date/Publication 2026-09-27 13:10:02 UTC

Contents

aaa0_tinycodet_help	2
aaa1_tinycodet_safer	3
aaa2_tinycodet_import	4
aaa3_tinycodet_strings	7
aaa4_tinycodet_dry	9
decimal_truth	10
import_as	13
import_data	15
import_diagnose	16
import_env	17
import_from	18
import_helper	19
import_legacy	21
import_ls	22
lock	24
matrix_ops	26
pkgs	27
safer_partialmatch	30
searchenv	31
strcut_loc	32
stri_join_mat	35
stri_locate_ith	37
str_arithmetic	43
str_search	45
str_subset_ops	50
s_pattern	52
tinyimport_alias	55
transform_if	57
with_pro	59
Index	62

aaa0_tinycodet_help	<i>tinycodet: Functions to Help in your Coding Etiquette</i>
---------------------	--

Description

'tinycodet' adds some functions to help in your coding etiquette.
It primarily focuses on 4 aspects:

(1) Safer decimal (in)equality testing, standard-evaluated alternatives to with() and aes(), and other functions for safer coding;
see [tinycodet_safer](#).

(2) A new package import system, that attempts to combine the benefits of using a package without attaching it, with the benefits of attaching a package;

see [tinycodet_import](#)

(3) Extending the string manipulation capabilities of the 'stringi' R-package;
see [tinycodet_strings](#).

(4) Reducing repetitive code;
see [tinycodet_dry](#).

Please check the Change-log (see links below) regularly for updates (such as bug fixes).

'tinycodet' adheres to the [tinyverse](#) philosophy. Besides linking to 'Rcpp', 'tinycodet' only has one other dependency: 'stringi'. No other dependencies, thus avoiding "dependency hell". Most functions in this R-package are vectorized and optimised.

Author(s)

Maintainer: Tony Wilkes <tonywilkes.nl@gmail.com> ([ORCID](#))

References

The badges shown in the documentation of this R-package were made using the services of: <https://shields.io/>

See Also

Useful links:

- 'tinycodet' GitHub main page and Read-Me: <https://github.com/tony-aw/tinycodet/>
- 'tinycodet' package website: <https://tony-aw.github.io/tinycodet/>
- Report bugs at: <https://github.com/tony-aw/tinycodet/issues/>
- Changelog: <https://github.com/tony-aw/tinycodet/blob/main/NEWS.md/> or <https://tony-aw.github.io/tinycodet/news/index.html>
- The 'fastverse', which is related to the 'tinyverse': <https://github.com/fastverse/fastverse/>

aaa1_tinycodet_safer *Overview of the 'tinycodet' "Safer" Functionality*

Description

To help make your code safer, the 'tinycodet' R-package introduces a few functions:

- [Safer decimal \(in\)equality testing](#).
- Standard evaluated versions of some common expression-evaluation functions: [with_pro](#) and [aes_pro](#).

- The [lock_TF](#) function to set and lock T and F to TRUE and FALSE, respectively.
- The [%<-c%](#) operator to assign locked constants.
- [safer_partialmatch](#) to set options for safer dollar, arguments, and attribute matching.

See Also

[tinycodet_help](#)

Examples

```
x <- c(0.3, 0.6, 0.7)
y <- c(0.1*3, 0.1*6, 0.1*7)
x == y # gives FALSE, but should be TRUE
x %d==% y # here it's done correctly
```

aaa2_tinycodet_import *Overview of the 'tinycodet' Import System*

Description

The 'tinycodet' R-package introduces a new package import system.

One can **use** a package **without attaching** the package - for example by using the `::` operator.

Or, one can explicitly **attach** a package - for example by using the [library](#) function.

The advantages and disadvantages of **using without attaching** a package versus **attaching** a package, at least those relevant here, are compactly presented in the following list:

(1) Prevent masking functions from other packages:

[use without attach](#): Yes(advantage); [attaching](#): No(disadvantage);

(2) Prevent masking core R functions:

[use without attach](#): Yes(advantage); [attaching](#): No(disadvantage);

(3) Clarify which function came from which package:

[use without attach](#): Yes(advantage); [attaching](#): No(disadvantage);

(4) Enable functions only in current/local environment instead of globally:

[use without attach](#): Yes(advantage); [attaching](#): No(disadvantage);

(5) Prevent namespace pollution:

[use without attach](#): Yes(advantage); [attaching](#): No(disadvantage);

(6) Minimise typing - especially for replacement or infix operators
 (i.e. typing `package::`%op%`(x, y)` instead of `x %op% y` is cumbersome):
 use without attach: No(disadvantage); attaching: Yes(advantage);

(7) Use multiple related packages, without constantly switching between package prefixes
 (i.e. doing `packagename1::some_function1()`;
`packagename2::some_function2()`;
`packagename3::some_function3()` is chaotic and cumbersome):
 use without attach: No(disadvantage); attaching: Yes(advantage);

What 'tinycodet' attempts to do with its import system, is to somewhat find the best of both worlds.
 It does this by introducing the following functions:

- `import_from`:
 Import specific objects from a package into the current or specific environment.
- `import_as`:
 Import a main package, and optionally its direct minimal dependencies, under a single alias.
 This essentially combines the attaching advantage of using multiple related packages (item 7 on the list), whilst keeping most advantages of using without attaching a package.
- `import_ls`:
 List names of exported objects by category (like "infix operators", or "replacement operators", etc.).
 Can be used in combination with, for example, `library` or `import_from`, to attach or expose all infix- and replacement operators at once.
 This gives the advantage of less typing (item 6 on the above list).
- `import_data`:
 Directly return a data set from a package, to allow straight-forward assignment.
- `import_diagnose`:
 Check for mismatch issues (i.e. version or lib.loc mismatch) between loaded packages and installed packages.

The import system also includes general helper functions:

- `help.import`:
 Get help file for imported objects.
- The `pkg` - functions:
 General helper functions regarding packages.
- The `searchenv` - functions:
 safe functions for safely adding, removing, and accessing custom search path environments.

See the examples section below to get an idea of how the 'tinycodet' import system works in practice. More examples can be found on the website (<https://tony-aw.github.io/tinycodet/>)

Details

When to Use or Not to Use the 'tinycodet' Import System

The 'tinycodet' import system is helpful particularly for packages that have at least one of the following properties:

- The namespace of the package(s) conflicts with other packages.
- The namespace of the package(s) conflicts with core R, or with those of recommended R packages.
- The package(s) have function names that are generic enough, such that it is not obvious which function came from which package.

See examples below.

There is no necessity for using the 'tinycodet' import system with every single package. One can safely attach the 'stringi' package, for example, as 'stringi' uses a unique and immediately recognisable naming scheme (virtually all 'stringi' functions start with "stri_"), and this naming scheme does not conflict with core R, nor with most other packages.

Of course, if one wishes to use a package (like 'stringi') **only** within a specific environment, it becomes advantageous to still import the package using the 'tinycodet' import system.

Some Additional Comments on the 'tinycodet' Import System

- Methods (like S3, S4) will automatically be registered.
- Pronouns, such as the .data and .env pronouns from the 'rlang' package, will work without any prefixes required.
- 'tinycodet' avoids the [exists](#) function, to prevent memory leakage.

For R Package Developers

It goes without saying, just like one should NEVER use `library()` or `require()` inside an R-package, similarly, one should NOT use tinycodet's import functions inside an R-Package. The import functions can still be used inside functions defined in a file to be sourced, though. Just not in functions inside an R-package.

See Also

[tinycodet_help](#)

Examples

```

all(c("dplyr", "tibble", "powerjoin", "magrittr") %installed in% .libPaths())

# import dplyr, tibble, and powerjoin, under aliases:
import_as(.dpr ~ dplyr, deps = "tibble")
import_as(.pj ~ powerjoin)

# attaching only the infix operators from 'magrittr':
library(magrittr, include.only = import_ls("magrittr", "infix") )

# directly assigning dplyr's "starwars" dataset to object "d":
d <- import_data("dplyr", "starwars")

# See it in Action:
d %>% .dpr$filter(species == "Droid") %>%
  .dpr$select(name, .dpr$ends_with("color"))

male_penguins <- .dpr$tribble(
  ~name, ~species, ~island, ~flipper_length_mm, ~body_mass_g,
  "Giordan", "Gentoo", "Biscoe", 222L, 5250L,
  "Lynden", "Adelie", "Torgersen", 190L, 3900L,
  "Reiner", "Adelie", "Dream", 185L, 3650L
)

female_penguins <- .dpr$tribble(
  ~name, ~species, ~island, ~flipper_length_mm, ~body_mass_g,
  "Alonda", "Gentoo", "Biscoe", 211, 4500L,
  "Ola", "Adelie", "Dream", 190, 3600L,
  "Mishayla", "Gentoo", "Biscoe", 215, 4750L,
)
.pj$check_specs()

.pj$power_inner_join(
  male_penguins[c("species", "island")],
  female_penguins[c("species", "island")]
)

mypaste <- function(x, y) {
  import_from("stringi", "stri_c")
  stri_c(x, y)
}
mypaste("hello ", "world")

```

Description

R's numerical functions are generally very fast. But R's native string functions are somewhat slow, do not have a unified naming scheme, and are not as comprehensive as R's impressive numerical functions.

The primary R-package that fixes this is 'stringi', which many, if not most, string related packages depend on (see the list of reverse-dependencies on CRAN).

As string manipulation is important to programming languages, even those primarily focused on mathematics, 'tinycodet' adds a little bit new functionality to 'stringi'.

'tinycodet' adds the following functions to extend 'stringi':

- Find i^{th} pattern occurrence ([stri_locate_ith](#)), or i^{th} text boundary ([stri_locate_ith_boundaries](#)).
- [Concatenate a character matrix row- or column-wise](#).

'tinycodet' adds the following operators, to complement the already existing 'stringi' operators:

- Infix operators for [string arithmetic](#).
- Infix operators for [string sub-setting](#), which get or remove the first and/or last n characters from strings.
- Infix operators for [detecting patterns](#), and [strfind\(\)<-](#) for locating/extracting/replacing found patterns.

And finally, 'tinycodet' adds the somewhat separate [strcut_-functions](#), to cut strings into pieces without removing the delimiters.

Regarding Vector Recycling in the 'stringi'-based Functions

Generally speaking, vector recycling is supported as 'stringi' itself supports it also.

There are, however, a few exceptions.

First, matrix inputs (like in [strcut_loc](#) and [string sub-setting operators](#)) will generally not be recycled.

Second, the i argument in [stri_locate_ith](#) does not support vector recycling.

Scalar recycling is virtually always supported.

