Package: rray (via r-universe)

November 18, 2026

Title Array Helpers

Version 0.2.0

Description Broadcast, reshape, reduce, and index base R arrays. Broadcasting repeats axes with a dimension of one so that arrays with different dimensions can be flexibly combined. Every function broadcasts its inputs, so operations such as arithmetic, comparison, and combining work consistently across axes.

License MIT + file LICENSE

URL https://github.com/r-lib/rray, https://rray.r-lib.org/

BugReports https://github.com/r-lib/rray/issues

Depends R (>= 4.3.0)

Imports rlang

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/build/compilation-database true

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Repository https://r-lib.r-universe.dev

Date/Publication 2026-11-17 12:34:01 UTC

RemoteUrl https://github.com/r-lib/rray

RemoteRef HEAD

RemoteSha 2e8f8ae6c5b96f06f3ee44d3334d266205247af3

Contents

faq-axes . . . 3
faq-common-names . . . 4
rray-arithmetic . . . 7
rray-compare . . . 8
rray-cumulative . . . 9
rray-equal . . . 12
rray-extremum . . . 13
rray-locate . . . 14
rray-logical . . . 15
rray-reduce . . . 16
rray-rep . . . 18
rray-roll . . . 20
rray-titles . . . 25
rray_axis_names . . . 26
rray_broadcast . . . 27
rray_broadcast_common . . . 27
rray_combine . . . 28
rray_dimensionality . . . 29
rray_dimensions . . . 29
rray_dimensions_common . . . 30
rray_extract . . . 31
rray_if_else . . . 32
rray_index . . . 33
rray_insert_axes . . . 34
rray_move_axes . . . 36
rray_names . . . 37
rray_permute_axes . . . 38
rray_remove_axes . . . 39
rray_reverse . . . 40
rray_set_axis_names . . . 41
rray_set_dimensions . . . 42
rray_set_names . . . 42
rray_size . . . 43
rray_slice . . . 44
rray_split . . . 46
rray_stack . . . 47
rray_unstack . . . 48

Index **50**

faq-axes *FAQ - How do I refer to an axis?*

Description

Functions that work along an axis typically allow you to provide it by position or by title.

Positions

A position counts axes from 1. Axis 1 runs down the rows, axis 2 runs across the columns, and so on.

```
x <- array(
  1:24,
  dim = c(2, 3, 4),
  dimnames = list(row = NULL, col = NULL, page = NULL)
)

rray_dimensions(rray_sum(x, axes = 1))
#> [1] 1 3 4
```

Titles

Titles are the names of the dimension names list, like `row` in `list(row = NULL)`. `rray_titles()` returns them. A title works most anywhere a position does:

```
rray_titles(x)
#> [1] "row"  "col"  "page"

identical(rray_sum(x, axes = "row"), rray_sum(x, axes = 1))
#> [1] TRUE

identical(rray_sum(x, axes = c("page", "row")), rray_sum(x, axes = c(1, 3)))
#> [1] TRUE
```

Positions in the output

Some arguments specify a position in the output rather than an axis of `x`, like `axes` in `rray_insert_axes()` and `.axis` in `rray_stack()`. That axis doesn't exist yet, so it has no title, and these arguments only take positions:

```
rray_dimensions(rray_insert_axes(x, axes = 1))
#> [1] 1 2 3 4

rray_insert_axes(x, axes = "row")
#> Error in `rray_insert_axes()`:
#> ! `axes` must be an integer or double vector, not the string "row".
```

faq-common-names *FAQ - How are common names and titles determined?*

Description

When a function builds one result from several inputs, the inputs must agree on their names and titles. If they don't, the function errors instead of silently picking one input's names or titles.

These functions follow the rules:

- `rray_add()`, `rray_subtract()`, `rray_multiply()`, `rray_divide()`, and `rray_exponentiate()`
- `rray_and()`, `rray_or()`, and `rray_xor()`
- `rray_pmax()` and `rray_pmin()`
- `rray_combine()` and `rray_stack()`

Rules

For each axis of the result:

All inputs with names along that axis must have exactly the same names. If all names are the same, the result has those names. If no input is named, the result is not named. If any names are different, an error is thrown to avoid arbitrarily picking between the inputs. If an input is broadcast or has no names, it is excluded from common name determination.

To resolve a conflict, preemptively remove the names from one input with `rray_set_names()`.

Examples

Matching names are kept, and different names are an error:

```
x <- array(1:2, 2, dimnames = list(c("r1", "r2")))
y <- array(1:2, 2, dimnames = list(c("a", "b")))

rray_add(x, x)
#> r1 r2
#>  2  4

rray_add(x, y)
#> Error in `rray_add()`:
#> ! Can't find common names at axis 1. `x` and `y` have different names at position 1: "r1" and "a".

rray_add(x, rray_set_names(y, NULL))
#> r1 r2
#>  2  4
```

An input without names on an axis doesn't participate there. Here, each input names a different axis, so both sets of names are kept:

```
x <- array(1:4, c(2, 2), dimnames = list(c("r1", "r2"), NULL))
y <- array(5:8, c(2, 2), dimnames = list(NULL, c("c1", "c2")))

rray_add(x, y)
#>    c1 c2
#> r1  6 10
#> r2  8 12
```

A broadcast axis doesn't participate either. `"total"` names a single row that gets repeated, so it no longer describes the result and is dropped:

```
x <- array(1:3, c(1, 3), dimnames = list("total", c("a", "b", "c")))
y <- array(1:6, c(2, 3), dimnames = list(c("r1", "r2"), c("a", "b", "c")))

rray_multiply(x, y)
#>    a b  c
#> r1 1 6 15
#> r2 2 8 18
```

Empty names count, so `c("r1", "")` and `c("r1", "r2")` are different:

```
x <- array(1:2, 2, dimnames = list(c("r1", "")))
y <- array(1:2, 2, dimnames = list(c("r1", "r2")))

rray_add(x, y)
#> Error in `rray_add()`:
#> ! Can't find common names at axis 1. `x` and `y` have different names at position 2: "" and "r2".
```

Titles

Titles are the names of the dimension names list, like `Row` in `list(Row = c("r1", "r2"))`. They label an axis rather than its positions.

For each axis of the result, all inputs with a title on that axis must have the same title, and the result gets that title. An empty title `""` counts as no title. If any titles are different, an error is thrown.

```
x <- array(1:2, 2, dimnames = list(Row = c("r1", "r2")))
y <- array(1:2, 2, dimnames = list(Sample = c("r1", "r2")))
z <- array(1:2, 2, dimnames = list(c("r1", "r2")))

rray_add(x, y)
#> Error in `rray_add()`:
#> ! Can't find common titles at axis 1. `x` has title "Row" and `y` has title "Sample".

rray_titles(rray_add(x, z))
#> [1] "Row"
```

Unlike names, a title survives broadcasting, so a broadcast input still has to agree. Here the names on axis 1 are fine because `x` is broadcast there, but the titles are not:

```
x <- array(1:3, c(1, 3), dimnames = list(Row = "total", Col = c("a", "b", "c")))
y <- array(1:6, c(2, 3), dimnames = list(Sample = c("r1", "r2"), Col = c("a", "b", "c")))

rray_multiply(x, y)
#> Error in `rray_multiply()`:
#> ! Can't find common titles at axis 1. `x` has title "Row" and `y` has title "Sample".
```

To resolve a conflict, remove the titles from one input with `rray_set_titles()`.

Combining and stacking

`rray_combine()` never checks names on `.axis`. They are combined from every input, with `""` for inputs that have none. Titles on `.axis` are checked like any other axis. Every other axis follows the rules above:

```
a <- array(1:2, c(1, 2), dimnames = list("r1", c("x", "y")))
b <- array(3:4, c(1, 2))
c <- array(5:6, c(1, 2), dimnames = list("r3", c("x", "z")))

rray_combine(a, b, c, .axis = 1)
#> Error in `rray_combine()`:
#> ! Can't find common names at axis 2. `..1` and `..3` have different names at position 2: "y" and "z".

c <- array(5:6, c(1, 2), dimnames = list("r3", c("x", "y")))

rray_combine(a, b, c, .axis = 1)
#>    x y
#> r1 1 2
#>    3 4
#> r3 5 6
```

`rray_stack()` names the new axis with the names of `...`, and gives it no title. Every other axis follows the rules above, which catches data that was reordered:

```
jan <- array(1:2, 2, dimnames = list(c("north", "south")))
feb <- array(3:4, 2, dimnames = list(c("south", "north")))

rray_stack(jan = jan, feb = feb, .axis = 2)
#> Error in `rray_stack()`:
#> ! Can't find common names at axis 1. `jan` and `feb` have different names at position 1: "north" and "s
```

Comparisons

Comparison functions like `rray_equal()` and `rray_greater_than()` return a logical relation about their inputs rather than a new value of the input type, so they don't return names or titles on the result and aren't subject to these checks:

```
x <- array(c(TRUE, FALSE), 2, dimnames = list(c("r1", "r2")))
```

```
y <- array(c(TRUE, TRUE), 2, dimnames = list(c("a", "b")))

rray_equal(x, y)
#> [1]  TRUE FALSE
```

`rray-arithmetic` *Array arithmetic*

Description

- `rray_add()` adds two arrays elementwise.
- `rray_subtract()` subtracts two arrays elementwise.
- `rray_multiply()` multiplies two arrays elementwise.
- `rray_divide()` divides two arrays elementwise.
- `rray_exponentiate()` raises the elements of one array to the power of the elements of another.

