

Package: csutil (via r-universe)

September 26, 2026

Title Common Base-R Problems Relating to Lists

Version 2026.8.22

Description Utility functions that help with common base-R problems relating to lists. Lists in base-R are very flexible. This package provides functions to quickly and easily characterize types of lists. That is, to identify if all elements in a list are null, data.frames, lists, or fully named lists. Other functionality is provided for the handling of lists, such as the easy splitting of lists into equally sized groups, and the unnesting of data.frames within fully named lists.

Depends R (>= 3.3.0)

License MIT + file LICENSE

URL <https://niphr.github.io/csutil/>, <https://github.com/niphr/csutil>

BugReports <https://github.com/niphr/csutil/issues>

Encoding UTF-8

Imports data.table, ggplot2

Suggests testthat, knitr, rmarkdown, rstudioapi, glue

VignetteBuilder knitr

Config/Needs/website niphr/cstemplate

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Repository <https://niphr.r-universe.dev>

Date/Publication 2026-08-26 21:51:30 UTC

RemoteUrl <https://github.com/niphr/csutil>

RemoteRef HEAD

RemoteSha bc5479f4f796582f2025f42c44de5b7b5658ae0b

Contents

apply_fn_via_hash_table	2
easy_split	3
is_all_list_elements_null_or_df	4
is_all_list_elements_null_or_fully_named_list	5
is_all_list_elements_null_or_list	6
is_fully_named_list	6
unnest_dfs_within_list_of_fully_named_lists	7

Index	10
--------------	-----------

apply_fn_via_hash_table	<i>Apply a function via hash table</i>
-------------------------	--

Description

Extracts the unique values of `x`, applies `fn` to those unique values, and maps the results back to the original vector. This avoids redundant computation when `x` contains many repeated values.

Usage

```
apply_fn_via_hash_table(x, fn, ...)
```

Arguments

<code>x</code>	A vector whose values need a function applied.
<code>fn</code>	A vectorised function, called once on the vector of unique values of <code>x</code> .
<code>...</code>	Additional arguments passed to <code>fn</code> .

Details

The function calls `fn` exactly once, on the vector of unique values of `x`. It does not call `fn` once per element. It does not call `fn` once per unique value either. `fn` MUST therefore be vectorised. It receives the whole vector of unique values in a single call.

This can dramatically speed up computation when there is a lot of data and a limited number of unique values. The work `fn` does scales with the number of unique values, not with the length of `x`.

Value

A vector of the same length as `x`, holding the value of `fn` for each element (computed via unique-value lookup). The result takes its class from the value `fn` returns.

See Also

`vignette("csutil", package = "csutil")`, which times this function against a direct call to `format()`.

Examples

```
x <- c("a", "b", "a", "c", "b", "a")
apply_fn_via_hash_table(x, toupper)

# passing extra arguments to fn
nums <- c(1.1, 2.2, 1.1, 3.3)
apply_fn_via_hash_table(nums, round, digits = 0)
```

easy_split

*Split a vector into a list of vectors***Description**

Splits `x` into a list of consecutive groups. You **MUST** specify exactly one of `size_of_each_group` or `number_of_groups`. Neither argument, or both arguments, is an error.

Usage

```
easy_split(x, size_of_each_group = NULL, number_of_groups = NULL)
```

Arguments

`x` The vector to split.

`size_of_each_group` The number of elements in each group. The last group may hold fewer.

`number_of_groups` The number of groups you request. The function can return fewer groups than you request. See Details.

Details

Every group has the same length except the last, which may be shorter. The function preserves the original order of `x`. It names the list elements "1", "2" and so on.

`number_of_groups` is a request, not a guarantee. The function computes the group size as `ceiling(length(x) / number_of_groups)`. The number of groups it returns is then `ceiling(length(x) / ceiling(length(x) / number_of_groups))`. That count can be smaller than `number_of_groups`. For example, `easy_split(1:4, number_of_groups = 3)` returns two groups of two.

Value

A list of vectors, holding the elements of `x` in their original order. Every element has the same length except the last, which may be shorter.

See Also

`vignette("csutil", package = "csutil")`, which splits `letters[1:20]` both ways.

Examples

```
easy_split(letters[1:20], size_of_each_group = 3)
easy_split(letters[1:20], number_of_groups = 3)

# number_of_groups is a request: this asks for 3 groups and returns 2
easy_split(1:4, number_of_groups = 3)
```

```
is_all_list_elements_null_or_df
```

Are all elements in a list null or data.frames?

Description

Checks two things: that 'x' is a list, and that all elements in 'x' are 'NULL' or data.frames.

Usage

```
is_all_list_elements_null_or_df(x)
```

Arguments

x An object

Value

TRUE or FALSE.

See Also

vignette("csutil", package = "csutil"), which runs all four predicates on small example lists.

Other list predicates: [is_all_list_elements_null_or_fully_named_list\(\)](#), [is_all_list_elements_null_or_list\(\)](#), [is_fully_named_list\(\)](#)

Examples

```
is_all_list_elements_null_or_df(data.frame())
is_all_list_elements_null_or_df(list(data.frame()))
is_all_list_elements_null_or_df(list(1, NULL))
is_all_list_elements_null_or_df(list(data.frame(), NULL))
is_all_list_elements_null_or_df(list("a"=1, 2))
```

`is_all_list_elements_null_or_fully_named_list`*Are all elements in a list null or fully named lists?*

Description

Checks two things: that 'x' is a list, and that all elements in 'x' are 'NULL' or fully named lists.