References

Gagolewski M., **stringi**: Fast and portable character string processing in R, *Journal of Statistical Software* 103(2), 2022, 1–59, [doi:10.18637/jss.v103.i02](https://doi.org/10.18637/jss.v103.i02)

See Also

[tinycodet_help](#), [s_pattern](#)

Examples

```
# character vector:
x <- c("3rd 1st 2nd", "5th 4th 6th")
print(x)

# detect if there are digits:
x %s{}% "\\d"

# find second last digit:
loc <- stri_locate_ith(x, i = -2, regex = "\\d")
stringi::stri_sub(x, from = loc)

# cut x into matrix of individual words:
mat <- strcut_brk(x, "word")

# sort rows of matrix using the fast %row~% operator:
rank <- stringi::stri_rank(as.vector(mat)) |> matrix(ncol = ncol(mat))
sorted <- mat %row~% rank
sorted[is.na(sorted)] <- ""

# join elements of every row into a single character vector:
stri_c_mat(sorted, margin = 1, sep = " ")
```

aaa4_tinycodet_dry	<i>Overview of the 'tinycodet' "Don't Repeat Yourself" Functionality</i>
--------------------	--

Description

"Don't Repeat Yourself", sometimes abbreviated as "DRY", is the coding principle not to write unnecessarily repetitive code. To help in that effort, the 'tinycodet' R-package introduces a few features:

- The [transform_if](#) function.
- [Operators](#) for short-hand re-ordering matrices Row- or Column-wise.

See Also

[tinycodet_help](#)

Examples

```
object <- matrix(c(-9:8, NA, NA) , ncol=2)

# in base R:
ifelse( # repetitive, and gives unnecessary warning
  is.na(object > 0), -Inf,
  ifelse(
    object > 0, log(object), object^2
```

```
)
)

# with tinycodet:
object |> transform_if(\(x) x > 0, log, \(x) x^2, \(x) -Inf) # compact & no warning
```

decimal_truth

Safer Decimal Number (In)Equality Testing Operators

Description

The `%d==%`, `%d!=%`, `%d<%`, `%d>%`, `%d<=%`, `%d>=%` (in)equality operators perform decimal (type "double") number truth testing.

They are virtually equivalent to the regular (in)equality operators,

`==`, `!=`, `<`, `>`, `<=`, `>=`,

except for 2 aspects:

1. The decimal number (in)equality operators assume that if the absolute difference between any 2 numbers x and y is smaller than the Machine tolerance, `sqrt(.Machine$double.eps)`, then x and y should be consider to be equal.
For example: `(0.1 * 7) == 0.7` returns `FALSE`, even though they are equal, due to the way decimal numbers are stored in programming languages like 'R' and 'Python'.
But `(0.1 * 7) %d==% 0.7` returns `TRUE`.
2. Only numeric input is allowed, so characters are not coerced to numbers.
I.e. `1 < "a"` gives `TRUE`, whereas `1 %d<% "a"` gives an error.
For character equality testing, see [%s==%](#) from the 'stringi' package.

Thus these operators provide safer decimal number (in)equality tests.

There are also the `x %d{ }% bnd` and `x %d!{ }% bnd` operators, where `bnd` is a vector of length 2, or a 2-column matrix (`nrow(bnd)==length(x)` or `nrow(bnd)==1`).

The `x %d{ }% bnd` operator checks if x is within the closed interval with bounds defined by `bnd`.

The `x %d!{ }% bnd` operator checks if x is outside the closed interval with bounds defined by `bnd`.

Moreover, the function `is_wholenumber()` is added, to safely test for whole numbers.

Usage

```
x %d==% y
```

```
x %d!=% y
```

```
x %d<% y
```

```

x %d>% y
x %d<=% y
x %d>=% y
x %d{%}% bnd
x %d!{%}% bnd

is_wholenumber(x, tol = sqrt(.Machine$double.eps))

```

Arguments

x, y	numeric vectors, matrices, or arrays.
bnd	either a vector of length 2, or a matrix with 2 columns and 1 row, or else a matrix with 2 columns where <code>nrow(bnd)==length(x)</code> (or can be recycled to be <code>nrow(bnd)==length(x)</code>). The first element/column of bnd gives the lower bound of the closed interval; The second element/column of bnd gives the upper bound of the closed interval.
tol	a single, strictly positive number close to zero, giving the tolerance.

Details

The operators described in this page are defined in terms of the existing base [Logic](#) operators, and should therefore be compatible with (S3) classes that have method dispatches defined for relational operators.

Value

For the %d...% operators:

A logical vector with the same dimensions as x, indicating the result of the element by element comparison.

NOTE: Inf by Inf and -Inf by -Inf comparisons with the %d...% operators return NA.

For is_wholenumber():

A logical vector with the same dimensions as x, indicating the result of the element by element comparison.

NOTE: Inf, -Inf, NaN and NA all return NA for is_wholenumber().

See Also

[tinycodet_safer](#)

Examples

```

x <- c(0.3, 0.6, 0.7)
y <- c(0.1 * 3, 0.1 * 6, 0.1 * 7)
print(x)
print(y)

x == y # gives FALSE, but should be TRUE
x != y # gives TRUE, should be FALSE
x > y # not wrong
x < y # gives TRUE, should be FALSE

# same as above, but here the results are correct:
x %d==% y # correct
x %d!=% y # correct
x %d<% y # correct
x %d>% y # correct
x %d<=% y # correct
x %d>=% y # correct

# check if numbers are in closed interval:
x <- c(0.3, 0.6, 0.7)
bnd <- cbind(x - 0.1, x + 0.1)
x %d{%}% bnd
x %d!{%}% bnd

# These operators work for integers also:
x <- 1L:5L
y <- 1L:5L
x %d==% y
x %d!=% y
x %d<% y
x %d>% y
x %d<=% y
x %d>=% y

x <- 1L:5L
y <- x + 1L
x %d==% y
x %d!=% y
x %d<% y
x %d>% y
x %d<=% y
x %d>=% y

x <- 1L:5L
y <- x - 1L
x %d==% y
x %d!=% y
x %d<% y
x %d>% y
x %d<=% y
x %d>=% y

```

```
# is_wholenumber:
is_wholenumber(1:10 + c(0, 0.1))
```

import_as

Import R-package (and Minimal Dependencies) Under an Alias

Description

The `import_as()` function imports the namespace of an R-package, and optionally also its direct minimal dependencies, all under the same alias. The specified alias, containing the exported functions from the specified packages, will be placed in the specified environment.

Usage

```
import_as(
  main,
  re_exports = TRUE,
  deps = NULL,
  lib.loc = .libPaths(),
  env = NULL
)
```

Arguments

- | | |
|-------------------------|---|
| <code>main</code> | <p>a 2-sided formula (or a string that evaluates as a 2-sided formula), where the left-hand side gives the alias, and the right-hand side gives the main package to import under the given alias.</p> <p>Starting the alias name with a dot will hide it from <code>ls()</code>, preventing accidental removal.</p> <p>For example :</p> <p><code>.alias ~ packagename</code></p> <p>Core R (i.e. "base", "stats", etc.) is not allowed for the main package.</p> |
| <code>re_exports</code> | <p>TRUE or FALSE.</p> <ul style="list-style-type: none"> • If <code>re_exports = TRUE</code> the re-exports from the main package (including those exported from Core R) are added to the alias together with the main package.
This is the default, as it is analogous to the behaviour of base R's <code>::</code> operator. • If <code>re_exports = FALSE</code>, these re-exports are not added together with the main package.
The user can still import the packages under the alias from which the re-exported functions came from, by specifying them in the dependencies argument. |

deps	an optional character vector, giving the names of the dependencies of the main package to be imported also under the alias. Only dependencies that appear in pkg_get_deps_minimal are allowed. Defaults to NULL, which means no dependencies are imported under the alias. Core R (i.e. "base", "stats", etc.) is not allowed.
lib.loc	a character vector describing the location of R library trees to search through.
env	see import_env .

Value

A locked environment object, similar to the output of [loadNamespace](#), with the name as specified in the `alias` argument, will be created.

This object, of class [tinyimport_alias](#), will contain the exported functions from the specified package(s).

The alias object will be placed in the specified environment.

For its usage, see [tinyimport_alias](#).

Note the following:

- No more than 5 packages (ignoring re-exports) are allowed to be imported under a single alias.
- Packages are imported in the following order:
First the dependencies in the order they are specified in `deps`, and then the main package, and then its re-exports (if `re_exports = TRUE`).
Thus main package will always overwrite the dependencies in case of conflicting names.

Why Aliasing A Package with its Dependencies is Useful

To use an R-package with its dependencies, whilst avoiding the disadvantages of attaching a package (see [tinycodet_import](#)), one would traditionally use the `::` operator like so:

```
main_package::some_function1()
dependency1::some_function2()
```

This becomes cumbersome as more packages are needed and/or as the package name(s) become longer.

The `import_as()` function avoids this issue by allowing multiple **related** packages to be imported under a single alias, allowing one to code like this:

```
import_as(.alias ~ main_package, deps = "dependency1")
.alias$some_function1()
.alias$some_function2()
```

Thus importing a package, or multiple directly related packages, under a single alias, which `import_as()` provides, avoids the above issues. Importing a package under an alias is referred to as "aliasing" a

package.

See Also

[tinycodet_import](#)

Examples

```
import_as(.stri ~ stringi)
.stri$stri_join("a", "b")

import_from("stringi", import_ls("stringi", "infix"))
"a" %s+% "b"

attr.import(.stri)
```

import_data

Directly Return a Data-set From a Package

Description

The `import_data()` function gets a specified data set from a package. Unlike `utils::data()`, the `import_data()` function returns the data set directly, and allows assigning the data set like so:

```
mydata <- import_data(...).
```

Usage

```
import_data(package, dataname, lib.loc = .libPaths())
```

Arguments

<code>package</code>	a single string, giving the name of the R-package.
<code>dataname</code>	a single string, giving the name of the data set.
<code>lib.loc</code>	a character vector describing the location of R library trees to search through.

Value

Returns the data directly. Thus, one can assign the data like so: `mydata <- import_data(...)`.

See Also

[tinycodet_import](#)

Examples

```
d <- import_data("datasets", "cars")
head(d)
```

import_diagnose

Check for Mismatches between Loaded and Installed Packages

Description

The `import_diagnose()` function compares the loaded packages with those installed in the specified `lib.loc`, and checks for version and library path mismatches. Any differences found will be reported in the form of a simple `data.frame`.

Usage

```
import_diagnose(lib.loc = .libPaths())
```

Arguments

`lib.loc` a character vector describing the location of R library trees to search through.

Value

If no issues are found, returns `NULL`.

Otherwise, a `data.frame` giving the packages where issues have been found.