Usage

```
rray_add(x, y)

rray_subtract(x, y)

rray_multiply(x, y)

rray_divide(x, y)

rray_exponentiate(x, y)
```

Arguments

x	An array.
y	An array.

Details

The arrays are broadcast to common dimensions first, so they do not have to be the same shape.

If the result of an integer operation would overflow, an error is thrown.

These functions participate in rray's common name determination rules.

Value

An array with the common dimensions of `x` and `y`.

Casting

Certain inputs are upcast, changing the return type:

- `rray_add()`: logicals are cast to integer.
- `rray_subtract()`: logicals are cast to integer.
- `rray_multiply()`: logicals are cast to integer.
- `rray_divide()`: logicals and integers are cast to double.
- `rray_exponentiate()`: logicals and integers are cast to double.

Examples

```
x <- array(1:6, c(3L, 2L))

rray_add(x, 1L)
rray_subtract(x, 1L)
rray_multiply(x, 2L)
rray_divide(x, 2L)
rray_exponentiate(x, 2L)

# Adding 10 to column 1 and 20 to column 2 via broadcasting
rray_add(x, array(c(10L, 20L), c(1L, 2L)))

# Scaling column 1 by 2 and column 2 by 3 the same way
rray_multiply(x, array(c(2L, 3L), c(1L, 2L)))

# Names are collected from both inputs, one axis at a time
rows <- array(1:3, c(3L, 1L), dimnames = list(c("r1", "r2", "r3")))
cols <- array(c(10L, 20L), c(1L, 2L), dimnames = list(NULL, c("c1", "c2")))
rray_add(rows, cols)

# A broadcast axis loses its names, because they no longer describe the
# result. `only` names a single row, but the result has three.
cols <- array(c(10L, 20L), c(1L, 2L), dimnames = list("only", c("c1", "c2")))
rray_add(x, cols)
```

---

rray-compare *Compare arrays*

Description

- `rray_greater_than()` tests whether `x` is greater than `y`.
- `rray_greater_than_or_equal()` tests whether `x` is greater than or equal to `y`.
- `rray_less_than()` tests whether `x` is less than `y`.
- `rray_less_than_or_equal()` tests whether `x` is less than or equal to `y`.

Usage

```
rray_greater_than(x, y)

rray_greater_than_or_equal(x, y)

rray_less_than(x, y)

rray_less_than_or_equal(x, y)
```

Arguments

x	An array.
y	An array.

Details

The arrays are broadcast to common dimensions before they are compared.

Value

A logical array with the common dimensions of `x` and `y`, and no names.

Examples

```
x <- array(1:6, c(3L, 2L))

rray_greater_than(x, 3L)
rray_less_than_or_equal(x, array(c(2L, 5L), c(1L, 2L)))
```

---

rray-cumulative *Accumulate along an axis*

Description

- `rray_cumulative_all()` checks if all values are `TRUE` so far along `axis`.
- `rray_cumulative_any()` checks if any value is `TRUE` so far along `axis`.
- `rray_cumulative_max()` computes the maximum so far along `axis`.
- `rray_cumulative_min()` computes the minimum so far along `axis`.
- `rray_cumulative_prod()` computes the product so far along `axis`.
- `rray_cumulative_sum()` computes the sum so far along `axis`.

Usage

```
rray_cumulative_all(x, ..., axis)

rray_cumulative_any(x, ..., axis)

rray_cumulative_max(x, ..., axis)

rray_cumulative_min(x, ..., axis)

rray_cumulative_prod(x, ..., axis)

rray_cumulative_sum(x, ..., axis)
```

Arguments

x	An array.
...	These dots are for future extensions and must be empty.
axis	The axis to accumulate along, given by position or by title.

Details

Each position along `axis` summarizes every position up to and including it.

Value

An array with the same dimensions and names as `x`.

All / Any

`x` must be logical. Missing values follow `all()` and `any()`, so a later `FALSE` or `TRUE` can settle an earlier `NA`:

```
rray_cumulative_all(c(TRUE, NA, FALSE), axis = 1L) # TRUE, NA, FALSE
rray_cumulative_any(c(FALSE, NA, TRUE), axis = 1L) # FALSE, NA, TRUE
```

Min / Max

`x` must be logical, integer, or double, and the result keeps the type of `x`. Once a line hits `NA` or `NaN`, the rest of it stays missing. A later `NA` replaces an earlier `NaN`, like `max()` and `min()`:

```
rray_cumulative_max(c(1, NaN, 2, NA, 3), axis = 1L) # 1, NaN, NaN, NA, NA
```

For logical `x`, a later `TRUE` or `FALSE` does not settle an earlier `NA`, unlike `rray_cumulative_any()` and `rray_cumulative_all()`:

```
rray_cumulative_max(c(NA, TRUE), axis = 1L) # NA, NA
rray_cumulative_any(c(NA, TRUE), axis = 1L) # NA, TRUE
```

Product

x must be logical, integer, double, or complex. Logicals and integers are cast to double.

When NA and NaN are both present, you get one of them, but which one depends on the platform and the order of the values. Either way, `is.na()` is TRUE from that point on:

```
rray_cumulative_prod(c(2, NA, NaN), axis = 1L) # 2, NA, NA or NaN
```

Sum

x must be logical, integer, double, or complex. Logicals are cast to integer.

An integer sum errors as soon as a running total overflows, even if a later value would bring it back in range. This differs from `rray_sum()`, which only checks the final total:

```
x <- c(.Machine$integer.max, 1L, -1L)
rray_sum(x, axes = 1L) # .Machine$integer.max
rray_cumulative_sum(x, axis = 1L) # Error
```

Once a line is NA, it stays NA, and a later overflow does not error.

When NA and NaN are both present, you get one of them, but which one depends on the platform and the order of the values. Either way, `is.na()` is TRUE from that point on:

```
rray_cumulative_sum(c(2, NA, NaN), axis = 1L) # 2, NA, NA or NaN
```

Examples

```
x <- array(c(TRUE, TRUE, FALSE, TRUE, NA, TRUE), c(2L, 3L))
x

# Down rows
rray_cumulative_all(x, axis = 1L)
rray_cumulative_any(x, axis = 1L)

# Across columns
rray_cumulative_all(x, axis = 2L)
rray_cumulative_any(x, axis = 2L)

x <- array(c(3L, 1L, 2L, 5L, 4L, 0L), c(2L, 3L))
x

# Running maximum and minimum across columns
rray_cumulative_max(x, axis = 2L)
rray_cumulative_min(x, axis = 2L)

# Running product down rows
rray_cumulative_prod(x, axis = 1L)

# Running sum across columns
rray_cumulative_sum(x, axis = 2L)
```

```
# Running sum across columns, by title
y <- rray_set_titles(x, c("row", "col"))
rray_cumulative_sum(y, axis = "col")
```

`rray-equal` *Equality*

Description

- `rray_equal()` tests whether `x` is equal to `y`.
- `rray_not_equal()` tests whether `x` is not equal to `y`.

Usage

```
rray_equal(x, y)

rray_not_equal(x, y)
```

Arguments

`x`	An array.
`y`	An array.

Details

The arrays are broadcast to common dimensions before they are compared.

Value

A logical array with the common dimensions of `x` and `y`, and no names.

Examples

```
x <- array(1:2, c(2L, 1L))

rray_equal(x, 1L)
rray_not_equal(x, array(c(1L, 3L), c(1L, 2L)))
```

rray-extremum *Elementwise maximum and minimum*

Description

- `rray_pmax()` computes the elementwise maximum of two arrays.
- `rray_pmin()` computes the elementwise minimum of two arrays.

Usage

```
rray_pmax(x, y, ..., na_rm = FALSE)

rray_pmin(x, y, ..., na_rm = FALSE)
```

Arguments

`x`	An array.
`y`	An array.
`...`	These dots are for future extensions and must be empty.
`na_rm`	If `TRUE`, missing values are removed before taking the maximum or minimum.

Details

The arrays are broadcast to common dimensions first, so they do not have to be the same shape.

When `NA` and `NaN` are both present, `NA` wins, like `rray_max()`, `rray_min()`, `max()`, and `min()`. This differs from `pmax()` and `pmin()`, which return whichever comes last:

```
rray_pmax(NA, NaN) # NA
rray_pmax(NaN, NA) # NA

pmax(NA, NaN) # NaN
```

These functions participate in rray's common name determination rules.

Value

An array with the common dimensions and common type of `x` and `y`.

