Package: phinterval (via r-universe)

October 16, 2026

Title Set Operations on Time Intervals

Version 1.0.0.9000

Description Implements the phinterval vector class for representing time spans that may contain gaps (disjoint intervals) or be empty. This class generalizes the 'lubridate' package's interval class to support vectorized set operations (intersection, union, difference, complement) that always return a valid time span, even when disjoint or empty intervals are created.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://github.com/EthanSansom/phinterval, https://ethansansom.github.io/phinterval/

BugReports https://github.com/EthanSansom/phinterval/issues

Depends R (>= 4.0.0)

Imports lubridate, methods, pillar, Rcpp, rlang, tibble, tzdb, vctrs (>= 0.7.0)

Suggests dplyr, knitr, rmarkdown, testthat (>= 3.0.0), tidyr, withr

Config/testthat/edition 3

LinkingTo Rcpp, tzdb

VignetteBuilder knitr

SystemRequirements C++17

Config/Needs/website rmarkdown

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

Date/Publication 2026-03-28 10:10:19 UTC

RemoteUrl https://github.com/ethansansom/phinterval

RemoteRef HEAD

RemoteSha fe67ef44c25824e53c333c833223d50863870669

Contents

as_duration . . . 2
as_phinterval . . . 3
flatten . . . 4
hole . . . 6
is_phinterval . . . 7
is_phintish . . . 8
is_recognized_tzone . . . 8
n_spans . . . 9
phint_discard_instants . . . 10
phint_invert . . . 11
phint_length . . . 12
phint_overlaps . . . 13
phint_sift . . . 14
phint_unnest . . . 16
phint_unoverlap . . . 18
phint_within . . . 20
phinterval . . . 21
phinterval-accessors . . . 24
phinterval-cumset-operations . . . 25
phinterval-overlap-predicates . . . 27
phinterval-set-operations . . . 30
phinterval-span-predicates . . . 32
phinterval_options . . . 33
squash . . . 34

Index **37**

`as_duration` *Convert a phinterval to a duration*

Description

`as_duration()` converts a `lubridate::interval()` or `phinterval()` vector into a `lubridate::duration()` vector. The resulting duration measures the length of time in seconds within each element of the interval or phinterval.

`as_duration()` is a wrapper around `lubridate::as.duration()`.

Usage

```
as_duration(x, ...)

## Default S3 method:
as_duration(x, ...)

## S3 method for class 'phinterval'
as_duration(x, ...)
```

Arguments

x	[phinterval / Interval] An object to convert.
...	Parameters passed to other methods. Currently unused.

Value

A <Duration> vector the same length as x.

Examples

```
monday <- interval(as.Date("2025-11-10"), as.Date("2025-11-11"))
friday <- interval(as.Date("2025-11-14"), as.Date("2025-11-15"))
mon_and_fri <- phint_union(monday, friday)

as_duration(c(mon_and_fri, monday))
as_duration(mon_and_fri) == as_duration(monday) + as_duration(friday)
```

---

as_phinterval *Convert an interval or datetime vector into a phinterval*

Description

as_phinterval() converts a lubridate::interval(), Date, POSIXct, or POSIXlt vector into an equivalent <phinterval> vector.

Usage

```
as_phinterval(x, ...)

## Default S3 method:
as_phinterval(x, ...)

## S3 method for class 'Interval'
as_phinterval(x, ...)
```

Arguments

x	[Interval / Date / POSIXct / POSIXlt] An object to convert.
...	Additional arguments passed to methods. Currently unused.

Details

Negative intervals (where start > end) are standardized to positive intervals via `lubridate::int_standardize()`.

Datetime vectors (Date, POSIXct, POSIXlt) are converted into instantaneous intervals where the start and end are identical.

Spans with partially missing endpoints (e.g., `interval(NA, end)` or `interval(start, NA)`) are converted to a fully `NA` element.

Value

A `<phinterval>` vector the same length as `x`.

See Also

`phinterval()`

Examples

```
# Convert Interval vector
years <- interval(
  start = as.Date(c("2021-01-01", "2023-01-01")),
  end = as.Date(c("2022-01-01", "2024-01-01"))
)
as_phinterval(years)

# Negative intervals are standardized
negative <- interval(as.Date("2000-10-11"), as.Date("2000-10-01"))
as_phinterval(negative)

# Partially missing endpoints become fully NA
partial_na <- interval(NA, as.Date("1999-08-02"))
as_phinterval(partial_na)

# Datetime vectors become instantaneous intervals
as_phinterval(as.Date(c("2000-10-11", "2001-05-03")))
```

`flatten` *Flatten a phinterval vector into a vector of spans or gaps*

Description

`phint_flatten()` and `datetime_flatten()` collapse all elements into a single set of non-overlapping time spans, then return them as a flat `<phinterval>` vector with one span per element. `NA` elements are ignored.

- `phint_flatten()` takes a `<phinterval>` or `<Interval>` vector.
- `datetime_flatten()` takes separate `start` and `end` datetime vectors.
- `what = "spans"` (default): returns the time spans covered by any element of `phint`.
- `what = "holes"`: returns the gaps between those spans.

Usage

```
phint_flatten(phint, what = c("spans", "holes"))

datetime_flatten(start, end, what = c("spans", "holes"))
```

Arguments

phint	[phinterval / Interval] A <phinterval> or <Interval> vector.
what	["spans" / "holes"] Whether to return the covered spans or the intervening gaps: • "spans" (default): Time spans covered by at least one element of phint. • "holes": Gaps between covered spans (excludes the infinite extents before the first span and after the last span).
start	[POSIXct / POSIXlt / Date] A vector of start times. Must be recyclable with end. Only used in datetime_flatten().
end	[POSIXct / POSIXlt / Date] A vector of end times. Must be recyclable with start. Only used in datetime_flatten().

Value

A <phinterval> vector with the invariant all(n_spans(result) == 1L).

See Also

phint_squash() and datetime_squash() to collapse into a single <phinterval> element rather than a vector of scalar spans.