This `data.frame` will have the following columns:

- "package": Character vector of package names
- "installed_in_lib.loc": logical vector indicating if the package is actually installed in the given `lib.loc` paths (`TRUE`) or not (`FALSE`).
- "version_loaded": character vector giving the version of the packages as loaded in [loaded-Namespaces](#).
- "version_installed": character vector giving the version of the packages as installed in `lib.loc`; It will be `NA` if the package is not installed in `lib.loc`.
- "versions_equal": logical vector indicating if the loaded and installed versions match. Gives `NA` if the package is not installed in `lib.loc`.

See Also[tinycodet_import](#)**Examples**

```
import_diagnose()
```

`import_env`*Environment Specification in the import_ Functions*

Description

The `env` argument in the `import_` functions specify where the functions, exported objects, or alias object will be placed.

The following can be specified for `env`:

- `NULL`: If `env = NULL`, the objects will be placed in the caller environment.
- an environment.
- a string, giving the name of the search path, as given by [search](#), to place the objects in. If multiple search paths have the specified name, an error is returned.
- a number larger than 1 and smaller than `length(search())`, giving the position of the search path to place the objects in.

If `env` is a string or number, and thus points to a place in the search path, the search path environment is not allowed to be any of the following:

- a package path (their names start with 'package:')
- a tools path (their names start with 'tools:')
- the Global environment ('.GlobalEnv')
- autoloads search path ('Autoloads')
- a path at position 1 or `length(search())`

Attempting to use such a path results in an error.

The user can use the [searchenv_](#) functions, provided by 'tinycodet', to safely add or remove custom search paths.

The default value for `env` is `NULL`.

See Also[tinycodet_import](#)**Examples**

```

search()
searchenv_add("my_ops")
search()

exports <- import_ls("stringi", c("infix", "rp"))
import_from("stringi", exports, env = "my_ops")

foo <- searchenv_get("my_ops")
all(exports %in% names(foo))

search()
searchenv_rm("my_ops")
search()

```

import_from

*Expose Exported Objects From Package Namespace in an Environment***Description**

import_from() exposes exported objects from a package to the specified environment.

Usage

```

import_from(
  package,
  ls,
  re_exports = TRUE,
  lock = FALSE,
  prefix = NULL,
  lib.loc = .libPaths(),
  env = NULL
)

```

Arguments

package	a single string, giving the package name.
ls	a character vector giving the names of exported objects to expose. To expose all objects of a certain type, like all (infix/replacement) operators or all non-functions, the import_ls function can be used to get those object names.

re_exports	TRUE or FALSE, indicating if re-exports of package should be included. Default is TRUE, as that is analogous to the behaviour of base R's <code>::</code> operator.
lock	TRUE or FALSE, indicating if the exported objects should be locked (see lock-Binding).
prefix	either NULL or a single string, giving the prefix to add to the functions, to avoid possible conflicts. If NULL no prefix is added; this is the default. Otherwise, a prefix is added to the exported object names. Note that no prefix is added to infix operators, primitive functions, and non-functions (like constants). The prefix cannot end with a single dot, as that might interfere with S3 methods.
lib.loc	a character vector describing the location of R library trees to search through.
env	see import_env .

Value

The objects specified in the given package will be placed & locked in the specified environment.

See Also

[tinycodet_import](#), [import_ls\(\)](#)

Examples

```
import_from("stringi", import_ls("stringi", "infix"))
import_from("stringi", import_ls("stringi", "rp"))
```

import_helper

Helper Functions for the 'tinycodet' Package Import System

Description

The `help.import()` function finds the help file for functions or topics, including exposed functions/operators as well as functions in a package alias object.

Usage

```
help.import(..., i, alias)
```

Arguments

<code>...</code>	further arguments to be passed to help .
<code>i</code>	either one of the following: - a function (use back-ticks when the function is an infix/replacement operator). Examples: <code>myfun, `%operator%`, `fun<-`, myalias.\$some_function.</code> If a function, the alias argument is ignored. - a string giving the function name or topic (i.e. "myfun", "thistopic"). If a string, argument alias must be specified also.
<code>alias</code>	an object of class tinyimport_alias as returned by import_as .

Details

For `help.import(...)`:

Do not use the topic / package and i / alias argument sets together. It's either one set or the other.

For example:

```
import_as(.str ~ stringi)
import_from("magrittr", import_ls("magrittr", "infix"))
help.import(i = .str$stri_sub)
help.import(i = `%>%`)
help.import(i = "stri_sub", alias = .str)
help.import(topic = "%>%", package = "magrittr")
help.import("%>%", package = "magrittr") # same as previous line
```

Value

For `help.import()`:

Opens the appropriate help page.

See Also

[tinycodet_import](#)

Examples

```
import_as(.stri ~ stringi)
.stri$stri_join("a", "b")

import_from("stringi", import_ls("stringi", "infix"))
"a" %s+% "b"

attr.import(.stri)
```

import_legacy	<i>Legacy import_functions</i>
---------------	--------------------------------

Description

These functions are deprecated, and will be removed in a future update.

Usage

```
import_inops(expose, lib.loc = .libPaths(), ...)  
  
import_LL(package, selection, lib.loc = .libPaths())
```

Arguments

<code>expose, package</code>	a single string, giving the name of the R-package.
<code>lib.loc</code>	a character vector describing the location of R library trees to search through.
<code>...</code>	further arguments passed to import_from or import_ls
<code>selection</code>	a character vector of function names (both regular functions and infix operators). Internal functions or re-exported functions are not supported.

Value

See [import_from](#).

See Also

[tinycodet_import](#), [import_from\(\)](#), [import_ls\(\)](#)

Examples

```
import_inops("stringi")  
import_LL("stringi", "stri_c")
```

import_ls

*List Exported Objects from Package Namespace***Description**

Lists exported objects defined (or re-exported) in a package namespace.

Note that `import_ls()` necessary loads the package, but does not attach the package.

The return value of `import_ls()` is for programmatically dynamic code (see section Value).

The side-effect of `import_ls()` is for syntactically readable code (see section Side Effect).

Usage

```
import_ls(
  package,
  types = c("reg", "infix", "rp", "nonfun"),
  re_exports = TRUE,
  lib.loc = .libPaths(),
  print = TRUE
)
```

Arguments

package	a single string, giving the package name.
types	a character vector, specifying the type. The following types are supported: • "reg": regular functions. • "infix": infix operators. • "rp": replacement operators, including their base functions. • "nonfun": non-functions.
re_exports	TRUE or FALSE, indicating if re-exports of package should be included. Default is TRUE, as that is analogous to the behaviour of base R's <code>::</code> operator.
lib.loc	a character vector describing the location of R library trees to search through.
print	TRUE (default) or FALSE, indicating if the exported objects should be printed to the console as literal code. Note that this is a side-effect , and does not impact the storable return value of <code>import_ls()</code> .

Value

A character vector of exported object names defined in the package.
 Can be used programmatically in [library](#) or [import_from](#) functions.
 I.e.:

Like so:

```
ls <- import_ls("packagename", "infix")
library(packagename, include.only = ls)
```

Or like so:

```
ls <- import_ls("packagename", "infix")
import_from(packagename, ls = ls)
```

Side Effect

(if print = TRUE)

The returned character vector is printed to your console as literal code.

I.e. 'c("obj1", "obj2")'.

One can then copy-paste the printed literal code, for syntactical clarity.

I.e.:

Like so:

```
import_ls("packagename", "infix") # prints literal code to your console
library(packagename, include.only = ...paste printed literal code here...)
```

Or like so:

```
import_ls("packagename", "infix") # prints literal code to your console
import_from("packagename", ls = ...paste printed literal code here...)
```

Why Listing Functions by Type is Useful

One can import a package under an alias using [import_as](#).

But using infix operators or replacement operators from an alias requires convoluted code like so:

```
.alias$`%op%`(x, y)
.alias$`fun<-`(x, ..., value)
```

Instead, it would be easier to just attach (via [library](#)) or expose (via [import_from](#)) such operators so that they can be used on their own.

The `import_ls()` function allows the user to get a list of all functions from a certain type, like "infix operators" or "replacement operators".

The listed functions can then be passed to [library](#) (to attach them) or [import_from](#) (to expose them).

See Also

[tinycodet_import](#), [import_from](#), [library](#)

Examples

```
# Programmatically Dynamic Code ====

ls <- import_ls("stringi", "infix")
import_from("stringi", ls)

if("bit64" %installed in% .libPaths()) {
  ls <- import_ls("bit64", "nonfun")
  import_from("bit64", ls)
}

# Syntactically Readable Code ====

import_ls("stringi", "rp") # copy-pasted the printed literal code
import_from(
  "stringi",
  ls = c("stri_datetime_add", "stri_datetime_add<-", "stri_sub", "stri_sub_all",
        "stri_sub_all<-", "stri_sub<-", "stri_subset", "stri_subset_charclass",
        "stri_subset_charclass<-", "stri_subset_coll", "stri_subset_coll<-",
        "stri_subset_fixed", "stri_subset_fixed<-", "stri_subset_regex",
        "stri_subset_regex<-", "stri_subset<-")
)
```

lock

Lock T, Lock F, or Create Locked Constants

Description

The `lock_TF()` function locks the T and F values and sets them to TRUE and FALSE, respectively, to prevent the user from re-assigning them.

Removing the created T and F objects allows re-assignment again.

The `X %<-c% A` operator creates a constant X and assigns A to it.

Constants cannot be changed, only accessed or removed. So if you have a piece of code that requires some unchangeable constant, use this operator to create said constant.

Removing constant X also removes its binding lock. Thus to change a constant, simply remove it and re-create it.

Usage

```
lock_TF(env)
```

```
X %<-c% A
```

Arguments

env	an optional environment to give, determining in which environment T and F should be locked. When not specified, the current environment is used.
X	a syntactically valid unquoted name of the object to be created.
A	any kind of object to be assigned to X.

Details

Note that following statement

```
x %<-c% 2+2  
print(x)
```

returns

```
[1] 2
```

due to R's precedence rules. Therefore, in such cases, the right hand side of `X %<-c% A` need to be surrounded with brackets. I.e.:

```
x %<-c% (2 + 2)
```

Note that the `lock_TF()` function and `%s<-c%` operator create constants through [lockBinding](#).

The constants are protected from modification by copy, but they are **not** protected from modification by reference (see for example `collapse::setv`).

Value

For `lock_TF()`:

Two constants, namely T and F, set to TRUE and FALSE respectively, are created in the specified or else current environment, and locked. Removing the created T and F objects allows re-assignment again.

For `X %<-c% A`:

The object X containing A is created in the current environment, and this object cannot be changed. It can only be accessed or removed.

See Also

[tinycodet_safer](#)

Examples

```
lock_TF()  
X %<-c% data.frame(x = 3, y = 2) # this data.frame cannot be changed. Only accessed or removed.  
X[1, ,drop=FALSE]
```

matrix_ops	<i>Row- or Column-wise Re-ordering of Matrices</i>
------------	--

Description

Infix operators for custom row- and column-wise re-ordering of matrices.