Examples

```
x <- array(1:6, c(3L, 2L))

rray_pmax(x, 4L)
rray_pmin(x, 4L)

# Compare each column with a different value through broadcasting
rray_pmax(x, array(c(2L, 5L), c(1L, 2L)))
```

rray-locate *Locate the maximum or minimum along an axis*

Description

- `rray_locate_max()` finds the position of the maximum along `axis`.
- `rray_locate_min()` finds the position of the minimum along `axis`.

Usage

```
rray_locate_max(x, ..., axis, na_rm = FALSE)

rray_locate_min(x, ..., axis, na_rm = FALSE)
```

Arguments

`x`	An array.
`...`	These dots are for future extensions and must be empty.
`axis`	The axis to locate along, given by position or by title.
`na_rm`	If `TRUE`, missing values are skipped.

Details

The dimensionality of `x` is retained, with `axis` collapsed to a dimension of 1.

Ties return the first position, like `which.max()` and `which.min()`:

```
rray_locate_max(c(1, 3, 3), axis = 1L) # 2
```

Missing values are infectious, so the result is `NA` if any value along `axis` is `NA` or `NaN`. Use `na_rm = TRUE` to skip them:

```
rray_locate_max(c(1, NaN, 3), axis = 1L) # NA
rray_locate_max(c(1, NaN, 3), axis = 1L, na_rm = TRUE) # 3
```

When there is no position to return, the result is `NA`. This happens when `axis` has dimension 0, or when every value is missing and `na_rm = TRUE`:

```
rray_locate_max(c(NA, NA), axis = 1L, na_rm = TRUE) # NA
```

Value

An integer array with the same dimensionality as `x`, but with the dimension along `axis` reduced to 1.

Examples

```
x <- array(c(3L, 1L, 2L, 4L, 6L, 5L), c(3L, 2L))

# Position of the maximum going down the rows
rray_locate_max(x, axis = 1L)

# Position of the minimum going across the columns
rray_locate_min(x, axis = 2L)

# Position of the minimum going across the columns, by title
y <- rray_set_titles(x, c("row", "col"))
rray_locate_min(y, axis = "col")
```

---

rray-logical *Logical operations*

Description

- `rray_and()` tests whether both `x` and `y` are `TRUE`.
- `rray_or()` tests whether at least one of `x` and `y` is `TRUE`.
- `rray_xor()` tests whether exactly one of `x` and `y` is `TRUE`.

Usage

```
rray_and(x, y)

rray_or(x, y)

rray_xor(x, y)
```

Arguments

`x`, `y`	Logical arrays.

Details

The arrays are broadcast to common dimensions before the operation is applied.

Missing values are handled the same way as `&`, `|`, and `xor()`.

These functions participate in rray's common name determination rules.

Value

A logical array with the common dimensions of `x` and `y`.

Examples

```
x <- array(c(TRUE, FALSE, TRUE), c(3L, 1L))
y <- array(c(TRUE, FALSE), c(1L, 2L))

rray_and(x, y)
rray_or(x, y)
rray_xor(x, y)

rray_and(NA, FALSE)
rray_or(NA, TRUE)
rray_xor(NA, TRUE)
```

---

rray-reduce *Reduce an array along axes*

Description

- `rray_sum()` computes the sum along the specified `axes`.
- `rray_prod()` computes the product along the specified `axes`.
- `rray_mean()` computes the mean along the specified `axes`.
- `rray_all()` checks if all values are `TRUE` along the specified `axes`.
- `rray_any()` checks if any value is `TRUE` along the specified `axes`.
- `rray_max()` computes the maximum along the specified `axes`.
- `rray_min()` computes the minimum along the specified `axes`.

Usage

```
rray_sum(x, ..., axes, na_rm = FALSE)

rray_prod(x, ..., axes, na_rm = FALSE)

rray_mean(x, ..., axes, na_rm = FALSE)

rray_all(x, ..., axes, na_rm = FALSE)

rray_any(x, ..., axes, na_rm = FALSE)

rray_max(x, ..., axes, na_rm = FALSE)

rray_min(x, ..., axes, na_rm = FALSE)
```

Arguments

`x`	An array.
`...`	These dots are for future extensions and must be empty.
`axes`	The axes to reduce along, given by position or by title.
`na_rm`	If `TRUE`, missing values are removed before reducing.

Details

The dimensionality of `x` is retained in the result, with the reduced axes collapsed to size 1.

Value

An array with the same dimensionality as `x`, but with the dimensions along `axes` reduced to 1.

Sum

Logicals are summed as integers. If an integer sum doesn't fit in an integer, an error is thrown. Only the final sum is checked, and `NA` wins over an overflow:

```
x <- c(.Machine$integer.max, 1L, -1L)
rray_sum(x, axes = 1L) # .Machine$integer.max

x <- c(.Machine$integer.max, 1L, NA)
rray_sum(x, axes = 1L) # NA
```

Product

Logicals and integers are cast to double.

When `NA` and `NaN` are both present, you get one of them, but which one depends on the platform and the order of the values. Either way, `is.na()` is `TRUE`. Integers can hit this too, since a product that overflows to `Inf` becomes `NaN` when multiplied by `0`:

```
rray_prod(c(NA, NaN), axes = 1L) # NA or NaN

x <- c(rep(.Machine$integer.max, 40L), 0L, NA)
rray_prod(x, axes = 1L) # NA or NaN
```

Mean

Logicals and integers are cast to double.

When `NA` and `NaN` are both present, you get one of them, but which one depends on the platform and the order of the values, like `mean()`. Either way, `is.na()` is `TRUE`:

```
rray_mean(c(NA, NaN), axes = 1L) # NA or NaN
rray_mean(c(NaN, NA), axes = 1L) # NA or NaN
```

Min / Max

`rray_max()` and `rray_min()` keep the type of `x`. With nothing to reduce, such as an axis of dimension 0, `rray_max()` returns the smallest value of that type and `rray_min()` returns the largest:

type	`rray_max()`	`rray_min()`
logical	`FALSE`	`TRUE`
integer	`-.Machine$integer.max`	`.Machine$integer.max`
double	`-Inf`	`Inf`

When NA and NaN are both present, NA wins, like max() and min():

```
rray_max(c(NA, NaN), axes = 1L) # NA
rray_max(c(NaN, NA), axes = 1L) # NA
```

Examples

```
x <- array(1:10, c(5L, 2L))

# Sum along rows
rray_sum(x, axes = 1L)

# Sum along columns
rray_sum(x, axes = 2L)

# Sum along both axes
rray_sum(x, axes = c(1L, 2L))

# Sum along rows, by title
y <- rray_set_titles(x, c("row", "col"))
rray_sum(y, axes = "row")

# Product along rows
rray_prod(x, axes = 1L)

# Mean along rows
rray_mean(x, axes = 1L)

y <- array(c(TRUE, TRUE, FALSE, TRUE), c(2L, 2L))

rray_all(y, axes = 1L)
rray_any(y, axes = 1L)

# Maximum and minimum along columns
rray_max(x, axes = 2L)
rray_min(x, axes = 2L)
```

---

rray-rep *Repeat an array*

Description

- rray_rep() repeats whole axes of x in bulk, i.e. repeat all of the columns 5 times. It can repeat several axes at once, with one times per axis.
- rray_rep_each() repeats each individual element of the axis of x separately, i.e. repeat the first column 2 times, the second column 3 times, and so on. For convenience, you can also provide a single number to repeat each element along the axis the same number of times.

These are the array versions of vctrs::vec_rep() and vctrs::vec_rep_each(), which repeat along the size of a vector rather than along an axis.

Usage

```
rray_rep(x, ..., times, axes)

rray_rep_each(x, ..., times, axis)
```

Arguments

x	An array.
...	These dots are for future extensions and must be empty.
times	For `rray_rep()`, a vector of integers greater than or equal to 0, either size 1 or the size of `axes`. For `rray_rep_each()`, a vector of integers greater than or equal to 0. It is recycled to the dimension of `x` along `axis`.
axes	The axes to repeat along, given by position or by title.
axis	The axis to repeat along, given by position or by title.

Details

Names repeat alongside the data, so a repeated axis can come back with duplicate names. Every other axis keeps its names untouched. The titles of `x` are kept.

Value

An array with the same dimensions as `x`, except along the repeated axes.