Examples

```
monday <- interval(as.Date("2025-11-10"), as.Date("2025-11-11"))
tuesday <- interval(as.Date("2025-11-11"), as.Date("2025-11-12"))
friday <- interval(as.Date("2025-11-14"), as.Date("2025-11-15"))
thurs_and_sat <- phint_union(
  interval(as.Date("2025-11-13"), as.Date("2025-11-14")),
  interval(as.Date("2025-11-15"), as.Date("2025-11-16"))
)
noon_wednesday <- as_phinterval(as.POSIXct("2025-11-12 12:00:00"))

# phint_flatten: flatten a phinterval/Interval vector
phint_flatten(c(monday, thurs_and_sat))

# datetime_flatten: flatten from start/end vectors
datetime_flatten(
  start = as.Date(c("2025-11-10", "2025-11-13", "2025-11-15")),
  end = as.Date(c("2025-11-11", "2025-11-14", "2025-11-16"))
)

# Flatten into gaps between spans
```

```
phint_flatten(c(monday, thurs_and_sat), what = "holes")
phint_flatten(thurs_and_sat, what = "holes") == friday

# Overlapping or adjacent elements are merged before flattening
phint_flatten(c(monday, tuesday, friday))

# NA elements are ignored
phint_flatten(c(monday, NA, friday))
phint_flatten(interval(NA, NA))

# Instants are preserved when flattening into spans
phint_flatten(c(monday, noon_wednesday, friday), what = "spans")

# Instants between two spans are ignored when flattening into gaps
phint_flatten(c(monday, noon_wednesday, friday), what = "holes")
```

`hole` *Create a hole phinterval*

Description

`hole()` creates a <phinterval> vector where each element is a hole (an empty set of time spans).

Usage

```
hole(n = 1L, tzone = "")
```

Arguments

`n`	`[integer(1)]`
	The number of hole elements to create. Must be a positive whole number.
`tzone`	`[character(1)]`
	A time zone to display the <phinterval> in. Defaults to `""`.

Details

A hole is a phinterval element with zero time spans, representing an empty interval. Holes are useful as placeholders or for representing the absence of time periods in interval algebra operations.

Value

A <phinterval> vector of length `n` where each element is a <hole>.

Examples

```
# Create a single hole
hole()

# Create multiple holes
hole(3)

# Specify time zone
hole(tzone = "UTC")

# Holes can be combined with other phintervals
jan <- phinterval(as.Date("2000-01-01"), as.Date("2000-02-01"))
c(jan, hole(), jan)
```

---

is_phinterval *Test if the object is a phinterval*

Description

This function returns TRUE for <phinterval> vectors and returns FALSE otherwise.

Usage

```
is_phinterval(x)
```

Arguments

x	An object to test.

Value

TRUE if x is a <phinterval>, FALSE otherwise.

Examples

```
is_phinterval(phinterval())
is_phinterval(interval())
```

is_phintish *Test if the object is a phinterval or interval*

Description

This function returns TRUE for <phinterval> and <Interval> vectors and returns FALSE otherwise.

Usage

```
is_phintish(x)
```

Arguments

x	An object to test.

Value

TRUE if x is a <phinterval> or <Interval>, FALSE otherwise.

Examples

```
is_phinterval(phinterval())
is_phinterval(interval())
is_phinterval(as.Date("2020-01-01"))
```

is_recognized_tzone *Test if the object is a recognized time zone*

Description

is_recognized_tzone() returns TRUE for strings that are recognized IANA time zone names, and FALSE otherwise.

Usage

```
is_recognized_tzone(x)
```

Arguments

x	An object to test.

Details

Recognized time zones are those listed in tzdb::tzdb_names(), which provides an up-to-date copy of time zones from the IANA time zone database.
<phinterval> vectors with an unrecognized time zone are formatted using the "UTC" time zone with a warning.

Value

TRUE if x is a recognized time zone, FALSE otherwise.

Examples

```
is_recognized_tzone("UTC")
is_recognized_tzone("America/New_York")
is_recognized_tzone("")
is_recognized_tzone("badzone")
is_recognized_tzone(10L)
```

---

n_spans *Count the number of spans in a phinterval*

Description

n_spans() counts the number of disjoint time spans in each element of phint.

Usage

```
n_spans(phint)

## Default S3 method:
n_spans(phint)

## S3 method for class 'Interval'
n_spans(phint)

## S3 method for class 'phinterval'
n_spans(phint)
```

Arguments

phint	[phinterval / Interval] A <phinterval> or <Interval> vector.

Value

An integer vector the same length as phint.

Examples

```
# Count spans
y2000 <- interval(as.Date("2000-01-01"), as.Date("2001-01-01"))
y2025 <- interval(as.Date("2025-01-01"), as.Date("2025-01-01"))

n_spans(c(
```

```
 phint_union(y2000, y2025),
 phint_intersect(y2000, y2025),
 y2000, y2025
))
```

`phint_discard_instants` *Remove instantaneous time spans from a phinterval*

Description

`phint_discard_instants()` removes instantaneous spans (spans with 0 duration) from phinterval elements. If all spans in an element are instantaneous, the result is a hole.

Usage

```
phint_discard_instants(phint)
```

Arguments

`phint`	`[phinterval / Interval]` A <phinterval> or <Interval> vector.

Value

A <phinterval> vector the same length as `phint`.

Examples

```
y2020 <- interval(as.Date("2020-01-01"), as.Date("2021-01-01"))
y2021 <- interval(as.Date("2021-01-01"), as.Date("2022-01-01"))
y2022 <- interval(as.Date("2022-01-01"), as.Date("2023-01-01"))

# The intersection of two adjacent intervals is instantaneous
new_years_2021 <- phint_intersect(y2020, y2021)
new_years_2021
phint_discard_instants(new_years_2021)

# phint_discard_instants() removes instants while keeping non-instantaneous spans
y2022_and_new_years <- phint_union(y2022, new_years_2021)
y2022_and_new_years
phint_discard_instants(y2022_and_new_years)
```

phint_invert *Get the gaps in a phinterval as time spans*

Description

`phint_invert()` returns the gaps within a phinterval as a <phinterval> vector. For phintervals with multiple disjoint spans, the gaps between those spans are returned. Contiguous time spans (e.g., `lubridate::interval()` vectors) have no gaps and are inverted to holes.

`phint_invert()` is similar to `phint_complement()`, except that time occurring outside the extent of `phint` (before its earliest start or after its latest end) is not included in the result.

Usage

```
phint_invert(phint, hole_to = c("hole", "inf", "na"))
```

Arguments

phint	[phinterval / Interval] A <phinterval> or <Interval> vector.
hole_to	["hole" / "inf" / "na"] How to handle holes (empty phinterval elements): • "hole" (default): Holes remain as holes • "inf": Return a span from -Inf to Inf (all time) • "na": Return an NA phinterval

Value

A <phinterval> vector the same length as `phint`.