The `x %row~% mat` operator re-orders the elements of every row, each row ordered independently from the other rows, of matrix `x`, according to the ordering ranks given in matrix `mat`.

The `x %col~% mat` operator re-orders the elements of every column, each column ordered independently from the other columns, of matrix `x`, according to the ordering ranks given in matrix `mat`.

Note that these operators strip all attributes, except dimensions.

Usage

```
x %row~% mat  
  
x %col~% mat
```

Arguments

<code>x</code>	a matrix
<code>mat</code>	a numeric matrix with the same dimensions as <code>x</code> , giving the new ordering ranks for every element of matrix <code>x</code> .

Value

A re-ordered matrix.

See Also

[tinycodet_dry](#)

Examples

```
# numeric matrix ====

x <- matrix(sample(1:25), nrow = 5)
print(x)
x %row% x # sort elements of every row independently
x %row% -x # reverse-sort elements of every row independently
x %col% x # sort elements of every column independently
x %col% -x # reverse-sort elements of every column independently

x <- matrix(sample(1:25), nrow = 5)
print(x)
mat <- sample(seq_along(x)) |> matrix(ncol = ncol(x))
x %row% mat # randomly shuffle every row independently
x %col% mat # randomly shuffle every column independently

# character matrix ====

x <- matrix(sample(letters, 25), nrow = 5)
print(x)
mat <- stringi::stri_rank(as.vector(x)) |> matrix(ncol = ncol(x))
x %row% mat # sort elements of every row independently
x %row% -mat # reverse-sort elements of every row independently
x %col% mat # sort elements of every column independently
x %col% -mat # reverse-sort elements of every column independently

x <- matrix(sample(letters, 25), nrow = 5)
print(x)
mat <- sample(seq_along(x)) |> matrix(ncol = ncol(x))
x %row% mat # randomly shuffle every row independently
x %col% mat # randomise shuffle every column independently
```

pkgs

Miscellaneous Package Related Functions

Description

The `pkgs %installed in% lib.loc` operator checks if one or more given packages (`pkgs`) exist in the given library paths (`lib.loc`), without loading the packages at all.

The syntax of this operator forces the user to make it syntactically explicit where to look for installed R-packages.

As `pkgs %installed in% lib.loc` does not even load a package, the user can safely use it without fearing any unwanted side-effects.

The `pkg_get_deps()` function gets the **direct** dependencies of a package from the Description file. It works on non-CRAN packages also.

The `pkg_get_deps_minimal()` function is the same as `pkg_get_deps()`, except with `base`, `recom`,

semi, shared_tidy all set to FALSE, and the default value for deps_type is c("Depends", "Imports").

Usage

```
pkgs %installed in% lib.loc

pkg_get_deps(
  package,
  lib.loc = .libPaths(),
  deps_type = c("LinkingTo", "Depends", "Imports"),
  base = FALSE,
  recom = TRUE,
  semi = TRUE,
  shared_tidy = TRUE
)

pkg_get_deps_minimal(
  package,
  lib.loc = .libPaths(),
  deps_type = c("Depends", "Imports")
)
```

Arguments

pkgs	a character vector with the package name(s).
lib.loc	character vector specifying library search path (the location of R library trees to search through). The lib.loc argument would usually be .libPaths(). See also loadNamespace . For pkg_get_deps() and pkg_get_deps_minimal(): lib.loc can also be set to NULL. In that case the loaded namespace of the package will be checked instead of the installed packages at lib.loc. If lib.loc is NULL, but the package is not loaded, an error is returned.
package	a single string giving the package name.
deps_type	a character vector, giving the dependency types to be used. The order of the character vector given in deps_type affects the order of the returned character vector; see Details sections.
base	TRUE or FALSE, indicating whether base/core R should be included (TRUE), or not included (FALSE).
recom	TRUE or FALSE, indicating whether the pre-installed 'recommended' R-packages should be included (TRUE), or not included (FALSE).
semi	TRUE or FALSE, indicating whether semi-ingrained R-packages ('S7', 'rstudioapi') should be included (TRUE) or not include (FALSE).

`shared_tidy` TRUE or FALSE, indicating whether the following packages should be included (TRUE) or not included (FALSE):
'rlang', 'lifecycle', 'cli', 'glue', and 'withr'.

Details

For `pkg_get_deps()`:

For each string in argument `deps_type`, the package names in the corresponding field of the Description file are extracted, in the order as they appear in that field.

The order given in argument `deps_type` also affects the order of the returned character vector:

For example, `c("LinkingTo", "Depends", "Imports")`,

means the package names are extracted from the fields in the following order:

1. "LinkingTo";
2. "Depends";
3. "Imports".

The unique (thus non-repeating) package names are then returned to the user.

Value

For `pkgs %installed in% lib.loc`:

Returns a named logical vector.

The names give the package names.

The value TRUE indicates a package is installed in `lib.loc`.

The value FALSE indicates a package is not installed in `lib.loc`.

The value NA indicates a package is not actually a separate package, but base/core 'R' (i.e. 'base', 'stats', etc.).

For `pkg_get_deps()` and `pkg_get_deps_minimal()`:

A character vector of direct dependencies, without duplicates.

References

O'Brien J., elegantly extract R-package dependencies of a package not listed on CRAN. *Stack Overflow*. (1 September 2023).

See Also

[tinycodet_import](#)

Examples

```
"dplyr" %installed in% .libPaths()

pkg_get_deps_minimal("dplyr")
pkgs <- pkg_get_deps("dplyr")
pkgs %installed in% .libPaths()
```

safer_partialmatch	<i>Set Safer Dollar, Arguments, and Attribute Matching</i>
--------------------	--

Description

The `safer_partialmatch()` function simply calls the following:

```
options(
  warnPartialMatchDollar = TRUE,
  warnPartialMatchArgs = TRUE,
  warnPartialMatchAttr = TRUE
)
```

Thus it forces 'R' to give a warning when partial matching occurs when using the dollar (\$) operator, or when other forms of partial matching occurs.

The `safer_partialmatch()` function is intended for when running R interactively (see [interactive](#)).

Usage

```
safer_partialmatch()
```

Value

Sets the options. Returns nothing.

See Also

[tinycodet_safer](#)

Examples

```
interactive()

safer_partialmatch()
data(iris)
head(iris)
iris$Sepal.Length <- iris$Sepal.Length^2
head(iris)
```

searchenv

Add, Remove, or Access Search Path Environments

Description

Functions for safely adding (`searchenv_add()`), removing (`searchenv_rm()`), or accessing (`searchenv_get()`) environments from the [search](#) path.

Usage

```
searchenv_add(name, pos = 2L)
```

```
searchenv_rm(name, pos)
```

```
searchenv_get(name, pos)
```

Arguments

name	a single string giving the name for the environment in the search path.
pos	a single positive integer giving the position for the environment in the search path.

Details

These functions were designed with safety in mind.

They do not allow adding, removing, or accessing search environments like the following:

- a package path (their names start with 'package:')
- a tools path (their names start with 'tools:')
- the Global environment ('.GlobalEnv')
- autoloader search path ('Autoloads')
- a path at position 1 or `length(search())`

Attempting to add a new search path environment whose name already exists gives an error.
Attempting to remove or access a search path environment whose name does not exist gives an error.

Value

`searchenv_add()` adds a new, empty environment to the [search](#) path;
returns nothing.

`searchenv_rm()` removes a (user-defined) environment from the [search](#) path;
returns nothing.

`searchenv_get()` returns a (user-defined) environment from the [search](#) path.

See Also

[tinycodet_import](#), [search](#)

Examples

```
search()
searchenv_add("my_ops")
search()

exports <- import_ls("stringi", c("infix", "rp"))
import_from("stringi", exports, env = "my_ops")

foo <- searchenv_get("my_ops")
all(exports %in% names(foo))

search()
searchenv_rm("my_ops")
search()
```

strcut_loc

Cut Strings

Description

The `strcut_loc()` function cuts every string in a character vector around a location range `loc`, such that every string is cut into the following parts:

- the sub-string **before** `loc`;
- the sub-string at `loc` itself;

- the sub-string **after** loc.

The location range loc would usually be matrix with 2 columns, giving the start and end points of some pattern match.

The strcut_brk() function (a wrapper around [stri_split_boundaries\(..., tokens_only = FALSE\)](#)) cuts every string into individual text breaks (like character, word, line, or sentence boundaries).

Usage

```
strcut_loc(str, loc)
```

```
strcut_brk(str, type = "character", tolist = FALSE, n = -1L, ...)
```

Arguments

str	a string or character vector.
loc	Either one of the following: • the result from the stri_locate_ith function. • a matrix of 2 integer columns, with <code>nrow(loc)==length(str)</code>, giving the location range of the middle part. • a vector of length 2, giving the location range of the middle part.
type	either one of the following: • a single string giving the break iterator type (i.e. "character", "line_break", "sentence", "word", or a custom set of ICU break iteration rules). • a list with break iteration options, like a list produced by stri_opts_brkiter. about search: boundaries
tolist	logical, indicating if strcut_brk should return a list (TRUE), or a matrix (FALSE, default).
n	see stri_split_boundaries .
...	additional arguments to be passed to stri_split_boundaries .

Details

The strcut_ functions provide a short and concise way to cut strings into pieces, without removing the delimiters, which is an operation that lies at the core of virtually all boundaries-operations in 'stringi'.

The main difference between the strcut_ - functions and [stri_split](#) / [strsplit](#), is that the latter generally removes the delimiter patterns in a string when cutting, while the strcut_-functions do not attempt to remove parts of the string by default, they only attempt to cut the strings into separate pieces. Moreover, the strcut_ - functions return a matrix by default.

Value

For `strcut_loc()`:

A character matrix with `length(str)` rows and 3 columns, where for every row `i` it holds the following:

- the first column contains the sub-string **before** `loc[i,]`, or NA if `loc[i,]` contains NA;
- the second column contains the sub_string at `loc[i,]`, or the uncut string if `loc[i,]` contains NA;
- the third and last column contains the sub-string **after** `loc[i,]`, or NA if `loc[i,]` contains NA.

For `strcut_brk(..., tolist = FALSE)`:

A character matrix with `length(str)` rows and a number of columns equal to the maximum number of pieces `str` was cut in.

Empty places are filled with NA.

For `strcut_brk(..., tolist = TRUE)`:

A list with `length(str)` elements, where each element is a character vector containing the cut string.