Examples

```
x <- array(1:6, c(3, 2))

# All rows in bulk, twice
rray_rep(x, times = 2, axes = 1)

# Each row individually, twice
rray_rep_each(x, times = 2, axis = 1)

# Columns rather than rows
rray_rep(x, times = 2, axes = 2)

# Rows and columns, twice each
rray_rep(x, times = 2, axes = c(1, 2))

# Rows twice and columns three times
rray_rep(x, times = c(2, 3), axes = c(1, 2))

# A different number of repeats per row
rray_rep_each(x, times = c(1, 2, 3), axis = 1)

# A different number of repeats per row, by title
z <- rray_set_titles(x, c("row", "col"))
```

```
rray_rep_each(z, times = c(1, 2, 3), axis = "row")

# Zeroes drop slices entirely
rray_rep_each(x, times = c(1, 0, 2), axis = 1)

# Names repeat with the data, duplicates and all
y <- array(1:6, c(3, 2), dimnames = list(c("a", "b", "c"), c("x", "y")))
rray_names(rray_rep(y, times = 2, axes = 1))
rray_names(rray_rep_each(y, times = 2, axis = 1))
```

`rray-roll` *Roll an array*

Description

- `rray_roll()` rolls the elements of `x` along one or more `axes`, with one roll per axis. Everything along an axis moves by the same amount.
- `rray_roll_each()` rolls the elements of `x` along a single `axis`, with a separate roll for each row, column, and so on. For example, with a matrix and `axis = 2`, each row can move by a different amount.

An element pushed off one end circles back on the other.

Usage

```
rray_roll(x, ..., n, axes)

rray_roll_each(x, ..., n, axis)
```

Arguments

`x`	An array.
`...`	These dots are for future extensions and must be empty.
`n`	For `rray_roll()`, an integer vector indicating the amount to roll, either size 1 or the size of `axes`. For `rray_roll_each()`, an integer array indicating the amount to roll for each row, column, etc., as determined by `axis`. It is broadcast to the dimensions of `x` with `axis` set to 1.
`axes`	The axes to roll along, given by position or by title.
`axis`	The axis to roll along, given by position or by title.

Details

A positive `n` moves elements toward the end of the axis, and a negative `n` moves them toward the start. Rolls circle around, so an `n` of 7 along an axis with dimension 5 is the same as an `n` of 2. An `n` of 0, or any multiple of the dimension, leaves the axis unchanged.

Value

An array with the same type and dimensions as `x`.

Roll

Rolling an axis with a uniform `n` can be thought of as a carefully crafted `rray_slice()`. Along an axis with dimension `d`, position `i` of the result comes from position `(i - 1 - n) %% d + 1` of `x`:

```
x <- c(a = 1L, b = 2L, c = 3L, d = 4L, e = 5L)

n <- 2L
d <- length(x)
i <- seq_len(d)

(i - 1L - n) %% d + 1L
#> [1] 4 5 1 2 3

rray_roll(x, n = n, axes = 1)
#> d e a b c
#> 4 5 1 2 3

rray_slice_axis(x, c(4, 5, 1, 2, 3), axis = 1)
#> d e a b c
#> 4 5 1 2 3
```

Rolling several axes at once is one `rray_slice()`, with these locations on each rolled axis and `TRUE` on the rest.

Since a uniform roll is a slice, names on a rolled axis move with the data, just like they do in `rray_slice()`. Every other axis keeps its names untouched. The titles of `x` are kept.

Roll each

`rray_roll_each()` rolls along a single `axis`, but each row, column, and so on can move by its own amount. That is why `n` is an array rather than a vector. It is broadcast to the dimensions of `x` with `axis` set to 1, so it holds one `n` for every row, column, and so on that gets rolled.

Variable rolling can be mimicked with the more flexible indexing of `rray_index()`. Along `axis`, the coordinates come from the same `(i - 1 - n) %% d + 1` formula as `rray_roll()`, with a different `n` for each row, column, and so on. Along every other axis, the coordinates are just the positions themselves.

`rray_roll_each()` drops names along `axis`, but keeps all other names and all titles. When rows move by different amounts, the output's column 1 can hold data from column `c` in one row and from column `b` in another, so the original column names are unlikely to make sense:

```
x <- matrix(1:6, nrow = 2, dimnames = list(c("r1", "r2"), c("a", "b", "c")))

rray_roll_each(x, n = matrix(c(1, 2), ncol = 1), axis = 2)
#>    [,1] [,2] [,3]
```

```
#> r1    5    1    3
#> r2    4    6    2
```

Because variable rolling can be a bit complicated the first time you see it, we'll work through 2D and 3D examples below.

Two dimensions:

Start with a matrix:

```
x <- matrix(1:12, nrow = 3, byrow = TRUE)

x
#>      [,1] [,2] [,3] [,4]
#> [1,]    1    2    3    4
#> [2,]    5    6    7    8
#> [3,]    9   10   11   12
```

Rolling along the columns, `axis = 2`, moves elements within each row. `x` has dimensions `c(3, 4)`, so `n` is broadcast to `c(3, 1)`, a one column matrix with one `n` per row:

```
n <- matrix(c(1L, 0L, -1L), ncol = 1)

rray_roll_each(x, n = n, axis = 2)
#>      [,1] [,2] [,3] [,4]
#> [1,]    4    1    2    3
#> [2,]    5    6    7    8
#> [3,]   10   11   12    9
```

The first row moves 1 toward the end, the second row stays put, and the third row moves 1 toward the start.

Rolling along the rows, `axis = 1`, moves elements within each column. Now `n` is broadcast to `c(1, 4)`, so it must be a one row matrix:

```
rray_roll_each(x, n = matrix(c(0L, 1L, 2L, 3L), nrow = 1), axis = 1)
#>      [,1] [,2] [,3] [,4]
#> [1,]    1   10    7    4
#> [2,]    5    2   11    8
#> [3,]    9    6    3   12
```

The last column has an `n` of 3 on a dimension of 3, a full circle, so it doesn't move.

Three dimensions:

Three dimensions get a bit more complicated, but demonstrate why `n` must be an array. Think of an array with dimensions `c(2, 5, 3)` as 3 tables, each with 2 rows and 5 columns:

```
x <- array(1:30, c(2, 5, 3))

x
#> , , 1
#>
#>      [,1] [,2] [,3] [,4] [,5]
```

```
#> [1,]    1    3    5    7    9
#> [2,]    2    4    6    8   10
#>
#> , , 2
#>
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]   11   13   15   17   19
#> [2,]   12   14   16   18   20
#>
#> , , 3
#>
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]   21   23   25   27   29
#> [2,]   22   24   26   28   30
```

Rolling along the columns, `axis = 2`, moves elements within each row. There are 2 rows on each of 3 tables, so there are 6 separate rows that can each roll by their own `n`. A vector can't hold one `n` for each of them, but an array can. `n` is broadcast to `c(2, 1, 3)`, which is one `n` for each row on each table:

```
n <- array(1:6, c(2, 1, 3))

n
#> , , 1
#>
#>      [,1]
#> [1,]    1
#> [2,]    2
#>
#> , , 2
#>
#>      [,1]
#> [1,]    3
#> [2,]    4
#>
#> , , 3
#>
#>      [,1]
#> [1,]    5
#> [2,]    6

rray_roll_each(x, n = n, axis = 2)
#> , , 1
#>
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    9    1    3    5    7
#> [2,]    8   10    2    4    6
#>
#> , , 2
```

```
#>
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]   15   17   19   11   13
#> [2,]   14   16   18   20   12
#>
#> , , 3
#>
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]   21   23   25   27   29
#> [2,]   30   22   24   26   28
```

Row 1 of table 1 moves 1, row 2 of table 1 moves 2, and so on. Row 1 of table 3 has an `n` of 5 on a dimension of 5, a full circle, so it doesn't move. Row 2 of table 3 has an `n` of 6, which circles around to 1.

Broadcasting allows you to provide a smaller `n` array when either all of the rows or all of the tables roll by the same amount:

```
# One `n` per row, the same on every table
rray_roll_each(x, n = array(c(1, 2), c(2, 1, 1)), axis = 2)

# One `n` per table, the same for both rows
rray_roll_each(x, n = array(c(0, 1, 2), c(1, 1, 3)), axis = 2)
```

Examples

```
# Positive `n` moves elements toward the end
rray_roll(1:5, n = 2, axes = 1)

# Negative `n` moves them toward the start
rray_roll(1:5, n = -1, axes = 1)

# Large `n` circles around
rray_roll(1:5, n = 7, axes = 1)

x <- matrix(1:6, nrow = 2, dimnames = list(c("r1", "r2"), c("a", "b", "c")))

# Roll the columns, names move with the data
rray_roll(x, n = 1, axes = 2)

# Roll the rows
rray_roll(x, n = 1, axes = 1)

# Roll both axes by the same `n`
rray_roll(x, n = 1, axes = c(1, 2))

# Or by one `n` per axis
rray_roll(x, n = c(1, -1), axes = c(1, 2))

y <- matrix(1:12, nrow = 3, byrow = TRUE)

# Roll each row by its own `n`, given as a one column matrix
rray_roll_each(y, n = matrix(c(1, 0, -1), ncol = 1), axis = 2)
```

```
# Roll each row by its own `n`, by title
z <- rray_set_titles(y, c("row", "col"))
rray_roll_each(z, n = matrix(c(1, 0, -1), ncol = 1), axis = "col")

# Roll each column by its own `n`, given as a one row matrix
rray_roll_each(y, n = matrix(c(0, 1, 2, 3), nrow = 1), axis = 1)

# Names on `axis` are dropped, since no single name fits each position
rray_roll_each(x, n = matrix(c(1, 2), ncol = 1), axis = 2)
```

`rray-titles` *Array titles*

Description

Titles are the names on an array's list of names. They appear in array print output.