Examples

```
monday <- interval(as.Date("2025-11-10"), as.Date("2025-11-11"))
friday <- interval(as.Date("2025-11-14"), as.Date("2025-11-15"))
sunday <- interval(as.Date("2025-11-16"), as.Date("2025-11-17"))

# Contiguous intervals have no gaps (inverted to holes)
phint_invert(monday)

# Disjoint intervals: gaps between spans are returned
phint_invert(phint_squash(c(monday, friday, sunday)))

# The gap between Monday and Friday is Tuesday through Thursday
tues_to_thurs <- interval(as.Date("2025-11-11"), as.Date("2025-11-14"))
phint_invert(phint_union(monday, friday)) == tues_to_thurs

# Invert vs complement: time before and after is excluded from invert
mon_and_fri <- phint_union(monday, friday)
```

```
phint_invert(mon_and_fri)
phint_complement(mon_and_fri)

# How to invert holes
hole <- phint_intersect(monday, friday)
phint_invert(hole, hole_to = "hole")
phint_invert(hole, hole_to = "inf")
phint_invert(hole, hole_to = "na")
```

`phint_length` *Compute the length of a phinterval in seconds*

Description

`phint_length()` calculates the total length of all time spans within each phinterval element in seconds. For phintervals with multiple disjoint spans, the lengths are summed. Instantaneous intervals and holes have length 0.

`phint_lengths()` returns the individual length in seconds of each time span within each phinterval element.

Usage

```
phint_length(phint)

## Default S3 method:
phint_length(phint)

## S3 method for class 'Interval'
phint_length(phint)

## S3 method for class 'phinterval'
phint_length(phint)

phint_lengths(phint)

## Default S3 method:
phint_lengths(phint)

## S3 method for class 'Interval'
phint_lengths(phint)

## S3 method for class 'phinterval'
phint_lengths(phint)
```

Arguments

`phint`	`[phinterval / Interval]` A <phinterval> or <Interval> vector.

Value

For `phint_length()`, a numeric vector the same length as `phint`.

For `phint_lengths()`, a list of numeric vectors the same length as `phint`.

Examples

```
monday <- interval(as.Date("2025-11-10"), as.Date("2025-11-11"))
friday <- interval(as.Date("2025-11-14"), as.Date("2025-11-15"))

phint_length(monday)
phint_length(phint_intersect(monday, friday))

# phint_length() sums the lengths of disjoint time spans
mon_and_fri <- phint_union(monday, friday)
phint_length(mon_and_fri) == phint_length(monday) + phint_length(friday)

# phint_lengths() returns the length of each disjoint time span
phint_lengths(mon_and_fri)
```

`phint_overlaps` *Test whether two phintervals overlap*

Description

`phint_overlaps()` tests whether the i-th element of `phint1` overlaps with the i-th element of `phint2`, returning a logical vector. Adjacent intervals (where one ends exactly when the other begins) are considered overlapping. `phint1` and `phint2` are recycled to their common length using vctrs-style recycling rules.

Usage

```
phint_overlaps(phint1, phint2, bounds = c("[]", "()"))
```

Arguments

`phint1, phint2`	`[phinterval / Interval]` A pair of `<phinterval>` or `<Interval>` vectors. `phint1` and `phint2` are recycled to a common length using vctrs-style recycling.
`bounds`	`["[]" / "()"]` Whether span endpoints are inclusive or exclusive: • `"[]"` (default): Closed intervals - both endpoints are included • `"()"`: Open intervals - both endpoints are excluded This affects adjacency and overlap detection. For example, with `bounds = "[]"`, the intervals `[1, 5]` and `[5, 10]` are considered adjacent (they share the endpoint 5), while with `bounds = "()"`, `(1, 5)` and `(5, 10)` are disjoint (neither includes 5).

Value

A logical vector.

Examples

```
monday <- interval(as.Date("2025-11-10"), as.Date("2025-11-11"))
tuesday <- interval(as.Date("2025-11-11"), as.Date("2025-11-12"))
friday <- interval(as.Date("2025-11-14"), as.Date("2025-11-15"))
mon_and_fri <- phint_union(monday, friday)

phint_overlaps(c(monday, monday, friday), c(mon_and_fri, friday, NA))

# Adjacent intervals are considered overlapping by default
phint_overlaps(monday, tuesday)

# Use exclusive bounds to consider adjacent intervals as disjoint
phint_overlaps(monday, tuesday, bounds = "()")

# Holes never overlap with anything (including other holes)
hole <- hole()
phint_overlaps(c(hole, monday), c(hole, hole))
```

---

phint_sift *Keep or discard spans by duration*

Description

`phint_sift()` keeps or removes spans within each element of `phint` based on their duration. At least one of `min_length` or `max_length` must be non-`NULL`. Duration bounds are inclusive.

When `action = "keep"` (the default):

- `min_length` and `max_length`: keep spans where `min_length <= length <= max_length`.
- `min_length` only: keep spans where `length >= min_length`.
- `max_length` only: keep spans where `length <= max_length`.

When `action = "discard"`, these conditions are negated; spans satisfying the condition are removed rather than kept. The function `phint_discard_instants()` is a wrapper around `phint_sift(phint, max_length = 0L, action = "discard")` for the common case of removing spans which are 0-seconds in length (i.e. `start == end`).

Span durations are computed in seconds, equivalent to `phint_lengths()`.

Usage

```
phint_sift(
  phint,
  min_length = NULL,
  max_length = NULL,
  action = c("keep", "discard")
)
```

Arguments

phint	[phinterval / Interval] A <phinterval> or <Interval> vector.
min_length	[Duration / numeric / NULL] The minimum span duration (inclusive). Must be recyclable with phint. If NULL (the default), no lower bound is applied.
max_length	[Duration / numeric / NULL] The maximum span duration (inclusive). Must be recyclable with phint. If NULL (the default), no upper bound is applied.
action	["keep" / "discard"] Whether to keep or discard spans satisfying the duration condition: • "keep" (default): Retain only spans satisfying the condition. • "discard": Remove spans satisfying the condition.

Value

A <phinterval> vector the same length as phint. Elements where all spans are removed become hole()s.

See Also

- phint_discard_instants() to remove zero-duration spans, equivalent to phint_sift(phint, max_length = 0, action = "discard").
- phint_lengths() to compute the duration of each span in seconds.