See Also

[tinycodet_strings](#)

Examples

```
x <- rep(paste0(1:10, collapse = ""), 10)
print(x)
loc <- stri_locate_ith(x, 1:10, fixed = as.character(1:10))
strcut_loc(x, loc)
strcut_loc(x, c(5, 5))
strcut_loc(x, c(NA, NA))
strcut_loc(x, c(5, NA))
strcut_loc(x, c(NA, 5))

test <- "The\u00a0above-mentioned    features are very useful. " %s+%
      "Spam, spam, eggs, bacon, and spam. 123 456 789"
strcut_brk(test, "line")
strcut_brk(test, "word")
strcut_brk(test, "sentence")
strcut_brk(test)
strcut_brk(test, n = 1)
strcut_brk(test, "line", tolist = TRUE)
strcut_brk(test, "word", tolist = TRUE)
strcut_brk(test, "sentence", tolist = TRUE)
```

```
brk <- stringi::stri_opts_brkiter(
  type = "line"
)
strcut_brk(test, brk)
```

stri_join_mat

*Concatenate Character Matrix Row-wise or Column-wise***Description**

The `stri_join_mat()` function (and their aliases `stri_c_mat` and `stri_paste_mat`) perform row-wise (`margin = 1`; the default) or column-wise (`margin = 2`) joining of a matrix of strings, thereby transforming a matrix of strings into a vector of strings.

Usage

```
stri_join_mat(mat, margin = 1, sep = "", collapse = NULL)
```

```
stri_c_mat(mat, margin = 1, sep = "", collapse = NULL)
```

```
stri_paste_mat(mat, margin = 1, sep = "", collapse = NULL)
```

Arguments

`mat` a matrix of strings

`margin` the margin over which the strings must be joined.

- If `margin = 1`, the elements within each row of matrix `mat` are joined into a single string. Thus if the matrix has 10 rows, it returns a vector of 10 strings.
- If `margin = 2`, the elements within each column of matrix `mat` are joined into a single string. Thus if the matrix has 10 columns, it returns a vector of 10 strings.

`sep, collapse` as in [stri_join](#).

Value

The `stri_join_mat()` function, and its aliases, return a vector of strings.

See Also

[tinycodet_strings](#)

Examples

```
#####

# Basic example

x <- matrix(letters[1:25], ncol = 5, byrow = TRUE)
print(x)
stri_join_mat(x, margin = 1)

x <- matrix(letters[1:25], ncol = 5, byrow = FALSE)
print(x)
stri_join_mat(x, margin = 2)

#####
# sorting characters in strings ====

x <- c(paste(sample(letters), collapse = ""),
      paste(sample(letters), collapse = ""))
print(x)
mat <- strcut_brk(x)
rank <- stringi::stri_rank(as.vector(mat)) |> matrix(ncol = ncol(mat))
sorted <- mat %row~% rank
sorted[is.na(sorted)] <- ""
print(sorted)
stri_join_mat(sorted, margin = 1)
stri_join_mat(sorted, margin = 2)

#####

# sorting words ====

x <- c("2nd 3rd 1st", "Goodbye everyone")
print(x)
mat <- strcut_brk(x, "word")
rank <- stringi::stri_rank(as.vector(mat)) |> matrix(ncol = ncol(mat))
sorted <- mat %row~% rank
sorted[is.na(sorted)] <- ""
stri_c_mat(sorted, margin = 1, sep = " ") # <- alias for stri_join_mat
stri_c_mat(sorted, margin = 2, sep = " ")

#####

# randomly shuffling sentences ====

x <- c("Hello, who are you? Oh, really?! Cool!",
      "I don't care. But I really don't.")
print(x)
mat <- strcut_brk(x, "sentence")
rank <- sample(seq_along(mat)) |> matrix(ncol = ncol(mat))
```

```
sorted <- mat %row~% rank
sorted[is.na(sorted)] <- ""
stri_paste_mat(sorted, margin = 1) # <- another alias for stri_join_mat
stri_paste_mat(sorted, margin = 2)
```

stri_locate_ith	<i>Locate i^{th} Pattern Occurrence or Text Boundary</i>
-----------------	---

Description

The `stri_locate_ith()` function locates the i^{th} occurrence of a pattern in each string of some character vector.

The `stri_locate_ith_boundaries()` function locates the i^{th} text boundary (like character, word, line, or sentence boundaries).

Usage

```
stri_locate_ith(str, i, ..., regex, fixed, coll, charclass)
stri_locate_ith_regex(str, pattern, i, ..., opts_regex = NULL)
stri_locate_ith_fixed(str, pattern, i, ..., opts_fixed = NULL)
stri_locate_ith_coll(str, pattern, i, ..., opts_collator = NULL)
stri_locate_ith_charclass(str, pattern, i, merge = TRUE, ...)
stri_locate_ith_boundaries(str, i, ..., opts_brkiter = NULL)
```

Arguments

<code>str</code>	a string or character vector.
<code>i</code>	an integer scalar, or an integer vector of appropriate length (vector recycling is not supported). Positive numbers count occurrences from the left/beginning of the strings. Negative numbers count occurrences from the right/end of the strings. I.e.: • <code>stri_locate_ith(str, i = 1, ...)</code> gives the position (range) of the first occurrence of a pattern. • <code>stri_locate_ith(str, i = -1, ...)</code> gives the position (range) of the last occurrence of a pattern. • <code>stri_locate_ith(str, i = 2, ...)</code> gives the position (range) of the second occurrence of a pattern.

	• <code>stri_locate_ith(str, i = -2, ...)</code> gives the position (range) of the second-last occurrence of a pattern. If <code>abs(i)</code> is larger than the number of pattern occurrences <code>n</code>, the first (if <code>i < -n</code>) or last (if <code>i > n</code>) instance will be given. For example: suppose a string has 3 instances of some pattern; then if <code>i >= 3</code> the third instance will be located, and if <code>i <= -3</code> the first instance will be located.
...	more arguments to be supplied to stri_locate_all or stri_locate_all_boundaries. Do not supply the arguments <code>omit_no_match</code> or <code>get_length</code>, as they are already specified internally. Supplying these arguments anyway will result in an error.
pattern, regex, fixed, coll, charclass	a character vector of search patterns, as in stri_locate_all. about search: regex about search: fixed about search: coll about search: charclass
opts_regex, opts_fixed, opts_collator, opts_brkiter	named list used to tune up the selected search engine's settings. see stri_opts_regex, stri_opts_fixed, stri_opts_collator, and stri_opts_brkiter. NULL for the defaults. about search: regex about search: fixed about search: coll about search: charclass about search: boundaries
merge	logical, indicating if charclass locations should be merged or not. Details: For the charclass pattern type, the <code>stri_locate_ith()</code> function gives the start and end of consecutive characters by default, just like stri_locate_all. To give the start and end positions of single characters, much like stri_locate_first or stri_locate_last, set <code>merge = FALSE</code>.

Details

The 'stringi' functions only support operations on the first, last, or all occurrences of a pattern. The `stri_locate_ith()` function allows locating the i^{th} occurrence of a pattern. This allows for several workflows for operating on the i^{th} pattern occurrence. See also the examples section.

Extract i^{th} Occurrence of a Pattern

For extracting the i^{th} pattern occurrence:
Locate the the i^{th} occurrence using `stri_locate_ith()`, and then extract it using, for example, [stri_sub](#).

Replace/Transform i^{th} Occurrence of a Pattern

For replacing/transforming the i^{th} pattern occurrence:

1. Locate the the i^{th} occurrence using `stri_locate_ith()`.
2. Extract the occurrence using [stri_sub](#).
3. Transform or replace the extracted sub-strings.
4. Return the transformed/replaced sub-string back, using again [stri_sub](#).

Capture Groups of i^{th} Occurrence of a Pattern

The `capture_groups` argument for `regex` is not supported within `stri_locate_ith()`.

To capture the groups of the i^{th} occurrences:

1. Use `stri_locate_ith()` to locate the i^{th} occurrences without group capture.
2. Extract the occurrence using [stri_sub](#).
3. Get the matched group capture on the extracted occurrences using [stri_match](#).

Value

The `stri_locate_ith()` function returns an integer matrix with two columns, giving the start and end positions of the i^{th} matches, two NAs if no matches are found, and also two NAs if `str` is NA.

If an empty string or empty pattern is supplied, a warning is given and a matrix with 0 rows is returned.

Note**Long Vectors**

The `stri_locate_ith`-functions do not support long vectors (i.e. character vectors with more than $2^{31} - 1$ strings).

Performance

The performance of `stri_locate_ith()` is about the same as that of [stri_locate_all](#).

See Also

[tinycodet_strings](#)

Examples

```
#####

# practical example: transform regex pattern ====

# input character vector:
x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)

# locate ith (second and second-last) vowel locations:
p <- rep("A|E|I|O|U", 2) # vowels
loc <- stri_locate_ith(x, c(2, -2), regex = p, case_insensitive = TRUE)
print(loc)

# extract ith vowels:
extr <- stringi::stri_sub(x, loc)
print(extr)

# transform & replace ith vowels with numbers:
repl <- chartr("aeiou", "12345", extr)
stringi::stri_sub(x, loc) <- repl

# result (notice ith vowels are now numbers):
print(x)

#####

# practical example: group-capture regex pattern ====

# input character:
# first group: c(breakfast=eggs, breakfast=bacon)
# second group: c(lunch=pizza, lunch=spaghetti)
x <- c('breakfast=eggs;lunch=pizza',
      'breakfast=bacon;lunch=spaghetti',
      'no food here') # no group here
print(x)

# locate ith=2nd group:
p <- '(\w+)= (\w+)'
loc <- stri_locate_ith(x, i = 2, regex = p)
print(loc)

# extract ith=2nd group:
extr <- stringi::stri_sub(x, loc)
print(extr)

# capture ith=2nd group:
stringi::stri_match(extr, regex = p)

#####
```

```

# practical example: replace words using boundaries ====

# input character vector:
x <- c("good morning and good night",
"hello ladies and gentlemen")
print(x)

# report ith word locations:
loc <- stri_locate_ith_boundaries(x, c(-3, 3), type = "word")
print(loc)

# extract ith words:
extr <- stringi::stri_sub(x, from = loc)
print(extr)

# transform and replace words (notice ith words have inverted case):
tf <- chartr(extr, old = "a-zA-Z", new = "A-Za-z")
stringi::stri_sub(x, loc) <- tf

# result:
print(x)

#####

# find pattern ====

extr <- stringi::stri_sub(x, from = loc)
repl <- chartr(extr, old = "a-zA-Z", new = "A-Za-z")
stringi::stri_sub_replace(x, loc, replacement=repl)

#####

# simple pattern ====

x <- rep(paste0(1:10, collapse = ""), 10)
print(x)
out <- stri_locate_ith(x, 1:10, regex = as.character(1:10))
cbind(1:10, out)

x <- c(paste0(letters[1:13], collapse = ""),
paste0(letters[14:26], collapse = ""))
print(x)
p <- rep("a|e|i|o|u", 2)
out <- stri_locate_ith(x, c(-1, 1), regex = p)
print(out)
substr(x, out[, 1], out[, 2])