- `rray_titles()` returns the titles for every axis.
- `rray_axis_title()` returns the title for one axis.
- `rray_set_titles()` sets the titles for every axis.
- `rray_set_axis_title()` sets the title for one axis.

Usage

```
rray_titles(x)

rray_axis_title(x, axis)

rray_set_titles(x, titles)

rray_set_axis_title(x, axis, title)
```

Arguments

`x`	An array.
`axis`	The axis to get or set the title for. • For `rray_set_axis_title()`, given by position or by title. • For `rray_axis_title()`, given by position.
`titles`	A character vector with one title per axis, or `NULL` to remove all titles.
`title`	A single character string, or `NULL` to remove the title for `axis`.

Value

- `rray_titles()` returns a character vector, or `NULL` if there are no titles.
- `rray_axis_title()` returns one character string, or `NULL` if there are no titles. It returns `""` when the titles vector exists but that axis has no title.
- The setters return `x` with new titles. Plain vectors become one-dimensional arrays.

Examples

```
x <- matrix(1:6, 2, 3)
x <- rray_set_titles(x, c("Row", "Column"))
rray_titles(x)
rray_axis_title(x, 2)
rray_set_axis_title(x, 1, NULL)
rray_set_axis_title(x, "Row", "R")
```

---

rray_axis_names *Get names for a single axis of an array*

Description

- `rray_axis_names()` returns the names for a single `axis` of an array, or `NULL` if that axis has no names.
- `rray_row_names()` and `rray_column_names()` are shortcuts for `rray_axis_names(x, 1)` and `rray_axis_names(x, 2)`.

Usage

```
rray_axis_names(x, axis)

rray_row_names(x)

rray_column_names(x)
```

Arguments

`x`	An array.
`axis`	The axis to get names for, given by position or by title.

Value

A character vector of names, or `NULL` if that axis has no names.

Examples

```
x <- array(1:6, c(2, 3), dimnames = list(c("r1", "r2"), c("c1", "c2", "c3")))

rray_axis_names(x, 1)
rray_axis_names(x, 2)

# By title
y <- rray_set_titles(x, c("row", "col"))
rray_axis_names(y, "col")

rray_row_names(x)
rray_column_names(x)
```

rray_broadcast *Broadcast an array to new dimensions*

Description

`rray_broadcast()` broadcasts an array to a new set of dimensions using tidyverse recycling rules. Each dimension of `x` must either match the corresponding target dimension or be 1, in which case it is repeated to fill the target.

Dimensionality can be expanded by supplying `dimensions` with greater dimensionality than `x` has. For example, a 2x3 array can be broadcast to a 2x3x4 array.

Usage

```
rray_broadcast(x, dimensions)
```

Arguments

`x`	An array.
`dimensions`	An integer vector of target dimensions.

Value

An array with dimensions of `dimensions`.

Examples

```
# Broadcast a vector to a matrix
rray_broadcast(1:3, c(3L, 2L))

# Broadcast a row to fill a matrix
rray_broadcast(array(1:2, c(1L, 2L)), c(3L, 2L))

# Add a new dimension
rray_broadcast(array(1:6, c(2L, 3L)), c(2L, 3L, 4L))
```

rray_broadcast_common *Broadcast arrays to common dimensions*

Description

`rray_broadcast_common()` broadcasts every array to the common dimensions found by `rray_dimensions_common()`.

Usage

```
rray_broadcast_common(..., .dimensions = NULL)
```

Arguments

`...`	Arrays to broadcast.
`.dimensions`	If provided, an integer vector of dimensions to broadcast to, rather than computing common dimensions from `...`.

Value

A list of arrays, all with the common dimensions.

Examples

```
rray_broadcast_common(array(1:3, c(3L, 1L)), array(1:2, c(1L, 2L)))

# Names of `...` are kept
rray_broadcast_common(x = 1:3, y = 1L)

# `.dimensions` overrides the common dimensions
rray_broadcast_common(1:3, .dimensions = c(3L, 2L))
```

---

rray_combine *Combine arrays along an existing axis*

Description

`rray_combine()` joins one or more arrays along `.axis`. Dimensions on every other axis are broadcast to common dimensions.

Usage

```
rray_combine(..., .axis)
```

Arguments

`...`	Arrays to combine.
`.axis`	The axis to combine along, given by position. Must be between 1 and the greatest input dimensionality.

Details

This function participates in rray's common name determination rules. Names on `.axis` are never checked. They are pasted together from every input, with `""` for inputs that have none.

Value

An array with the following dimensions:

- Along `.axis`, the input dimensions are added together.
- Along all other axes, the common dimensions are taken via broadcasting.

Examples

```
x <- array(1:6, c(2, 3))
y <- array(7:12, c(2, 3))

rray_combine(x, y, .axis = 1)
rray_combine(x, y, .axis = 2)

# Missing trailing axes have an implicit dimension of 1
rray_combine(1:2, array(3:8, c(2, 3)), .axis = 2)
```

---

rray_dimensionality *Find the dimensionality of an array*

Description

`rray_dimensionality()` returns the number of dimensions of an array as a single integer.

Usage

```
rray_dimensionality(x)
```

Arguments

x	An array.

Value

A single integer representing the number of dimensions.

Examples

```
rray_dimensionality(1)
rray_dimensionality(array(1, c(2, 3)))
rray_dimensionality(array(1, c(2, 3, 4)))
```

---

rray_dimensions *Get the dimensions of an array*

Description

`rray_dimensions()` returns the dimension of each axis of an array.

Usage

```
rray_dimensions(x)
```

Arguments

x	An array.

Details

For a plain vector without a dim attribute, this returns its length as a single integer (i.e., a 1-dimensional result).

Value

An integer vector of dimensions.

Examples

```
rray_dimensions(1:5)
rray_dimensions(array(1, c(2, 3)))
rray_dimensions(array(1, c(2, 3, 4)))
```

---

rray_dimensions_common

Find common dimensions

Description

rray_dimensions_common() finds the common dimensions among multiple arrays using broadcasting rules. For each axis, dimensions are compatible if they are equal or if one of them is 1.

Usage

```
rray_dimensions_common(..., .dimensions = NULL)
```

Arguments

...	Arrays.
.dimensions	If provided, an integer vector of dimensions to use as an override, rather than computing common dimensions from ....

Value

An integer vector of common dimensions.

Examples

```
rray_dimensions_common(array(1, c(2, 3)), array(1, c(1, 3)))
rray_dimensions_common(1:5, array(1, c(1, 3)))
rray_dimensions_common(1:5, .dimensions = c(5L, 3L))
```

`rray_extract` *Extract values from an array*

Description

`rray_extract()` selects values from `x` by 1D location or by coordinate point. The result is always a 1D array.

`rray_extract_assign()` replaces the selected values with `value` and returns a modified copy of `x`.

Usage

```
rray_extract(x, i)

rray_extract_assign(x, i, value)
```

Arguments

`x`	An array.
`i`	A numeric vector of locations, a logical vector, a logical array with the same dimensions as `x`, or a numeric matrix of coordinate points.
`value`	A 1D array to assign to the selected values. It is cast to the type of `x`, then broadcast to the number of selected or missing positions. It must be size 1 or have one element per position.

Details

The type and shape of `i` decide how it is used:

- An integer or double vector holds 1D locations. `x` is treated as its equivalent flattened array, so location `4` in a 2 x 3 matrix is row 2, column 2. Locations follow the usual R rules: negative values drop elements, zero is ignored, duplicates repeat elements, and `NA` gives a missing value.
- A logical vector must be size 1 or the size of `x`. A logical array must have the same dimensions as `x`, so `rray_extract(x, x > 5)` works. When `TRUE`, the corresponding value in `x` is returned.
- An integer or double matrix holds coordinate points. Each row selects one element, and there must be one column per axis of `x`. Coordinates must be positive and within the dimension of their axis. `NA` at any position within the row generates a missing value.

For raw arrays the missing value is `as.raw(0)`, and for list arrays it is `NULL`.

For assignment, a missing entry in `i` leaves `x` unchanged at that position but uses one element of `value`. This also applies to a point row with a missing coordinate. Zero locations use no element of `value`. Negative locations can't be combined with missing values.

Value

- `rray_extract()` returns a one-dimensional array. Names are always dropped.
- `rray_extract_assign()` returns `x` with the selected values replaced. The type, dimensions, and names of `x` are kept.