Examples

```
one_day <- interval(as.Date("2025-10-10"), as.Date("2025-10-11"))
two_days <- interval(as.Date("2025-11-12"), as.Date("2025-11-14"))
three_days <- interval(as.Date("2025-12-14"), as.Date("2025-12-17"))

# Keep spans which are at most 2 days long
phint_sift(
  phint_squash(c(one_day, two_days, three_days)),
  min_length = duration(2, "days")
)

# Keep spans which are at least 2 days long
phint_sift(
```

```
  phint_squash(c(one_day, two_days, three_days)),
  max_length = duration(2, "days")
)

# Keep spans with duration in [1.5 days, 2 days]
phint_sift(
  phint_squash(c(one_day, two_days, three_days)),
  min_length = duration(1.5, "days"),
  max_length = duration(2, "days")
)

# Discard spans with duration in [1.5 days, 2 days]
phint_sift(
  phint_squash(c(one_day, two_days, three_days)),
  min_length = duration(1.5, "days"),
  max_length = duration(2, "days"),
  action = "keep"
)

# All spans removed results in a hole
phint_sift(one_day, max_length = duration(0, "days"))

# Spans within a disjoint element are sifted independently
phint_sift(
  phint_squash(c(two_days, three_days)),
  max_length = duration(2, "days")
)

# min_length and max_length are vectorized
phint_sift(
  c(one_day, two_days),
  min_length = duration(c(0, 3), "days")
)
```

`phint_unnest` *Unnest a phinterval into a data frame*

Description

`phint_unnest()` converts a <phinterval> vector into a `tibble::tibble()` where each time span becomes a row.

Usage

```
phint_unnest(phint, key = NULL, hole_to = c("na", "drop"))
```

Arguments

phint	[phinterval / Interval]
	A <phinterval> or <Interval> vector to unnest.
key	[vector / data.frame / NULL]
	An optional vector or data frame to use as the key column in the output. If provided, must be the same length as phint. If NULL (the default), the key column contains row indices (position in phint).
	key may be any vector in the vctrs sense. See vctrs::obj_is_vector() for details.
hole_to	["na" / "drop"]
	How to handle hole elements (phintervals with zero spans). If "na" (the default), a row with NA start and end times and a size of 0 is included for each hole. If "drop", holes are excluded from the output.

Details

phint_unnest() expands each phinterval element into its constituent time spans, creating one row per span. The resulting data frame contains a key column identifying which phinterval element each span came from, a start and end column for each span's boundaries, and a size column counting the number of spans in the phinterval element.

For phinterval elements containing multiple disjoint spans, all spans are included with the same key value and size. Scalar phinterval elements (single spans) produce a single row. Both NA elements and hole()s produce NA values in the start and end columns, but have a size of NA and 0 respectively.

Value

A tibble::tibble() with columns:

- key:
  - If key = NULL: A numeric vector identifying the index of the phinterval element.
  - Otherwise: The element of key corresponding to the phinterval element.
- start: POSIXct start time of the span.
- end: POSIXct end time of the span.
- size: Integer count of spans in the phinterval element.

Examples

```
# Unnest scalar phintervals
phint <- phinterval(
  start = as.Date(c("2000-01-01", "2000-02-01")),
  end = as.Date(c("2000-01-15", "2000-02-15"))
)
phint_unnest(phint)

# Unnest multi-span phinterval
phint <- phinterval(
```

```
  start = as.Date(c("2000-01-01", "2000-03-01")),
  end = as.Date(c("2000-01-15", "2000-03-15")),
  by = 1
)
phint_unnest(phint)

# Handle holes
phint <- c(
  phinterval(as.Date("2000-01-01"), as.Date("2000-01-15")),
  hole(),
  phinterval(as.Date("2000-02-01"), as.Date("2000-02-15"))
)
phint_unnest(phint, hole_to = "na")
phint_unnest(phint, hole_to = "drop")

# Use a custom `key`
phint_unnest(phint, key = c("A", "B", "C"), hole_to = "na")
```

`phint_unoverlap` *Resolve overlapping intervals sequentially or by priority*

Description

`phint_unoverlap()` removes overlaps across elements of a `<phinterval>` vector by trimming each element against all preceding elements. The result is a vector where no two elements share any time.

Without `priority`, each element is trimmed by the union of all previous elements:

```
result[i] = phint_setdiff(phint[i], phint_squash(phint[1:(i - 1)]))
```

With `priority`, elements are grouped and processed in `priority_order`. Each element is trimmed by all elements from earlier priority groups. Within a priority group, `within_priority` controls whether overlapping elements within each group are kept as-is or trimmed.

Usage

```
phint_unoverlap(
  phint,
  priority = NULL,
  priority_order = c("asc", "desc", "appearance"),
  within_priority = c("sequential", "keep"),
  na_propagate = FALSE
)
```

Arguments

phint	[phinterval / Interval] A <phinterval> or <Interval> vector.
priority	[vector / NULL] An optional grouping vector defining priority groups. Earlier groups (per priority_order) are processed first and block later groups. Must be recyclable with phint. If NULL (the default), all elements are resolved considered to be within the same group. priority may be any vector in the vctrs sense. See vctrs::obj_is_vector() for details.
priority_order	["asc" / "desc" / "appearance"] How to order priority groups for processing: • "asc" (default): Lower values are processed first (priority 1 before 2). • "desc": Higher values are processed first (priority 9 before 2). • "appearance": Groups are processed in order of first appearance in priority.
within_priority	["sequential" / "keep"] How to handle overlaps within the same priority group: • "sequential" (default): Overlaps within a group are resolved by row order, so earlier elements block later elements within the same group. • "keep": Overlaps within a group are preserved; only overlaps with higher-priority groups are removed.
na_propagate	[FALSE / TRUE] Whether NA elements propagate to subsequent elements: • FALSE (default): NA elements are left as-is and do not affect subsequent results. • TRUE: An NA element causes all subsequent elements (or lower-priority group elements) to become NA.

Value

A <phinterval> vector the same length as phint, where no two elements overlap.

See Also

- phint_has_overlaps() to test whether a <phinterval> vector has cross-element overlaps.

Examples

```
monday <- interval(as.Date("2025-11-10"), as.Date("2025-11-11"))
tuesday <- interval(as.Date("2025-11-11"), as.Date("2025-11-12"))
wednesday <- interval(as.Date("2025-11-12"), as.Date("2025-11-13"))
mon_to_wed <- interval(as.Date("2025-11-10"), as.Date("2025-11-13"))
mon_to_tue <- interval(as.Date("2025-11-10"), as.Date("2025-11-12"))

# Sequential removal: each element is trimmed by all previous elements
```

```
phint_unoverlap(c(wednesday, mon_to_wed, mon_to_tue))

# Priority-based: lower priority values are processed first
phint_unoverlap(
  c(mon_to_wed, mon_to_tue, wednesday),
  priority = c(1, 2, 1)
)

# within_priority = "keep": overlaps within a group are preserved
phint_unoverlap(
  c(mon_to_wed, mon_to_tue, wednesday),
  priority = c(1, 1, 2),
  within_priority = "keep"
)

# priority_order = "desc": higher priority values are processed first
phint_unoverlap(
  c(mon_to_wed, mon_to_tue, wednesday),
  priority = c(1, 2, 1),
  priority_order = "desc"
)

# NA elements are treated as ignored by default
phint_unoverlap(c(mon_to_wed, NA, wednesday))

# NA elements propagate forward with na_propagate = TRUE
phint_unoverlap(c(mon_to_wed, NA, wednesday), na_propagate = TRUE)
```

`phint_within` *Test whether a datetime or phinterval is within another phinterval*

Description

`phint_within()` tests whether the i-th element of `x` is contained within the i-th element of `phint`, returning a logical vector. `x` may be a datetime (Date, POSIXct, POSIXlt), `lubridate::interval()`, or `phinterval()`, while `phint` must be a `lubridate::interval()` or `phinterval()`. `x` and `phint` are recycled to their common length using vctrs-style recycling rules.