```

```
#####

# ignore case pattern ====

x <- c(paste0(letters[1:13], collapse = ""),
       paste0(letters[14:26], collapse = ""))
print(x)
p <- rep("A|E|I|O|U", 2)
out <- stri_locate_ith(x, c(1, -1), regex = p, case_insensitive = TRUE)
substr(x, out[, 1], out[, 2])

#####

# multi-character pattern ====

x <- c(paste0(letters[1:13], collapse = ""),
       paste0(letters[14:26], collapse = ""))
print(x)
# multi-character pattern:
p <- rep("AB", 2)
out <- stri_locate_ith(x, c(1, -1), regex = p, case_insensitive = TRUE)
print(out)
substr(x, out[, 1], out[, 2])

#####

# Replacement transformation using stringi ====

x <- c("hello world", "goodbye world")
loc <- stri_locate_ith(x, c(1, -1), regex = "a|e|i|o|u")
extr <- stringi::stri_sub(x, from = loc)
repl <- chartr(extr, old = "a-zA-Z", new = "A-Za-z")
stringi::stri_sub_replace(x, loc, replacement = repl)

#####

# Boundaries ====

test <- c(
  paste0("The\u00a0above-mentioned   features are very useful. ",
        "Spam, spam, eggs, bacon, and spam. 123 456 789"),
  "good morning, good evening, and good night"
)

loc <- stri_locate_ith_boundaries(test, i = c(1, -1), type = "word")
stringi::stri_sub(test, from = loc)
```

str_arithmetic

*String Arithmetic Operators***Description**

String arithmetic operators.

The `x %s+% y` operator is exported from 'stringi', and concatenates character vectors `x` and `y`.

The `x %s-% p` operator removes character/pattern defined in `p` from `x`.

The `x %s*% n` operator is exported from 'stringi', and duplicates each string in `x` `n` times, and concatenates the results.

The `x %s/% p` operator counts how often character/pattern defined in `p` occurs in each element of `x`.

The `x %s//% brk` operator counts how often the text boundary specified in list `brk` occurs in each element of `x`.

The `e1 %s$% e2` operator is exported from 'stringi', and provides access to [stri_sprintf](#) in the form of an infix operator.

The `x %ss% p` operator splits the strings in `x` by a delimiter character/pattern defined in `p`, and removes `p` in the process.

For cutting strings by text boundaries, or around a location, see [strcut_brk](#) and [strcut_loc](#).

Usage

`x %s-% p`

`x %s/% p`

`x %s//% brk`

`x %ss% p`

Arguments

<code>x</code>	a string or character vector.
<code>p</code>	either a list with 'stringi' arguments (see s_pattern), or else a character vector with regular expressions. about search: regex about search: fixed about search: coll about search: charclass

brk a list with break iteration options, like a list produced by [stri_opts_brkiter](#).
 about search: boundaries

Value

The %s+%, %s-%, and %s*% operators return a character vector of the same length as x.

The %s/% and %s//% both return an integer vector of the same length as x.

The %s\$% operator returns a character vector.

The %ss% operator returns a list of the split strings - or, if simplify = TRUE / simplify = NA, returns a matrix of the split strings.

See Also

[tinycodet_strings](#)

Examples

```
x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)
y <- c("a", "b")
p <- rep("a|e|i|o|u", 2) # same as p <- list(regex = rep("a|e|i|o|u", 2))
n <- c(3, 2)
```

```
x %s+% y # = paste0(x,y)
x %s-% p # remove all vowels from x
x %s*% n
x %s/% p # count how often vowels appear in each string of vector x
x %ss% p # split x around vowels, removing the vowels in the process
x %ss% s_regex(p, simplify = NA) # same as above, but in matrix form
```

```
test <- c(
  paste0("The\u00a0above-mentioned features are very useful. ",
    "Spam, spam, eggs, bacon, and spam. 123 456 789"),
  "good morning, good evening, and good night"
)
test %s//% list(type = "character")
```

```
x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)
y <- "a"
# pattern that ignores case:
p <- list(regex = rep("A|E|I|O|U", 2), case_insensitive = TRUE)
n <- c(2, 3)

x %s+% y # = paste0(x,y)
x %s-% p # remove all vowels from x
x %s*% n
x %s/% p # count how often vowels appears in each string of vector x.
```

```
x <- c(paste(letters, collapse = ", "), paste(LETTERS, collapse = ", "))
print(x)
x %ss% " ", "
t(x %ss% s_fixed(" ", " ", simplify = NA))
```

str_search

'stringi' Pattern Search Operators

Description

The `x %s{ }% p` and `x %s!{ }% p` Operators:

The `x %s{ }% p` operator checks for every string in character vector `x` if the pattern defined in `p` is present.

When supplying a list on the right hand side (see [s_pattern](#)), one can optionally include the list element `at = "start"` or `at = "end"`:

- Supplying `at = "start"` will check if the pattern appears at the start of a string (like [stri_startswith](#)).
- Supplying `at = "end"` will check if the pattern appears at the end of a string (like [stri_endswith](#)).

The `x %s!{ }% p` operator is the same as `x %s{ }% p`, except it checks for **absence** of the pattern, rather than presence.

For string (in)equality operators, see [%s==%](#) from the 'stringi' package.

`strfind()` <-:

`strfind()` <- locates, extracts, or replaces found patterns.

It complements the other string-related operators, and uses the same [s_pattern](#) API.

It functions as follows:

- `strfind()` finds all pattern matches, and returns the extractions of the findings in a list, just like [stri_extract_all](#).
- `strfind(..., i = "all" )`, finds all pattern matches like [stri_locate_all](#).
- `strfind(..., i = i)`, where `i` is an integer vector, locates the i^{th} occurrence of a pattern, and reports the locations in a matrix, just like [stri_locate_ith](#).
- `strfind() <- value` finds pattern matches in variable `x`, replaces the pattern matches with the character vector specified in `value`, and assigns the transformed character vector back to `x`.

This is somewhat similar to [stri_replace](#), though the replacement is done in-place.

Usage

```
x %s{ }% p

x %s!{ }% p

strfind(x, p, ..., i, rt)

strfind(x, p, ..., i, rt) <- value
```

Arguments

- | | |
|-------|--|
| x | a string or character vector.
For <code>strfind()</code> <-, x must obviously be the variable containing the character vector/string, since <code>strfind()</code> <- performs assignment in-place. |
| p | either a list with 'stringi' arguments (see s_pattern), or else a character vector with regular expressions.
See also the Details section.
about search: regex
about search: fixed
about search: coll
about search: charclass |
| ... | additional arguments to be specified. |
| i | either one of the following can be given for i: <ul style="list-style-type: none"> • if i is not given or NULL, <code>strfind()</code> extracts all found pattern occurrences. • if i is the string "all", <code>strfind()</code> locates all found pattern occurrences. • if i is an integer, <code>strfind()</code> locates the <i>i</i>th pattern occurrences.
See the i argument in stri_locate_ith for details. For <code>strfind()</code> <- value, i must not be specified. |
| rt | use rt to specify the Replacement Type that <code>strfind()</code> <- should perform.
Either one of the following can be given for rt: <ul style="list-style-type: none"> • if rt is not given, NULL or "vec", <code>strfind()</code><- performs regular, vectorized replacement of all occurrences. • if rt = "dict", <code>strfind()</code><- performs dictionary replacement of all occurrences. • if rt = "first", <code>strfind()</code><- replaces only the first occurrences. • if rt = "last", <code>strfind()</code><- replaces only the last occurrences. Note: rt = "first" and rt = "last" only exist for convenience; for more specific locational replacement, use stri_locate_ith or <code>strfind(..., i)</code> with numeric i (see the Examples section).
For <code>strfind()</code> , rt must not be specified. |
| value | a character vector giving the replacement values. |

Details

Right-hand Side List for the %s{}% and %s!{}% Operators

When supplying a list to the right-hand side of the %s{}% and %s!{}% operators, one can add the argument `at`.

If `at = "start"`, the operators will check if the pattern is present/absent at the start of the string.

If `at = "end"`, the operators will check if the pattern is present/absent at the end of the string.

Unlike `stri_startswith` or `stri_endswith`, regex is supported by the %s{}% and %s!{}% operators.

See examples below.

Vectorized Replacement vs Dictionary Replacement

- Vectorized replacement:
`x`, `p`, and `value` are of the same length (or recycled to become the same length).
All occurrences of pattern `p[j]` in `x[j]` is replaced with `value[j]`, for every `j`.
- Dictionary replacement:
`p` and `value` are of the same length, and their length is independent of the length of `x`.
For every single string in `x`, all occurrences of pattern `p[1]` are replaced with `value[1]`,
all occurrences of pattern `p[2]` are replaced with `value[2]`, etc.

Notice that for single replacement, i.e. `rt = "first"` or `rt = "last"`, it makes no sense to distinguish between vectorized or dictionary replacement, since then only a single occurrence is being replaced per string.

See examples below.

Value

For the `x %s{}% p` and `x %s!{}% p` operators:

Return logical vectors.

For `strfind()`:

Returns a list with extractions of all found patterns.

For `strfind(..., i = "all")`:

Returns a list with all found pattern locations.

For `strfind(..., i = i)` with integer vector `i`:

Returns an integer matrix with two columns, giving the start and end positions of the i^{th} matches, two NAs if no matches are found, and also two NAs if `str` is NA.

For `strfind() <- value`:

Returns nothing, but performs in-place replacement (using R's default in-place semantics) of the found patterns in variable `x`.

Note

strfind()<- performs in-place replacement.
 Therefore, the character vector or string to perform replacement on, must already exist as a variable.
 So take for example the following code:

```
strfind("hello", p = "e") <- "a" # this obviously does not work

y <- "hello"
strfind(y, p = "e") <- "a" # this works fine
```

In the above code, the first strfind()<- call does not work, because the string needs to exist as a variable.