Examples

```
x <- array(1:24, c(2L, 3L, 4L))

# 1D locations
rray_extract(x, c(1, 4, 24))

# Negative locations drop elements
rray_extract(x, -(1:20))

# A logical mask with the same dimensions as `x`
rray_extract(x, x %% 5 == 0)

# One row per point, one column per axis
points <- rbind(
  c(1, 1, 1),
  c(2, 3, 4),
  c(1, 2, 3)
)
rray_extract(x, points)

# Assign one value to every selected element
rray_extract_assign(x, x %% 5 == 0, 0L)

# Or one value per selected element
rray_extract_assign(x, points, c(100L, 200L, 300L))
```

---

rray_if_else *If-else for arrays*

Description

`rray_if_else()` selects values from `true` and `false` using a logical `condition`. When an element of `condition` is `NA`, it selects from `missing`, or returns a missing value if `missing` is `NULL`.

All supplied inputs are broadcast to common dimensions. The common output type comes from `true`, `false`, and `missing`.

Usage

```
rray_if_else(condition, true, false, ..., missing = NULL, dimensions = NULL)
```

Arguments

condition	A logical array.
true	Values to use where condition is TRUE.
false	Values to use where condition is FALSE.
...	These dots are for future extensions and must be empty.
missing	Values to use where condition is NA. If NULL, missing conditions produce missing values of the output type.
dimensions	An optional integer vector of output dimensions. When supplied, every input must broadcast to these dimensions. Use rray_dimensions(condition) to keep the shape of condition.

Value

An array with the common dimensions of the inputs, or dimensions when supplied. Its type comes from true, false, and missing. Names are dropped.

Examples

```
x <- array(1:12, c(3L, 4L))
x[1, 1] <- NA_integer_
y <- array(101:104, c(1L, 4L))
rray_if_else(x > 5L, x, y, missing = 0L)

condition <- array(c(TRUE, FALSE), c(2L, 1L))
true <- array(1:6, c(2L, 3L))
false <- array(11:16, c(2L, 3L))
rray_if_else(condition, true, false)
```

---

rray_index *Index an array by coordinates*

Description

rray_index() selects values from x with one coordinate array for each source axis. The coordinate arrays broadcast to common dimensions and are read pointwise. Their common dimensions become the dimensions of the result.

Usage

```
rray_index(x, ...)
```

Arguments

x	A bare array or vector.
...	Exactly one unnamed integer coordinate array for each axis of x.

Details

Each argument in `...` identifies positions on one axis of `x`. The first argument identifies positions on the first axis, the second argument identifies positions on the second axis, and so on. You must supply exactly one unnamed coordinate array for every axis of `x`.

A coordinate array must be a bare integer vector or array. Vectors are treated as one-dimensional arrays. Each coordinate must be a positive, one-based position within its source axis. Zero, negative, and out-of-bounds coordinates are errors.

Coordinate arrays use left-aligned broadcasting. At each point in the common dimensions, `rray_index()` takes one coordinate from each broadcast array and uses the complete set of coordinates to select one value from `x`. Singleton dimensions can be placed on different axes to form a Cartesian product.

A missing coordinate on any source axis produces a missing value at that result point. For raw arrays the missing value is `as.raw(0)`, and for list arrays it is `NULL`.

The result has the same storage type as `x`. Its dimensions are the common dimensions of the coordinate arrays. All names are dropped because a result axis does not necessarily correspond to one source axis.

Value

An array with the common dimensions of `...` and the same storage type as `x`.

Examples

```
x <- array(1:6, c(2L, 3L))

# Pair coordinates pointwise
rray_index(x, c(1L, 2L, 1L), c(1L, 2L, 3L))

# Form a Cartesian product through broadcasting
rows <- array(c(1L, 2L), c(2L, 1L))
columns <- array(1:3, c(1L, 3L))
rray_index(x, rows, columns)

# Any missing coordinate produces a missing result
rray_index(x, c(1L, NA_integer_), c(2L, 3L))

# Coordinate lists can be spliced into `...`
coordinates <- list(rows, columns)
rray_index(x, !!!coordinates)
```

---

rray_insert_axes *Insert array axes*

Description

`rray_insert_axes()` inserts new axes with a dimension of 1.

Usage

```
rray_insert_axes(x, axes)
```

Arguments

x	An array.
axes	The axes to insert, given by position in the output.

Details

`axes` refer to axes of the result, not axes of `x`. Inserting 2 axes into an array with a dimensionality of 3 gives a result with a dimensionality of 5 (2 + 3), so `axes` can be any value between 1 and 5.

To work out the result, write out the axes of the result and mark the ones listed in `axes` with 1. Then use the dimensions of `x` to fill in the rest in order.

```
x       (2, 3, 4)
axes    2

result  (x, 1, x, x)
      = (2, 1, 3, 4)
```

This allows you to insert two axes side by side:

```
x       (2, 3, 4)
axes    c(2, 3)

result  (x, 1, 1, x, x)
      = (2, 1, 1, 3, 4)
```

Inserted axes have no names and an empty title. The axes of `x` keep their names and titles and carry them to their new locations.

Value

An array with new axes of dimension 1 at `axes`.

See Also

`rray_remove_axes()`

Examples

```
x <- array(1:24, c(2, 3, 4))

# Insert one axis in the middle
# (2, 3, 4) -> (2, 1, 3, 4)
rray_dimensions(rray_insert_axes(x, 2))

# Insert at the front and at the back
```

```
# (2, 3, 4) -> (1, 2, 3, 4, 1)
rray_dimensions(rray_insert_axes(x, c(1, 5)))

# Two new axes side by side
# (2, 3, 4) -> (2, 1, 1, 3, 4)
rray_dimensions(rray_insert_axes(x, c(2, 3)))

# Inserting no axes returns `x` unchanged
rray_insert_axes(x, integer())

# `rray_remove_axes()` undoes an insertion
rray_remove_axes(rray_insert_axes(x, 2), 2)
```

`rray_move_axes` *Move array axes*

Description

`rray_move_axes()` takes axes `from` `x` and moves them `to` axes in the output. The axes that don't move keep their relative order, and fill in the locations that are left over.

Usage

```
rray_move_axes(x, ..., from, to)
```

Arguments

`x`	An array.
`...`	These dots are for future extensions and must be empty.
`from`	The axes to move, given by position.
`to`	The axes to move to, given by position in the output.

Details

Names and titles travel with their axis to its new position.

Value

An array.

See Also

`rray_permute_axes()`

Examples

```
x <- array(1:24, c(2, 3, 4))

# Move the first axis to the end
# (2, 3, 4) -> (3, 4, 2)
rray_dimensions(rray_move_axes(x, from = 1, to = 3))

# Move the last axis to the front
# (2, 3, 4) -> (4, 2, 3)
rray_dimensions(rray_move_axes(x, from = 3, to = 1))

# Move two axes at once, the axis that stays fills in what is left
# (2, 3, 4) -> (4, 2, 3)
rray_dimensions(rray_move_axes(x, from = c(1, 3), to = c(2, 1)))

# Swap the first two axes
# (2, 3, 4) -> (3, 2, 4)
rray_dimensions(rray_move_axes(x, from = c(1, 2), to = c(2, 1)))
```

---

rray_names *Get names for each axis of an array*

Description

`rray_names()` returns the names for each axis of an array, or `NULL` if `x` has no names at all. Unlike `dimnames()`, it returns a one-element list for named vectors.

Usage

```
rray_names(x)
```

Arguments

`x`	An array.

Value

Either:

- A list with length equal to the dimensionality of `x`, where each element is either a character vector of names or `NULL`.
- `NULL` if `x` has no names.

Examples

```
# No names at all
rray_names(1:3)
# Named vectors return a one-element list
rray_names(c(a = 1, b = 2))

# All dimension names
x <- array(1:6, c(2, 3), dimnames = list(c("r1", "r2"), c("c1", "c2", "c3")))
rray_names(x)
```

---

rray_permute_axes *Permute array axes*

Description

`rray_permute_axes()` reorders the axes of an array.

Usage

```
rray_permute_axes(x, axes)
```

Arguments

`x`	An array.
`axes`	The axes of `x` in their new order, given by position or by title. It must use each axis of `x` exactly once.

Details

Names and titles travel with their axis to its new position.

`rray_permute_axes(x, c(2, 1))` transposes a matrix, like `t()`.

Value

An array with the axes of `x` reordered by `axes`.

See Also

`rray_move_axes()`

Examples

```
x <- array(1:6, c(2, 3))

# Transpose a matrix
rray_permute_axes(x, c(2, 1))

# Transpose a matrix, by title
z <- rray_set_titles(x, c("row", "col"))
rray_permute_axes(z, c("col", "row"))

y <- array(1:24, c(4, 3, 2))

# Reverse every axis
# (4, 3, 2) -> (2, 3, 4)
rray_dimensions(rray_permute_axes(y, c(3, 2, 1)))

# Swap the first two axes, leaving the third alone
# (4, 3, 2) -> (3, 4, 2)
rray_dimensions(rray_permute_axes(y, c(2, 1, 3)))
```

---

rray_remove_axes *Remove array axes*

Description

`rray_remove_axes()` removes axes with a dimension of 1.

Usage

```
rray_remove_axes(x, axes)
```

Arguments

`x`	An array.
`axes`	The axes to remove, given by position or by title. Each selected axis must have a dimension of 1. At least one axis must remain.