Datetimes on an endpoint of an interval are considered to be within the interval. An interval is considered to be within itself.

Usage

```
phint_within(x, phint, bounds = c("[]", "()"))
```

Arguments

`x`	`[phinterval / Interval / Date / POSIXct / POSIXlt]` The object to test.

phint [phinterval / Interval]

A <phinterval> or <Interval> vector.

bounds ["[]" / "()"]

Whether span endpoints are inclusive or exclusive:

- "[]" (default): Closed intervals - both endpoints are included
- "()": Open intervals - both endpoints are excluded

This affects adjacency and overlap detection. For example, with bounds = "[]", the intervals [1, 5] and [5, 10] are considered adjacent (they share the endpoint 5), while with bounds = "()", (1, 5) and (5, 10) are disjoint (neither includes 5).

Value

A logical vector.

Examples

```
jan_1_to_5 <- interval(as.Date("2000-01-01"), as.Date("2000-01-05"))
jan_2_to_4 <- interval(as.Date("2000-01-02"), as.Date("2000-01-04"))
jan_3_to_9 <- interval(as.Date("2000-01-03"), as.Date("2000-01-09"))

phint_within(
  c(jan_2_to_4, jan_3_to_9, jan_1_to_5),
  c(jan_1_to_5, jan_1_to_5, NA)
)

phint_within(as.Date(c("2000-01-06", "2000-01-20")), jan_3_to_9)

# Intervals are within themselves
phint_within(jan_1_to_5, jan_1_to_5)

# By default, interval endpoints are considered within
phint_within(as.Date("2000-01-01"), jan_1_to_5)

# Use bounds to consider intervals as exclusive of endpoints
phint_within(as.Date("2000-01-01"), jan_1_to_5, bounds = "()")

# Holes are never within any interval (including other holes)
hole <- hole()
phint_within(c(hole, hole), c(hole, jan_1_to_5))
```

---

phinterval *Create a new phinterval*

Description

phinterval() creates a new <phinterval> vector from start and end times. A phinterval (think "potentially holey interval") is a span of time which may contain gaps.

Usage

```
phinterval(
  start = POSIXct(),
  end = POSIXct(),
  tzone = NULL,
  by = NULL,
  order_by = FALSE
)
```

Arguments

start, end	[POSIXct / POSIXlt / Date] A pair of datetime vectors to represent the endpoints of the spans. start and end are recycled to a common length using vctrs-style recycling rules.
tzone	[character(1)] A time zone to display the <phinterval> in. If tzone is NULL (the default), then the time zone is taken from that of start. tzone can be any non-NA string, but unrecognized time zones (see is_recognized_tzone()) will be formatted using "UTC" with a warning.
by	[vector / data.frame / NULL] An optional grouping vector or data frame. When provided, start[i] and end[i] pairs are grouped by by[i], creating one phinterval element per unique value of by. Overlapping or abutting spans within each group are merged. If NULL (the default), each start/end pair creates a separate phinterval element. by is recycled to match the common length of start and end. by may be any vector in the vctrs sense. See vctrs::obj_is_vector() for details.
order_by	[TRUE / FALSE] Should the output be ordered by the values in by? If FALSE (the default), the output order matches the first appearance of each group in by. If TRUE, the output is sorted by the unique values of by. Only used when by is not NULL.

Details

The <phinterval> class is designed as a generalization of the lubridate::interval(). While an <Interval> element represents a single contiguous span between two fixed times, a <phinterval> element can represent a time span that may be empty, contiguous, or disjoint (i.e. containing gaps). Each element of a <phinterval> is stored as a (possibly empty) set of non-overlapping and non-abutting time spans.

When by = NULL (the default), phinterval() creates scalar phinterval elements, where each element contains a single time span from start[i] to end[i]. This is equivalent to lubridate::interval():

```
interval(start, end, tzone = tzone)   # <Interval> vector
phinterval(start, end, tzone = tzone) # <phinterval> vector
```

When by is provided, phinterval() groups the start/end pairs by the values in by, creating phinterval elements that may contain multiple disjoint time spans. Overlapping or abutting spans within each group are automatically merged.

Value

When by = NULL, a <phinterval> vector the same length as the recycled length of start and end.

When by is provided, a <phinterval> vector with one element per unique value of by.

Differences from interval()

While phinterval() is designed as a drop-in replacement for lubridate::interval(), there are three key differences regarding the start and end arguments:

- **Stricter recycling**: phinterval() uses vctrs recycling rules instead of base R recycling. Length-1 vectors recycle to any length, but mismatched lengths (e.g., 2 vs 3) cause an error.
- **No character inputs**: phinterval() does not accept character vectors for start and end. Character starts and ends (e.g. "2021-01-01") must be converted to datetimes first using as.POSIXct(), lubridate::ymd(), or a similar function.
- **Standardized endpoints**: lubridate::interval() allows negative length spans where end[i] < start[i]. phinterval() flips the order of the i-th span's endpoints when end[i] < start[i] to ensure that all spans are positive, similar to lubridate::int_standardize().

Examples

```
# Scalar phintervals (equivalent to interval())
phinterval(
  start = as.Date(c("2000-01-01", "2000-02-01")),
  end = as.Date(c("2000-02-01", "2000-03-01"))
)

# Grouped phintervals with multiple spans per element
phinterval(
  start = as.Date(c("2000-01-01", "2000-03-01", "2000-02-01")),
  end = as.Date(c("2000-02-01", "2000-04-01", "2000-03-01")),
  by = c(1, 1, 2)
)

# Overlapping spans are merged within groups
phinterval(
  start = as.Date(c("2000-01-01", "2000-01-15")),
  end = as.Date(c("2000-02-01", "2000-02-15")),
  by = 1
)

# Empty phinterval
phinterval()

# Specify time zone
phinterval(
  start = as.Date("2000-01-01"),
  end = as.Date("2000-02-01"),
  tzone = "America/New_York"
)
```

phinterval-accessors *Accessors for the endpoints of a phinterval*

Description

phint_start() and phint_end() return the earliest and latest endpoint of each phinterval element, respectively. Holes (empty time spans) are returned as NA.

phint_starts() and phint_ends() return lists of all start and end points for each phinterval element, respectively. For phintervals with multiple disjoint spans, each span's endpoint is included. Holes are returned as length-0 <POSIXct> vectors.