See Also

[tinycodet_strings](#)

Examples

```
# example of %s{}% and %s!{}% ====

x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)
x %s{}% "a"
x %s!{}% "a"
which(x %s{}% "a")
which(x %s!{}% "a")
x[x %s{}% "a"]
x[x %s!{}% "a"]
x[x %s{}% "a"] <- 1
x[x %s!{}% "a"] <- 1
print(x)

x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
x %s{}% "1"
x %s!{}% "1"
which(x %s{}% "1")
which(x %s!{}% "1")
x[x %s{}% "1"]
x[x %s!{}% "1"]
x[x %s{}% "1"] <- "a"
x[x %s!{}% "1"] <- "a"
print(x)

#####
```

```

# Example of %s{%} and %s!{%} with "at" argument ====

x <- c(paste0(letters, collapse = ""),
      paste0(rev(letters), collapse = ""), NA)
p <- s_fixed("abc", at = "start")
x %s{%} p
stringi::stri_startswith(x, fixed = "abc") # same as above

p <- s_fixed("xyz", at = "end")
x %s{%} p
stringi::stri_endswith(x, fixed = "xyz") # same as above

p <- s_fixed("cba", at = "end")
x %s{%} p
stringi::stri_endswith(x, fixed = "cba") # same as above

p <- s_fixed("zyx", at = "start")
x %s{%} p
stringi::stri_startswith(x, fixed = "zyx") # same as above

#####

# Example of transforming ith occurrence ====

# new character vector:
x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)

# report ith (second and second-last) vowel locations:
p <- s_regex( # vowels
  rep("A|E|I|O|U", 2),
  case_insensitive = TRUE
)
loc <- strfind(x, p, i = c(2, -2))
print(loc)

# extract ith vowels:
extr <- stringi::stri_sub(x, from = loc)
print(extr)

# replace ith vowels with numbers:
repl <- chartr("aeiou", "12345", extr) # transformation
stringi::stri_sub(x, loc) <- repl
print(x)

#####

```

```
# Example of strfind for regular vectorized replacement ====
```

```
x <- rep('The quick brown fox jumped over the lazy dog.', 3)
print(x)
p <- c('quick', 'brown', 'fox')
rp <- c('SLOW', 'BLACK', 'BEAR')
x %s{}% p
strfind(x, p)
strfind(x, p) <- rp
print(x)
```

```
#####
```

```
# Example of strfind for dictionary replacement ====
```

```
x <- rep('The quick brown fox jumped over the lazy dog.', 3)
print(x)
p <- c('quick', 'brown', 'fox')
rp <- c('SLOW', 'BLACK', 'BEAR')
# thus dictionary is:
# quick => SLOW; brown => BLACK; fox => BEAR
strfind(x, p, rt = "dict") <- rp
print(x)
```

```
#####
```

```
# Example of strfind for first and last replacement ====
```

```
x <- rep('The quick brown fox jumped over the lazy dog.', 3)
print(x)
p <- s_fixed("the", case_insensitive = TRUE)
rp <- "One"
strfind(x, p, rt = "first") <- rp
print(x)
```

```
x <- rep('The quick brown fox jumped over the lazy dog.', 3)
print(x)
p <- s_fixed("the", case_insensitive = TRUE)
rp <- "Some Other"
strfind(x, p, rt = "last") <- rp
print(x)
```

Description

String subsetting operators.

The `x %s><% ss` operator gets a certain number of the first and last characters of every string in character vector `x`.

`%sget%` is an alias for `%s><%`.

The `x %s<>% ss` operator trims a certain number of the first and last characters of every string in character vector `x`.

`%strim%` is an alias for `%s<>%`.

Usage

```
x %s><% ss
```

```
x %s<>% ss
```

```
x %sget% ss
```

```
x %strim% ss
```

Arguments

<code>x</code>	a character vector.
<code>ss</code>	a vector of length 2, or a matrix with 2 columns with <code>nrow(ss) == length(x)</code> . The object <code>ss</code> should consist entirely of non-negative and non-missing integers, or be coerce-able to such integers. (thus negative integers, and missing values are not allowed; decimal numbers will be converted to integers). The first element/column of <code>ss</code> gives the number of characters counting from the left side to be extracted/removed from <code>x</code> . The second element/column of <code>ss</code> gives the number of characters counting from the right side to be extracted/removed from <code>x</code> .

Details

These operators serve as a way to provide straight-forward string sub-setting.

Value

Both operators return a character vector of the same length as `x`.

The `x %s><% ss` operator gives a certain number of the first and last characters of each string in the input character vector `x`.

The `x %s<>% ss` operator removes a certain number of the first and last characters of each string in the input character vector `x`.

See Also

[tinycodet_strings](#)

Examples

```
x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)
ss <- c(2, 3)
x %s<% ss

x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)
ss <- c(1, 0)
x %s>% ss

x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)
ss <- c(2, 3)
x %s<>% ss

x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)
ss <- c(1, 0)
x %s<>% ss
```

Description

The `%s-%`, `%s/ %`, `%ss%` operators, as well as the string search operators ([str_search](#)), perform pattern matching for some purpose, where the pattern is given in the second argument (`p`).

When a character vector or string is given as the second argument (`p`), this is interpreted as case-sensitive regex patterns from 'stringi'.

Instead of giving a string or character vector of regex patterns, one can also supply a list to specify

exactly how the pattern should be interpreted. The list should use the exact same argument convention as 'stringi'.

For example:

- `list(regex = p, case_insensitive = FALSE, ...)`
- `list(fixed = p, ...)`
- `list(coll = p, ...)`
- `list(charclass = p, ...)`

All arguments in the list are simply passed to the appropriate functions in 'stringi'.

For example:

```
x %s/% p
```

counts how often regular expression specified in character vector `p` occurs in `x`, whereas the following,

```
x %s/% list(fixed = p, case_insensitive = TRUE)
```

will do the same, except it uses `fixed` (i.e. literal) expression, and it does not distinguish between upper case and lower case characters.

'tinycodet' adds some convenience functions based on the `stri_opts_` - functions in 'stringi':

- `s_regex(p, ...)` is equivalent to `list(regex = p, ...)`
- `s_fixed(p, ...)` is equivalent to `list(fixed = p, ...)`
- `s_coll(p, ...)` is equivalent to `list(coll = p, ...)`
- `s_chrcls(p, ...)` is equivalent to `list(charclass = p, ...)`

With the ellipsis (...) being passed to the appropriate 'stringi'-functions when it matches their arguments.

'stringi' infix operators start with "%s", though they all have an alias starting with "%stri". In analogy to that, the above functions start with "s_" rather than "stri_", as they are all meant for operators only.

Usage

```
s_regex(
  p,
  case_insensitive,
  comments,
  dotall,
  multiline,
  time_limit,
  stack_limit,
```

```

    ...
)

s_fixed(p, case_insensitive, overlap, ...)

s_coll(
    p,
    locale,
    strength,
    alternate_shifted,
    french,
    uppercase_first,
    case_level,
    numeric,
    normalization,
    ...
)

s_chrcls(p, ...)

```

Arguments

p a character vector giving the pattern to search for.
 about search: [regex](#)
 about search: [fixed](#)
 about search: [coll](#)
 about search: [charclass](#)

case_insensitive
 see [stri_opts_regex](#) and [stri_opts_fixed](#).

comments, dotall, multiline
 see [stri_opts_regex](#).

time_limit, stack_limit
 see [stri_opts_regex](#).

... additional arguments not part of the `stri_opts` - functions to be passed here.
 For example: the `at` argument for the [str_search](#) operators.

overlap see [stri_opts_fixed](#).

locale, strength, alternate_shifted
 see [stri_opts_collator](#).

french, normalization, numeric
 see [stri_opts_collator](#).

uppercase_first, case_level
 see [stri_opts_collator](#).

Value

A list with arguments to be passed to the appropriate operators.

See Also[tinycodet_strings](#)**Examples**

```

x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)
p <- rep("a|e|i|o|u", 2) # same as p <- list(regex = rep("a|e|i|o|u", 2))
x %s/% p # count how often vowels appear in each string of vector x.

x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)
x %s/% list(regex = rep("A|E|I|O|U", 2), case_insensitive = TRUE)
x %s/% s_regex(rep("A|E|I|O|U", 2), case_insensitive = TRUE)

x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""))
print(x)
p <- list(fixed = c("A", "A"), case_insensitive = TRUE)
x %s{%}% p
x %s!{%}% p
p <- s_fixed(c("A", "A"), case_insensitive = TRUE)
x %s{%}% p
x %s!{%}% p

x <- c(paste0(letters[1:13], collapse = ""),
      paste0(letters[14:26], collapse = ""), NA)
p <- s_fixed("abc", at = "start")
x %s{%}% p
stringi::stri_startswith(x, fixed = "abc") # same as above

p <- s_fixed("xyz", at = "end")
x %s{%}% p
stringi::stri_endswith(x, fixed = "xyz") # same as above

```

tinyimport_alias

Class tinyimport_alias

Description

The [import_as](#) function creates an object of class "tinyimport_alias". This help page documents its usage.

To get a function from a tinyimport_alias, once can use the \$ operator.

To use, for example, function "some_function()" from alias ".alias", use:
`.alias$some_function()`.

To "unimport" the package alias object, simply remove it from the environment it was placed in.

`is.tinyimport_alias()` checks if an object truly is a package alias as created by the [import_as](#) function.

The `attr.import()` function gets one or all special attribute(s) from an alias object returned by [import_as](#).

Usage

```
is.tinyimport_alias(alias)
```

```
attr.import(alias, which = NULL)
```

Arguments

<code>alias</code>	the alias object as created by the import_as function.
<code>which</code>	The attributes to list. If NULL, all attributes will be returned. Possibilities: "pkgs", "conflicts", and "ordered_object_names".

Value

For `is.tinyimport_alias()`:
TRUE or FALSE, indicating if the object is a tiny package alias.

For `attr.import(alias, which = NULL)`:
All special attributes of the given alias object are returned as a list.

For `attr.import(alias, which = "pkgs")`:
Returns a list with 3 elements:

- `packages_order`: a character vector of package names, giving the packages in the order they were imported in the alias object.
- `main_package`: a string giving the name of the main package. Re-exported functions, if present, are taken together with the main package.
- `re_exports.pkgs`: a character vector of package names, giving the packages from which the re-exported functions in the main package were taken.

For `attr.import(alias, which = "conflicts")`:
The order in which packages are imported in the alias object (see attribute `pkgs$packages_order`)

matters: Functions from later named packages overwrite those from earlier named packages, in case of conflicts.

The "conflicts" attribute returns a data.frame showing exactly which functions overwrite functions from earlier named packages, and as such "win" the conflicts.

For `attr.import(alias, which = "ordered_object_names")`:

Gives the names of the objects in the alias, in the order as they were imported.

For conflicting objects, the last imported ones are used for the ordering.

Note that if argument `re_exports` is `TRUE`, re-exported functions are imported when the main package is imported, thus changing this order slightly.

See Also

[tinycodet_import](#)

Examples

```
import_as(.stri ~ stringi)
.stri$stri_join("a", "b")

import_from("stringi", import_ls("stringi", "infix"))
"a" %s+% "b"

attr.import(.stri)
```

transform_if

transform_if: Conditional Sub-set Transformation of Atomic objects

Description

The `transform_if()` function transforms an object `x`, based on the logical result (`TRUE`, `FALSE`, `NA`) of condition function `cond(x)` or logical vector `cond`, such that:

- For every value where `cond(x)==TRUE` / `cond==TRUE`, function `yes(x)` is run or scalar `yes` is returned.
- For every value where `cond(x)==FALSE` / `cond==FALSE`, function `no(x)` is run or scalar `no` is returned.
- For every value where `cond(x)==NA` / `cond==NA`, function `other(x)` is run or scalar `other` is returned.