Details

Removed axes lose their names and titles. Surviving axes keep their names and titles and carry them to their new locations.

Value

An array with the selected `axes` removed.

Examples

```
x <- array(1:10, c(10, 1, 1))

rray_remove_axes(x, 2)
rray_remove_axes(x, c(2, 3))

y <- rray_set_titles(x, c("row", "col", "page"))
rray_remove_axes(y, c("col", "page"))
```

---

rray_reverse *Reverse elements along axes*

Description

`rray_reverse()` reverses the order of the elements of `x` along one or more `axes`.

Usage

```
rray_reverse(x, ..., axes)
```

Arguments

`x`	An array.
`...`	These dots are for future extensions and must be empty.
`axes`	The axes to reverse along, given by position or by title.

Details

Names on a reversed axis move with the data. Every other axis keeps its names untouched. The titles of `x` are kept.

Value

An array with the same type and dimensions as `x`.

Examples

```
rray_reverse(1:5, axes = 1)

x <- matrix(1:6, nrow = 2, dimnames = list(c("r1", "r2"), c("a", "b", "c")))

# Reverse the rows, names move with the data
rray_reverse(x, axes = 1)

# Reverse the columns
rray_reverse(x, axes = 2)

# Reverse both axes
```

```
rray_reverse(x, axes = c(1, 2))

# Reverse the columns, by title
y <- rray_set_titles(x, c("row", "col"))
rray_reverse(y, axes = "col")
```

`rray_set_axis_names` *Set names for a single axis of an array*

Description

- `rray_set_axis_names()` sets the names for a single `axis` of an array, leaving every other axis untouched. The titles of `x` are kept.
- `rray_set_row_names()` and `rray_set_column_names()` are shortcuts for `rray_set_axis_names(x, 1, names)` and `rray_set_axis_names(x, 2, names)`.

Usage

```
rray_set_axis_names(x, axis, names)

rray_set_row_names(x, names)

rray_set_column_names(x, names)
```

Arguments

`x`	An array.
`axis`	The axis to set names for, given by position or by title.
`names`	A character vector of names for `axis`, the same length as the dimension of `axis`. Can also be `NULL` to remove names from `axis`.

Value

`x` with new names for `axis`.

Examples

```
x <- array(1:6, c(2, 3))

rray_set_axis_names(x, 1, c("r1", "r2"))
rray_set_axis_names(x, 2, c("c1", "c2", "c3"))

# By title
y <- rray_set_titles(x, c("row", "col"))
rray_set_axis_names(y, "col", c("c1", "c2", "c3"))

rray_set_row_names(x, c("r1", "r2"))
rray_set_column_names(x, c("c1", "c2", "c3"))
```

rray_set_dimensions *Set the dimensions of an array*

Description

`rray_set_dimensions()` sets the dimensions of `x` to a new set of dimensions without changing the total number of elements. Unlike `rray_broadcast()`, which repeats elements to fill new dimensions, `rray_set_dimensions()` simply reinterprets the existing elements under new dimensions without changing its size.

Usage

```
rray_set_dimensions(x, dimensions)
```

Arguments

`x`	An array.
`dimensions`	An integer vector of new dimensions.

Value

An array with new `dimensions` but the same size as `x`.

Examples

```
x <- 1:6

# Set the dimensions to turn a vector into a matrix
rray_set_dimensions(x, c(2L, 3L))

# Set the dimensions to turn a vector into a 3D array
rray_set_dimensions(x, c(3L, 2L, 1L))

# Setting dimensions can't change the size
try(rray_set_dimensions(x, c(6L, 2L)))
```

rray_set_names *Set names for every axis of an array*

Description

`rray_set_names()` sets the names for every axis of an array at once.

Usage

```
rray_set_names(x, names)
```

Arguments

x	An array.
names	A list with length equal to the dimensionality of x, where each element is either a character vector of names for that axis or NULL. Can also be NULL to remove all names from x. The names of names become the titles of x.

Value

x with new names.

Examples

```
x <- array(1:6, c(2, 3))

rray_set_names(x, list(c("r1", "r2"), c("c1", "c2", "c3")))

# `NULL` clears all names
y <- rray_set_names(x, list(c("r1", "r2"), NULL))
rray_set_names(y, NULL)
```

---

rray_size *Find the size of an array*

Description

rray_size() returns the total number of elements in an array, computed as the product of its dimensions.

Usage

```
rray_size(x)
```

Arguments

x	An array.

Value

A single double representing the total number of elements.

Examples

```
rray_size(1:5)
rray_size(array(1, c(2, 3)))
rray_size(array(1, c(2, 3, 4)))
```

`rray_slice` *Slice an array*

Description

`rray_slice()` selects positions along each axis of `x`. The result always has the same dimensionality as `x`.

`rray_slice_axis()` is `rray_slice()` restricted to a single axis, which can be more ergonomic to write. `rray_slice_rows()` and `rray_slice_columns()` are shortcuts for `axis = 1` and `axis = 2`.

The `_assign` versions replace the selected positions with `value` and return a modified copy of `x`.

Usage

```
rray_slice(x, ...)

rray_slice_axis(x, i, ..., axis)

rray_slice_rows(x, i)

rray_slice_columns(x, i)

rray_slice_assign(x, ..., value)

rray_slice_assign_axis(x, i, ..., axis, value)

rray_slice_assign_rows(x, i, value)

rray_slice_assign_columns(x, i, value)
```

Arguments

`x`	An array.
`...`	One unnamed subscript for each axis of `x`, in axis order. Each subscript is one of: • `TRUE`, to select the whole axis. • A logical vector the size of the axis. `TRUE` selects a position. • An integer or double vector of locations. Negative values drop positions, zero is ignored, and duplicates repeat positions. • A character vector of names, matched against the names of the axis. The first match is used when names are duplicated. • `NULL`, to select nothing. For `rray_slice_axis()` and `rray_slice_assign_axis()`, these dots must be empty.
`i`	A subscript for the selected axis. It takes any of the forms allowed in `...`.

axis	The axis to slice along, given by position or by title.
value	An array to assign to the selected positions. It is cast to the type of x, then broadcast to the dimensions of the selection.

Details

NA in a subscript gives missing values in the result. For raw arrays the missing value is as.raw(0), and for list arrays it is NULL. If the axis has names, the name of a missing value is "". For assignment, NA in a subscript leaves x unchanged at that position but uses one element of value.

Unlike [, an empty argument does not select a whole axis. Use TRUE instead, so that every axis is provided explicitly:

```
x <- array(1:24, c(2L, 3L, 4L))

# Base R
x[1, , , drop = FALSE]

# rray
rray_slice(x, 1, TRUE, TRUE)

# Or, more simply
rray_slice_axis(x, 1, axis = 1)
rray_slice_rows(x, 1)
```

Value

- rray_slice(), rray_slice_axis(), rray_slice_rows(), and rray_slice_columns() return an array with the same type and dimensionality as x. The dimension of each axis is the number of positions selected on it. The names of each axis are selected along with the values. The titles of x are kept.
- The _assign versions return x with the selected positions replaced. The type, dimensions, and names of x are kept.

Examples

```
x <- array(1:24, c(2L, 3L, 4L))

# The first row
rray_slice(x, 1, TRUE, TRUE)

# Reorder and repeat positions
rray_slice(x, c(2, 1, 2), 1, 4)

# Drop positions with negative locations
rray_slice(x, TRUE, -2, -(1:2))

# Select by name
y <- array(1:6, c(2L, 3L), dimnames = list(c("a", "b"), c("c", "d", "e")))
rray_slice(y, "b", c("e", "c"))
```

```
# Subscripts can be spliced into `...`
subscripts <- rep(list(TRUE), rray_dimensionality(x))
subscripts[[3]] <- c(4, 1)
rray_slice(x, !!!subscripts)

# Slice a single axis
rray_slice_axis(x, c(4, 1), axis = 3)
rray_slice_rows(y, c("b", "a"))
rray_slice_columns(x, -2)

# Slice a single axis, by title
z <- rray_set_titles(y, c("row", "col"))
rray_slice_axis(z, c("e", "c"), axis = "col")

# Assign one value to every first row
rray_slice_assign(x, 1, TRUE, TRUE, value = 0L)
rray_slice_assign_rows(x, 1, 0L)

# Or broadcast `value` to the dimensions of the selection
value <- array(c(100L, 200L), c(1L, 2L))
rray_slice_assign(x, TRUE, c(1, 3), 1, value = value)

value <- array(c(100L, 200L), c(2L, 1L))
rray_slice_assign_axis(x, 3, axis = 2, value = value)
```

`rray_split` *Split an array along an axis*

Description

`rray_split()` divides `x` into contiguous arrays along an `axis`.

Using `rray_combine()` along the same `axis` reconstructs `x`.

Usage

```
rray_split(x, ..., axis, dimensions)
```

Arguments

`x`	An array.
`...`	These dots are for future extensions and must be empty.
`axis`	The axis to split along, given by position or by title.
`dimensions`	One of: • A single positive integer, used as the dimension of every array. It must evenly divide the dimension of `x` along `axis`. • A vector of positive (or zero) integers, giving the dimension of each array directly. They must sum to the dimension of `x` along `axis`.