Usage

```
is_all_list_elements_null_or_fully_named_list(x)
```

Arguments

x	An object
---	-----------

Details

A fully named list is a list in which every element has a name.

Value

TRUE or FALSE.

See Also

`vignette("csutil", package = "csutil")`, which runs all four predicates on small example lists.

Other list predicates: [is_all_list_elements_null_or_df\(\)](#), [is_all_list_elements_null_or_list\(\)](#), [is_fully_named_list\(\)](#)

Examples

```
is_all_list_elements_null_or_fully_named_list(data.frame())
is_all_list_elements_null_or_fully_named_list(list(data.frame()))
is_all_list_elements_null_or_fully_named_list(list(1, NULL))
is_all_list_elements_null_or_fully_named_list(list(list(), NULL))
is_all_list_elements_null_or_fully_named_list(list(list("a" = 1), NULL))
is_all_list_elements_null_or_fully_named_list(list("a"=1, 2))
```

```
is_all_list_elements_null_or_list
```

Are all elements in a list null or lists?

Description

Checks two things: that 'x' is a list, and that all elements in 'x' are 'NULL' or lists.

Usage

```
is_all_list_elements_null_or_list(x)
```

Arguments

x An object

Value

TRUE or FALSE.

See Also

`vignette("csutil", package = "csutil")`, which runs all four predicates on small example lists.

Other list predicates: `is_all_list_elements_null_or_df()`, `is_all_list_elements_null_or_fully_named_list()`, `is_fully_named_list()`

Examples

```
is_all_list_elements_null_or_list(data.frame())
is_all_list_elements_null_or_list(list(data.frame()))
is_all_list_elements_null_or_list(list(1, NULL))
is_all_list_elements_null_or_list(list(list(), NULL))
is_all_list_elements_null_or_list(list("a"=1, 2))
```

```
is_fully_named_list      Is this a fully named list?
```

Description

Checks if 'x' is a list with each element named.

Usage

```
is_fully_named_list(x)
```

Arguments

x An object

Details

The empty list is a special case. `is_fully_named_list(list())` returns FALSE, because a fully named list must have at least one element. The other three predicates in this family return TRUE for `list()`.

Value

TRUE or FALSE.

See Also

`vignette("csutil", package = "csutil")`, which runs all four predicates on small example lists.

Other list predicates: [is_all_list_elements_null_or_df\(\)](#), [is_all_list_elements_null_or_fully_named_list\(\)](#), [is_all_list_elements_null_or_list\(\)](#)

Examples

```
is_fully_named_list(list())
is_fully_named_list(list(1))
is_fully_named_list(list("a"=1))
is_fully_named_list(list("a"=1, 2))
```

```
unnest_dfs_within_list_of_fully_named_lists
```

Unnest data.frames within fully named list

Description

Consider a function that returns a list of two data.frames. Call that function repeatedly and store each return value in a list. You then hold a list of fully named lists, and each inner list holds a data.frame. Typically you want the two data.frames out of that nested structure, row-bound with `data.table::rbindlist()`.

Usage

```
unnest_dfs_within_list_of_fully_named_lists(
  x,
  returned_name_when_dfs_are_not_nested = "data",
  ...
)
```

Arguments

`x` A list of fully named lists (which then contain data.frames)
`returned_name_when_dfs_are_not_nested` The name to give the returned element when `x` is a single list of data.frames.
`...` Parameters passed to `data.table::rbindlist()`.

Details

When every element of `x` is `NULL` or a `data.frame`, the function row-binds those elements into a single `data.table`. It returns that table in a length-1 list, named by `returned_name_when_dfs_are_not_nested`.

Otherwise every element of `x` MUST be `NULL` or a fully named list. The function stops with an error if one element is not. The function names the returned list by the sorted union of the inner names. Each element is the `data.table::rbindlist()` of the inner elements that carry that name.

Value

A fully named list, each element a `data.table`. `NULL` if `x` is not a list.

See Also

`vignette("csutil", package = "csutil")`, which unnests a two-element list of fully named lists.

Examples

```
x <- list(
  list(
    "a" = data.frame("v1"=1),
    "b" = data.frame("v2"=3)
  ),
  list(
    "a" = data.frame("v1"=10),
    "b" = data.frame("v2"=30),
    "d" = data.frame("v3"=50)
  ),
  list(
    "a" = NULL
  ),
  NULL
)
print(x)
csutil::unnest_dfs_within_list_of_fully_named_lists(x)

x <- list(
  data.frame("v1"=1),
  data.frame("v3"=50)
)
print(x)
csutil::unnest_dfs_within_list_of_fully_named_lists(
  x,
```

```
    returned_name_when_dfs_are_not_nested = "NAME",  
    fill = TRUE  
  )
```

Index

* **list predicates**

- is_all_list_elements_null_or_df, [4](#)
- is_all_list_elements_null_or_fully_named_list,
[5](#)
- is_all_list_elements_null_or_list,
[6](#)
- is_fully_named_list, [6](#)

apply_fn_via_hash_table, [2](#)

easy_split, [3](#)

- is_all_list_elements_null_or_df, [4](#)
- is_all_list_elements_null_or_df(), [5–7](#)
- is_all_list_elements_null_or_fully_named_list,
[5](#)
- is_all_list_elements_null_or_fully_named_list(),
[4, 6, 7](#)
- is_all_list_elements_null_or_list, [6](#)
- is_all_list_elements_null_or_list(), [4](#),
[5, 7](#)
- is_fully_named_list, [6](#)
- is_fully_named_list(), [4–6](#)

unnest_dfs_within_list_of_fully_named_lists,
[7](#)