For a more `ifelse`-like function where `yes`, `no`, and `other` are vectors, see `kit::iif`.

Usage

```
transform_if(x, cond, yes = function(x) x, no = function(x) x, other = NA)
```

Arguments

<code>x</code>	a vector, matrix, or array.
<code>cond</code>	either an object of class <code>logical</code> with the same length as <code>x</code> , or a (possibly anonymous) function that returns an object of class <code>logical</code> with the same length as <code>x</code> . For example: <code>\(x)x>0</code> .
<code>yes</code>	the (possibly anonymous) transformation function to use when function <code>cond(x)==TRUE</code> / <code>logical cond==TRUE</code> . Alternatively, one can also supply an atomic scalar. If argument <code>yes</code> is not specified, it defaults to <code>\(x)x</code> .
<code>no</code>	the (possibly anonymous) transformation function to use when function <code>cond(x)==FALSE</code> / <code>logical cond==FALSE</code> . Alternatively, one can also supply an atomic scalar. If argument <code>no</code> is not specified, it defaults to <code>\(x)x</code> .
<code>other</code>	the (possibly anonymous) transformation function to use when function <code>cond(x)</code> / <code>logical cond</code> returns <code>NA</code> . Alternatively, one can also supply an atomic scalar. If argument <code>other</code> is not specified, it defaults to <code>NA</code> . Note that function <code>other(x)</code> is run or scalar <code>other</code> is returned when function <code>cond(x)</code> or <code>logical cond</code> is <code>NA</code> , not necessarily when <code>x</code> itself is <code>NA</code> .

Details

Be careful with coercion! For example the following code:

```
x <- c("a", "b")
transform_if(x, \(x) x == "a", as.numeric, as.logical)
```

returns:

```
[1] NA NA
```

due to the same character vector being given 2 incompatible classes.

Value

The transformed vector, matrix, or array (attributes are conserved).

See Also

[tinycodet_dry](#)

Examples

```
x <- c(-10:9, NA, NA)
object <- matrix(x, ncol = 2)
attr(object, "helloworld") <- "helloworld"
print(object)
y <- 0
z <- 1000

object |> transform_if(\(x) x > y, log, \(x) x^2, \(x) -z)
object |> transform_if(object > y, log, \(x) x^2, -z) # same as previous line
```

with_pro	<i>Standard Evaluated Versions of Some Common Expression-Evaluation Functions</i>
----------	---

Description

The `with_pro()` and `aes_pro()` functions are standard-evaluated versions of the expression-evaluation functions `with` and `ggplot2::aes`, respectively.

These alternative functions are more programmatically friendly:

They use proper standard evaluation, through the usage of one-sided formulas, instead of non-standard evaluation, tidy evaluation, or similar programmatically unfriendly evaluations.

Usage

```
with_pro(data, form)
```

```
aes_pro(...)
```

Arguments

<code>data</code>	a list or <code>data.frame</code> .
<code>form</code>	a one-sided formula giving the expression to evaluate in <code>with_pro</code> . If the formula has an environment, that environment is used to find any variables or objects not present in <code>data</code> .
<code>...</code>	arguments to be passed to <code>ggplot2::aes</code> , but given as one-sided formulas.

Details

The `aes_pro()` function is the standard evaluated alternative to `ggplot2::aes`.

Due to the way `aes_pro()` is programmed, it should still work when tidy evaluation changes in 'ggplot2'.

To support functions in combinations with references of the variables, the input used here are formula inputs, rather than string inputs.
See the Examples section below.

Value

For `with_pro()`: see [with](#).

For `aes_pro()`: see `ggplot2::aes`.

Non-Standard Evaluation

Non-Standard Evaluation (sometimes abbreviated as "NSE"), is somewhat controversial.
Consider the following example:

```
aplot <- "ggplot2"  
library(aplot)
```

What package will be attached? It will not be 'ggplot2', nor will an error occur. Instead, the package 'aplot' will be attached.

This is due to evaluating the expression 'aplot' as a quoted expression, instead of evaluating the contents (i.e. string or formula) of the variable. In other words: Non-Standard Evaluation.

Regular Standard Evaluation does not have the above problem.

Note

The `with_pro()` function, like the original [with](#) function, is made for primarily for convenience. When using modelling or graphics functions with an explicit data argument (and typically using [formulas](#)), it is typically preferred to use the data argument of that function, rather than to use either `with(data, ...)` or `with_pro(data, ...)`.

See Also

[tinycodet_safer](#)

Examples

```
requireNamespace("ggplot2")  
  
d <- import_data("ggplot2", "mpg")  
  
# mutate data:
```

```
myform <- ~ displ + cyl + cty + hwy
d$mysum <- with_pro(d, myform)
summary(d)

# plotting data:
x <- ~ cty
y <- ~ sqrt(hwy)
color <- ~ drv

ggplot2::ggplot(d, aes_pro(x, y, color = color)) +
  ggplot2::geom_point()
```

Index

*** join_mat**
 stri_join_mat, 35
::, 4, 13, 14, 19, 22
\$, 30
%<-c%(lock), 24
%col~%(matrix_ops), 26
%d!=%(decimal_truth), 10
%d<=%(decimal_truth), 10
%d<%(decimal_truth), 10
%d==%(decimal_truth), 10
%d>=%(decimal_truth), 10
%d>%(decimal_truth), 10
%installed in%(pkgs), 27
%row~%(matrix_ops), 26
%s-%(str_arithmetic), 43
%s//%(str_arithmetic), 43
%s/%(str_arithmetic), 43
%s<>%(str_subset_ops), 51
%s><%(str_subset_ops), 51
%sget%(str_subset_ops), 51
%ss%(str_arithmetic), 43
%strim%(str_subset_ops), 51
%<-c%, 4
%s-%, %s/%, %ss%, 52
%s==%, 10, 45

aaa0_tinycodet_help, 2
aaa1_tinycodet_safer, 3
aaa2_tinycodet_import, 4
aaa3_tinycodet_strings, 7
aaa4_tinycodet_dry, 9
aes, 59, 60
aes_pro, 3
aes_pro(with_pro), 59
attaching, 4, 5
attr.import(tinyimport_alias), 55

Concatenate a character matrix row- or
 column-wise, 8

decimal_truth, 10
detecting patterns, 8

exists, 6

formula, 60

help, 20
help.import, 5
help.import(import_helper), 19

iif, 57
import_as, 5, 13, 20, 23, 55, 56
import_data, 5, 15
import_diagnose, 5, 16
import_env, 14, 17, 19
import_from, 5, 18, 21, 23, 24
import_from(), 21
import_helper, 19
import_inops(import_legacy), 21
import_legacy, 21
import_LL(import_legacy), 21
import_ls, 5, 18, 21, 22
import_ls(), 19, 21
interactive, 30
is.tinyimport_alias(tinyimport_alias),
 55
is_wholenumber(decimal_truth), 10

library, 4, 5, 23, 24
loadedNamespaces, 16
loadNamespace, 14, 28
lock, 24
lock_TF, 4
lock_TF(lock), 24
lockBinding, 19, 25
Logic, 11

matrix_ops, 26

Operators, 9

- pkg, 5
- pkg_get_deps (pkgs), 27
- pkg_get_deps_minimal, 14
- pkg_get_deps_minimal (pkgs), 27
- pkgs, 27
- s_chrcls (s_pattern), 52
- s_coll (s_pattern), 52
- s_fixed (s_pattern), 52
- s_pattern, 8, 43, 45, 46, 52
- s_regex (s_pattern), 52
- Safer decimal (in)equality testing, 3
- safer_partialmatch, 4, 30
- search, 17, 31, 32
- searchenv, 5, 31
- searchenv_, 17
- searchenv_add (searchenv), 31
- searchenv_get (searchenv), 31
- searchenv_rm (searchenv), 31
- setv, 25
- str_arithmetic, 43
- str_search, 45, 52, 54
- str_subset_ops, 50
- strcut_-functions, 8
- strcut_brk, 43
- strcut_brk (strcut_loc), 32
- strcut_loc, 8, 32, 43
- strfind (str_search), 45
- strfind()<-, 8
- strfind<- (str_search), 45
- stri_c_mat (stri_join_mat), 35
- stri_endswith, 45, 47
- stri_extract_all, 45
- stri_join, 35
- stri_join_mat, 35
- stri_locate_all, 38, 39, 45
- stri_locate_all_boundaries, 38
- stri_locate_first, 38
- stri_locate_ith, 8, 33, 37, 45, 46
- stri_locate_ith_boundaries, 8
- stri_locate_ith_boundaries
 (stri_locate_ith), 37
- stri_locate_ith_charclass
 (stri_locate_ith), 37
- stri_locate_ith_coll (stri_locate_ith),
 37
- stri_locate_ith_fixed
 (stri_locate_ith), 37
- stri_locate_ith_regex
 (stri_locate_ith), 37
- stri_locate_last, 38
- stri_match, 39
- stri_opts_brkiter, 33, 38, 44
- stri_opts_collator, 38, 54
- stri_opts_fixed, 38, 54
- stri_opts_regex, 38, 54
- stri_paste_mat (stri_join_mat), 35
- stri_replace, 45
- stri_split, 33
- stri_split_boundaries, 33
- stri_sprintf, 43
- stri_startswith, 45, 47
- stri_sub, 38, 39
- string arithmetic, 8
- string sub-setting, 8
- string sub-setting operators, 8
- strsplit, 33
- tinycodet (aaa0_tinycodet_help), 2
- tinycodet-package
 (aaa0_tinycodet_help), 2
- tinycodet_dry, 3, 26, 58
- tinycodet_dry (aaa4_tinycodet_dry), 9
- tinycodet_help, 4, 6, 8, 9
- tinycodet_help (aaa0_tinycodet_help), 2
- tinycodet_import, 3, 14, 15, 17–21, 24, 29,
 32, 57
- tinycodet_import
 (aaa2_tinycodet_import), 4
- tinycodet_safer, 2, 11, 25, 30, 60
- tinycodet_safer (aaa1_tinycodet_safer),
 3
- tinycodet_strings, 3, 34, 35, 39, 44, 48, 52,
 55
- tinycodet_strings
 (aaa3_tinycodet_strings), 8
- tinyimport_alias, 14, 20, 55
- transform_if, 9, 57
- use without attach, 4, 5
- with, 59, 60
- with_pro, 3, 59