Value

A list of arrays each with the same dimensions as `x`, except along `axis`.

Examples

```
x <- array(1:12, c(6, 2))

# Three arrays of two rows
rray_split(x, axis = 1, dimensions = 2)

# Arrays of one and five rows
rray_split(x, axis = 1, dimensions = c(1, 5))

# One array per column
rray_split(x, axis = 2, dimensions = 1)

# One array per column, by title
y <- rray_set_titles(x, c("row", "col"))
rray_split(y, axis = "col", dimensions = 1)

# Arrays of dimension zero are allowed in the explicit form
rray_split(x, axis = 2, dimensions = c(0, 2))

# Splitting and combining along the same axis are inverses
rray_combine(!!!rray_split(x, axis = 1, dimensions = 3), .axis = 1)
```

`rray_stack` *Stack arrays along a new axis*

Description

`rray_stack()` joins arrays along a new axis inserted at `.axis`. All preexisting axes are broadcast to their common dimension.

The new axis has a dimension equal to the number of arrays you are stacking.

Usage

```
rray_stack(..., .axis)
```

Arguments

`...`	Arrays to stack.
`.axis`	The axis to stack along, given by position in the output. Must be between 1 and one greater than the maximum dimensionality of the input arrays. For example, you can stack 2D matrices along axis 3, but not axis 4.

Details

This function participates in rray's common name determination rules on every existing axis. Names of `...` become the names of the new axis.

Value

An array with the following dimensions:

- Along `.axis`, the number of inputs.
- Along all other axes, the common dimensions of the inputs taken via broadcasting.

See Also

`rray_unstack()`, `rray_combine()`

Examples

```
x <- array(1:12, c(3, 4))
y <- array(13:24, c(3, 4))

# (3, 4) -> (2, 3, 4)
rray_dimensions(rray_stack(x, y, .axis = 1))

# (3, 4) -> (3, 4, 2)
rray_stack(x, y, .axis = 3)

# Existing axes are broadcast
# (3, 1) and (3, 4) -> (2, 3, 4)
a <- array(1:3, c(3, 1))
b <- array(1:12, c(3, 4))
rray_dimensions(rray_stack(a, b, .axis = 1))

# Names of `...` name the new axis
rray_stack(first = 1:2, second = 3:4, .axis = 2)
```

---

rray_unstack *Unstack an array*

Description

`rray_unstack()` splits an array along `axis` and then removes that `axis` from each of the resulting arrays. The result is a list of arrays that have a dimensionality 1 less than `x` itself.

Using `rray_stack()` along the same `axis` reconstructs `x`.

Usage

```
rray_unstack(x, ..., axis)
```

Arguments

x	An array.
...	These dots are for future extensions and must be empty.
axis	The axis to unstack along, given by position or by title.

Details

Names along `axis` become the names of the returned list. Names on the surviving axes are carried over to their new locations, as are their titles.

Because the dimensionality is reduced by 1, `x` must have a dimensionality of at least 2 to begin with.

Value

A list of arrays with the same dimensions as `x`, except that `axis` has been removed.

See Also

`rray_stack()`

Examples

```
x <- array(1:6, c(2, 3))

# One array per row
rray_unstack(x, axis = 1)

# One array per column
rray_unstack(x, axis = 2)

# One array per column, by title
z <- rray_set_titles(x, c("row", "col"))
rray_unstack(z, axis = "col")

# Names along the axis become list names
y <- array(1:4, c(2, 2), dimnames = list(c("a", "b"), c("x", "y")))
rray_unstack(y, axis = 2)

# Unstacking and stacking along the same axis are inverses
rray_stack(!!!rray_unstack(x, axis = 2), .axis = 2)
```

Index

all(), *10*
any(), *10*

by position, *25*, *28*, *36*
by position in the output, *35, 36*, *47*
by position or by title, *10*, *14*, *16*, *19, 20*, *25, 26*, *38–41*, *45, 46*, *49*

common name determination rules, *7*, *13*, *15*, *28*, *48*

dimnames(), *37*

faq-axes, 3
faq-common-names, 4

max(), *10*, *13*, *18*
mean(), *17*
min(), *10*, *13*, *18*

pmax(), *13*
pmin(), *13*

rray-arithmetic, 7
rray-compare, 8
rray-cumulative, 9
rray-equal, 12
rray-extremum, 13
rray-locate, 14
rray-logical, 15
rray-reduce, 16
rray-rep, 18
rray-roll, 20
rray-titles, 25
rray_add *(*rray-arithmetic*)*, 7
rray_add(), *4*
rray_all *(*rray-reduce*)*, 16
rray_and *(*rray-logical*)*, 15
rray_and(), *4*
rray_any *(*rray-reduce*)*, 16
rray_axis_names, 26
rray_axis_title *(*rray-titles*)*, 25
rray_broadcast, 27
rray_broadcast(), *42*
rray_broadcast_common, 27
rray_column_names *(*rray_axis_names*)*, 26
rray_combine, 28
rray_combine(), *4*, *6*, *48*
rray_cumulative_all *(*rray-cumulative*)*, 9
rray_cumulative_any *(*rray-cumulative*)*, 9
rray_cumulative_max *(*rray-cumulative*)*, 9
rray_cumulative_min *(*rray-cumulative*)*, 9
rray_cumulative_prod *(*rray-cumulative*)*, 9
rray_cumulative_sum *(*rray-cumulative*)*, 9
rray_dimensionality, 29
rray_dimensions, 29
rray_dimensions_common, 30
rray_dimensions_common(), *27*
rray_divide *(*rray-arithmetic*)*, 7
rray_divide(), *4*
rray_equal *(*rray-equal*)*, 12
rray_equal(), *6*
rray_exponentiate *(*rray-arithmetic*)*, 7
rray_exponentiate(), *4*
rray_extract, 31
rray_extract_assign *(*rray_extract*)*, 31
rray_greater_than *(*rray-compare*)*, 8
rray_greater_than(), *6*
rray_greater_than_or_equal *(*rray-compare*)*, 8
rray_if_else, 32
rray_index, 33
rray_index(), *21*
rray_insert_axes, 34
rray_insert_axes(), *3*
rray_less_than *(*rray-compare*)*, 8
rray_less_than_or_equal *(*rray-compare*)*, 8
rray_locate_max *(*rray-locate*)*, 14

rray_locate_min *(rray-locate)*, 14
rray_max *(rray-reduce)*, 16
rray_max(), *13*
rray_mean *(rray-reduce)*, 16
rray_min *(rray-reduce)*, 16
rray_min(), *13*
rray_move_axes, 36
rray_move_axes(), *38*
rray_multiply *(rray-arithmetic)*, 7
rray_multiply(), *4*
rray_names, 37
rray_not_equal *(rray-equal)*, 12
rray_or *(rray-logical)*, 15
rray_or(), *4*
rray_permute_axes, 38
rray_permute_axes(), *36*
rray_pmax *(rray-extremum)*, 13
rray_pmax(), *4*
rray_pmin *(rray-extremum)*, 13
rray_pmin(), *4*
rray_prod *(rray-reduce)*, 16
rray_remove_axes, 39
rray_remove_axes(), *35*
rray_rep *(rray-rep)*, 18
rray_rep_each *(rray-rep)*, 18
rray_reverse, 40
rray_roll *(rray-roll)*, 20
rray_roll_each *(rray-roll)*, 20
rray_row_names *(rray_axis_names)*, 26
rray_set_axis_names, 41
rray_set_axis_title *(rray-titles)*, 25
rray_set_column_names *(rray_set_axis_names)*, 41
rray_set_dimensions, 42
rray_set_names, 42
rray_set_names(), *4*
rray_set_row_names *(rray_set_axis_names)*, 41
rray_set_titles *(rray-titles)*, 25
rray_set_titles(), *6*
rray_size, 43
rray_slice, 44
rray_slice(), *21*
rray_slice_assign *(rray_slice)*, 44
rray_slice_assign_axis *(rray_slice)*, 44
rray_slice_assign_columns *(rray_slice)*, 44
rray_slice_assign_rows *(rray_slice)*, 44
rray_slice_axis *(rray_slice)*, 44
rray_slice_columns *(rray_slice)*, 44
rray_slice_rows *(rray_slice)*, 44
rray_split, 46
rray_stack, 47
rray_stack(), *3*, *4*, *6*, *48*, *49*
rray_subtract *(rray-arithmetic)*, 7
rray_subtract(), *4*
rray_sum *(rray-reduce)*, 16
rray_sum(), *11*
rray_titles *(rray-titles)*, 25
rray_titles(), *3*
rray_unstack, 48
rray_unstack(), *48*
rray_xor *(rray-logical)*, 15
rray_xor(), *4*

t(), *38*

which.max(), *14*
which.min(), *14*

xor(), *15*