Usage

```
phint_start(phint)

## Default S3 method:
phint_start(phint)

## S3 method for class 'Interval'
phint_start(phint)

## S3 method for class 'phinterval'
phint_start(phint)

phint_end(phint)

## Default S3 method:
phint_end(phint)

## S3 method for class 'Interval'
phint_end(phint)

## S3 method for class 'phinterval'
phint_end(phint)

phint_starts(phint)

## Default S3 method:
phint_starts(phint)

## S3 method for class 'Interval'
phint_starts(phint)

## S3 method for class 'phinterval'
phint_starts(phint)
```

```
phint_ends(phint)

## Default S3 method:
phint_ends(phint)

## S3 method for class 'Interval'
phint_ends(phint)

## S3 method for class 'phinterval'
phint_ends(phint)
```

Arguments

phint	[phinterval / Interval] A <phinterval> or <Interval> vector.

Value

For phint_start() and phint_end(), a <POSIXct> vector the same length as phint.

For phint_starts() and phint_ends(), a list of <POSIXct> vectors the same length as phint.

Examples

```
int1 <- interval(as.Date("2020-01-10"), as.Date("2020-02-01"))
int2 <- interval(as.Date("2023-05-02"), as.Date("2023-06-03"))

phint_start(int1)
phint_end(int1)

# Holes have no endpoints; disjoint phintervals have multiple endpoints
hole <- phint_intersect(int1, int2)
disjoint <- phint_union(int1, int2)

phint_start(c(hole, disjoint))
phint_starts(c(hole, disjoint))

phint_end(c(hole, disjoint))
phint_ends(c(hole, disjoint))

# phint_start() and phint_end() return the minimum and maximum endpoints
negative <- interval(as.Date("1980-01-01"), as.Date("1979-12-27"))
phint_start(negative)
phint_end(negative)
```

---

phinterval-cumset-operations

Cumulative set operations

Description

These functions perform cumulative elementwise set operations on <phinterval> vectors, treating each element as a set of non-overlapping intervals. They return a new <phinterval> vector where each element is the result of applying the corresponding set operation across all preceding elements.

- phint_cumunion() returns the running union of all elements up to and including phint[i].
- phint_cumintersect() returns the running intersection of all elements up to and including phint[i].

Usage

```
phint_cumunion(phint, na_propagate = FALSE)

phint_cumintersect(phint, na_propagate = FALSE, bounds = c("[]", "()"))
```

Arguments

phint	[phinterval / Interval] A <phinterval> or <Interval> vector.
na_propagate	[FALSE / TRUE] Whether NA values propagate forward through the cumulative result: • FALSE (default): NA elements are converted to hole()s and do not affect subsequent results. • TRUE: An NA element causes all subsequent elements to become NA.
bounds	["[]" / "()"] For phint_cumintersect(), whether span endpoints are inclusive or exclusive: • "[]" (default): Closed intervals - both endpoints are included. • "()": Open intervals - both endpoints are excluded. This affects adjacency and overlap detection. For example, with bounds = "[]", the intervals [1, 5] and [5, 10] are considered adjacent (they share the endpoint 5), while with bounds = "()", (1, 5) and (5, 10) are disjoint (neither includes 5).

Value

A <phinterval> vector the same length as phint.

See Also

phint_union() and phint_intersect() for the elementwise versions.

Examples

```
monday <- interval(as.Date("2025-11-10"), as.Date("2025-11-11"))
tuesday <- interval(as.Date("2025-11-11"), as.Date("2025-11-12"))
wednesday <- interval(as.Date("2025-11-12"), as.Date("2025-11-13"))
mon_to_wed <- interval(as.Date("2025-11-10"), as.Date("2025-11-13"))
```

```
# Cumulative union expands with each new element
phint_cumunion(c(monday, tuesday, wednesday))

# NA elements are treated as holes by default
phint_cumunion(c(monday, NA, wednesday))

# NA elements propagate forward with na_propagate = TRUE
phint_cumunion(c(monday, NA, wednesday), na_propagate = TRUE)

# Cumulative intersection narrows with each new element
phint_cumintersect(c(mon_to_wed, monday, tuesday))

# Once the intersection becomes a hole, it remains a hole
phint_cumintersect(c(monday, wednesday, mon_to_wed))

# Bounds affect the intersection of adjacent intervals
phint_cumintersect(c(monday, tuesday), bounds = "[]")
phint_cumintersect(c(monday, tuesday), bounds = "()")
```

`phinterval-overlap-predicates`

Test whether a phinterval vector has cross-element overlaps

Description

`phint_has_overlaps()` returns a logical vector indicating which elements of `phint` would be modified by `phint_unoverlap()`, i.e. elements which overlap with at least one preceding element (or lower-priority group element). Blockers, i.e. preceding elements which overlap with a following element, are not marked as overlapping. `phint_any_overlaps()` is a fast scalar equivalent to `any(phint_has_overlaps(...), na.rm = TRUE)`.

Both functions accept the same arguments as `phint_unoverlap()` and use the same priority and within-priority resolution rules.

The following invariants hold:

```
# phint_unoverlap() ensures that phint_has_overlaps() is FALSE
phint <- phint_unoverlap(phint, ...)
!phint_any_overlaps(phint, ...)

# phint_unoverlap() does not alter non-overlapping elements
overlapping <- phint_has_overlaps(phint, ...)
all(phint[!overlapping] == phint_unoverlap(phint, ...)[!overlapping], na.rm = TRUE)

# phint_any_overlaps() is equivalent to any(phint_has_overlaps(...))
phint_any_overlaps(phint, ...) == any(phint_has_overlaps(phint, ...), na.rm = TRUE)
```

Usage

```
phint_has_overlaps(
  phint,
  priority = NULL,
  priority_order = c("asc", "desc", "appearance"),
  within_priority = c("sequential", "keep"),
  na_propagate = FALSE
)

phint_any_overlaps(
  phint,
  priority = NULL,
  priority_order = c("asc", "desc", "appearance"),
  within_priority = c("sequential", "keep"),
  na_propagate = FALSE
)
```

Arguments

`phint` `[phinterval / Interval]`

A <phinterval> or <Interval> vector.

`priority` `[vector / NULL]`

An optional grouping vector defining priority groups. Earlier groups (per `priority_order`) are processed first and block later groups. Must be recyclable with `phint`. If `NULL` (the default), all elements are resolved considered to be within the same group.

`priority` may be any vector in the vctrs sense. See `vctrs::obj_is_vector()` for details.

`priority_order` `["asc" / "desc" / "appearance"]`

How to order priority groups for processing:

- `"asc"` (default): Lower values are processed first (priority 1 before 2).
- `"desc"`: Higher values are processed first (priority 9 before 2).
- `"appearance"`: Groups are processed in order of first appearance in `priority`.

`within_priority`

`["sequential" / "keep"]`

How to handle overlaps within the same priority group:

- `"sequential"` (default): Overlaps within a group are resolved by row order, so earlier elements block later elements within the same group.
- `"keep"`: Overlaps within a group are preserved; only overlaps with higher-priority groups are removed.

`na_propagate` `[FALSE / TRUE]`

Whether `NA` elements propagate to subsequent elements:

- `FALSE` (default): `NA` elements are left as-is and do not affect subsequent results.
- `TRUE`: An `NA` element causes all subsequent elements (or lower-priority group elements) to become `NA`.

Value

`phint_has_overlaps()` returns a logical vector the same length as `phint`:

- `TRUE`: the element overlaps with at least one preceding element and would be modified by `phint_unoverlap()`.
- `FALSE`: the element does not overlap with any preceding element and would not be affected by `phint_unoverlap()`. This includes `NA` elements when `na_propagate = FALSE`.
- `NA`: the element is itself `NA` or would become `NA` due to propagation. This is only applicable when `na_propagate = TRUE`.

`phint_any_overlaps()` returns a single `TRUE` or `FALSE`.

See Also

- `phint_unoverlap()` to resolve the overlaps detected by this function.

Examples

```
monday <- interval(as.Date("2025-11-10"), as.Date("2025-11-11"))
tuesday <- interval(as.Date("2025-11-11"), as.Date("2025-11-12"))
wednesday <- interval(as.Date("2025-11-12"), as.Date("2025-11-13"))
mon_to_wed <- interval(as.Date("2025-11-10"), as.Date("2025-11-13"))
mon_to_tue <- interval(as.Date("2025-11-10"), as.Date("2025-11-12"))

# No overlaps: all FALSE
phint_has_overlaps(c(monday, tuesday, wednesday))
phint_any_overlaps(c(monday, tuesday, wednesday))

# Only the blocked element is TRUE, not the blocker
phint_has_overlaps(c(mon_to_wed, mon_to_tue))
phint_any_overlaps(c(mon_to_wed, mon_to_tue))

# Non-overlapping elements are FALSE even when others overlap
phint_has_overlaps(c(mon_to_wed, mon_to_tue, wednesday))

# Priority-based: same rules as phint_unoverlap()
phint_has_overlaps(
  c(mon_to_wed, mon_to_tue, wednesday),
  priority = c(1, 2, 1)
)

# NA elements return NA
phint_has_overlaps(c(mon_to_wed, NA, wednesday))

# na_propagate = TRUE: NA propagates forward
phint_has_overlaps(c(monday, NA, wednesday), na_propagate = TRUE)
phint_any_overlaps(c(monday, NA, wednesday), na_propagate = TRUE)
```

phinterval-set-operations
Vectorized set operations

Description

These functions perform elementwise set operations on <phinterval> vectors, treating each element as a set of non-overlapping intervals. They return a new <phinterval> vector representing the result of the corresponding set operation. All functions follow vctrs-style recycling rules.

- phint_complement() returns all time spans *not covered* by phint.
- phint_union() returns the intervals that are within either phint1 or phint2.
- phint_intersect() returns the intervals that are within both phint1 and phint2.
- phint_setdiff() returns intervals in phint1 that are not within phint2.
- phint_symmetric_setdiff() returns intervals which are within either phint1 or phint2, but which are not within both phint1 and phint2.

phint_symmetric_setdiff(phint1, phint2) is equivalent to (but usually faster than) the following:

```
phint_setdiff(
 phint_union(phint1, phint2),
 phint_intersect(phint1, phint2)
)
```

Usage

```
phint_complement(phint)

phint_union(phint1, phint2)

phint_intersect(phint1, phint2, bounds = c("[]", "()"))

phint_setdiff(phint1, phint2)

phint_symmetric_setdiff(phint1, phint2)
```

Arguments

phint	[phinterval / Interval]
	A <phinterval> or <Interval> vector.
phint1, phint2	[phinterval / Interval]
	A pair of <phinterval> or <Interval> vectors. phint1 and phint2 are recycled to a common length using vctrs-style recycling.
bounds	["[]" / "()"]
	For phint_intersect(), whether span endpoints are inclusive or exclusive:

- `"[]"` (default): Closed intervals - both endpoints are included.
- `"()"`: Open intervals - both endpoints are excluded.

This affects adjacency and overlap detection. For example, with `bounds = "[]"`, the intervals `[1, 5]` and `[5, 10]` are considered adjacent (they share the endpoint 5), while with `bounds = "()"`, `(1, 5)` and `(5, 10)` are disjoint (neither includes 5).

Value

A `<phinterval>` vector.

Under Development

`phint_symmetric_setdiff()` is under development and may contain bugs. To use `phint_symmetric_setdiff()`, install the development version of phinterval from GitHub with `pak::pak("EthanSansom/phinterval")`.

Examples

```
monday <- interval(as.Date("2025-11-10"), as.Date("2025-11-11"))
tuesday <- interval(as.Date("2025-11-11"), as.Date("2025-11-12"))
friday <- interval(as.Date("2025-11-14"), as.Date("2025-11-15"))
jan_1_to_5 <- interval(as.Date("2000-01-01"), as.Date("2000-01-05"))
jan_3_to_9 <- interval(as.Date("2000-01-03"), as.Date("2000-01-09"))

# Complement
phint_complement(jan_1_to_5)

# The complement of a hole is an infinite span covering all time
hole <- hole()
phint_complement(hole)

# Union
phint_union(c(monday, monday, monday), c(tuesday, friday, NA))

# Elements of length 1 are recycled
phint_union(monday, c(tuesday, friday, NA))

# Intersection
phint_intersect(jan_1_to_5, jan_3_to_9)

# The intersection of non-overlapping intervals is a hole
phint_intersect(monday, friday)

# By default, the intersection of adjacent intervals is instantaneous
phint_intersect(monday, tuesday)

# Use bounds to set the intersection of adjacent intervals to a hole
phint_intersect(monday, tuesday, bounds = "()")

# Set difference
phint_setdiff(jan_1_to_5, jan_3_to_9)
phint_setdiff(jan_3_to_9, jan_1_to_5)
```

```
# Instantaneous intervals do not affect the set difference
noon_monday <- as.POSIXct("2025-11-10 12:00:00")
phint_setdiff(monday, interval(noon_monday, noon_monday)) == monday

# Symmetric difference
phint_symmetric_setdiff(jan_1_to_5, jan_3_to_9)

# The symmetric difference of non-overlapping intervals is their union
phint_symmetric_setdiff(monday, friday)

# The symmetric difference of identical intervals is a hole
phint_symmetric_setdiff(monday, monday)
```

`phinterval-span-predicates`

Test for empty, contiguous, or disjoint intervals

Description

These predicates test structural properties of each element of a <phinterval> vector.

- `is_hole()`: Is the element empty (zero spans)?
- `is_span()`: Is the element a single contiguous span?
- `is_disjoint()`: Is the element made up of two or more disjoint spans?

Usage

```
is_hole(phint)

is_span(phint)

is_disjoint(phint)
```

Arguments

`phint`	`[phinterval / Interval]` A <phinterval> or <Interval> vector.

Value

A logical vector the same length as `phint`.

Under Development

`is_span()` and `is_disjoint()` are under development and may contain bugs. To use these functions, install the development version of phinterval from GitHub with `pak::pak("EthanSansom/phinterval")`.

Examples

```
y2000 <- interval(as.Date("2000-01-01"), as.Date("2001-01-01"))
y2025 <- interval(as.Date("2025-01-01"), as.Date("2026-01-01"))
holey <- phint_union(y2000, y2025)

# Detect holes
is_hole(c(hole(), y2000, NA))

# The intersection of disjoint intervals is a hole
is_hole(phint_intersect(y2000, y2025))

# Detect single contiguous spans
is_span(c(y2000, holey, hole(), NA))

# Detect disjoint (multi-span) elements
is_disjoint(c(y2000, holey, hole(), NA))
```

`phinterval_options` *Package options*

Description

The `phinterval` package uses the following global options to control printing and default behaviors. These options can be set using `options()` and queried using `getOption()`.

Options

- `phinterval.print_max_width`: Character width at which a printed or formatted `<phinterval>` element is truncated for display, default: `90`.

Examples

```
monday <- phinterval(as.Date("2025-11-10"), as.Date("2025-11-11"))
friday <- phinterval(as.Date("2025-11-14"), as.Date("2025-11-15"))

# Get the default setting
getOption("phinterval.print_max_width")
phint_squash(c(monday, friday))

# Change the setting for the session duration
opts <- options(phinterval.print_max_width = 25)
phint_squash(c(monday, friday))

# Reset to the previous settings
options(opts)
```

squash *Squash overlapping intervals into non-overlapping spans*

Description

`phint_squash()` and `datetime_squash()` merge overlapping or adjacent intervals into a single `<phinterval>` element containing a minimal set of non-overlapping, non-adjacent time spans.

- `phint_squash()` takes a `<phinterval>` or `<Interval>` vector.
- `datetime_squash()` takes separate `start` and `end` datetime vectors.

`phint_squash_by()` and `datetime_squash_by()` merge intervals within groups defined by the `by` argument. The result is a `<phinterval>` vector containing one element per unique value of `by`.

Usage

```
phint_squash(phint, na_rm = TRUE, empty_to = c("hole", "na"))

datetime_squash(start, end, na_rm = TRUE, empty_to = c("hole", "na"))

phint_squash_by(
  phint,
  by,
  na_rm = TRUE,
  empty_to = c("hole", "na"),
  order_by = TRUE
)

datetime_squash_by(
  start,
  end,
  by,
  na_rm = TRUE,
  empty_to = c("hole", "na"),
  order_by = TRUE
)
```

Arguments

`phint`	`[phinterval / Interval]`
	A `<phinterval>` or `<Interval>` vector.
`na_rm`	`[TRUE / FALSE]`
	Should `NA` elements be removed before squashing? If `FALSE` and any `NA` elements are present, the result is `NA`. Defaults to `TRUE`.
`empty_to`	`["hole" / "na"]`
	How to handle empty inputs (length-0 vectors):

- "hole" (default): Return a hole.
- "na": Return an NA phinterval.

start [POSIXct / POSIXlt / Date]

A vector of start times. Must be recyclable with end. Only used in datetime_squash() and datetime_squash_by().

end [POSIXct / POSIXlt / Date]

A vector of end times. Must be recyclable with start. Only used in datetime_squash() and datetime_squash_by().

by [vector / data.frame]

A grouping vector or data frame. Intervals are grouped by by and merged separately within each group, returning one <phinterval> element per unique value of by.

For datetime_squash_by(), by must be recyclable with the recycled length of start and end.

by may be any vector in the vctrs sense. See vctrs::obj_is_vector() for details.

order_by [TRUE / FALSE]

Should the output be ordered by the values in by? If TRUE (the default), the output is sorted by the unique values of by. If FALSE, the output order matches the first appearance of each group in by. Only used in phint_squash_by() and datetime_squash_by().

Details

These functions are particularly useful in aggregation workflows with dplyr::summarize() to combine intervals within groups.

The phint_squash_by() and datetime_squash_by() variants are designed to replicate a call to dplyr::group_by() followed by dplyr::summarize(), but are typically faster. In particular, the following produce identical results:

```
phint_squash_by(phint, by = by)

dplyr::tibble(phint = phint, by = by) |>
  dplyr::group_by(by) |>
  dplyr::summarize(phint = phint_squash(phint)) |>
  dplyr::ungroup()
```

Value

phint_squash() and datetime_squash() return a length-1 <phinterval> vector.

phint_squash_by() and datetime_squash_by() return a tibble::tibble() with columns by and phint, with one row per unique value of by.

See Also

phint_flatten() and datetime_flatten() to merge a <phinterval> vector into a vector of scalar spans rather than a single element.

Examples

```
jan_1_to_5 <- interval(as.Date("2000-01-01"), as.Date("2000-01-05"))
jan_3_to_9 <- interval(as.Date("2000-01-03"), as.Date("2000-01-09"))
jan_11_to_12 <- interval(as.Date("2000-01-11"), as.Date("2000-01-12"))

# phint_squash: merge intervals from a phinterval/Interval vector
phint_squash(c(jan_1_to_5, jan_3_to_9, jan_11_to_12))

# datetime_squash: merge intervals from start/end vectors
datetime_squash(
  start = as.Date(c("2000-01-01", "2000-01-03", "2000-01-11")),
  end = as.Date(c("2000-01-05", "2000-01-09", "2000-01-12"))
)

# NA values are removed by default
phint_squash(c(jan_1_to_5, jan_3_to_9, jan_11_to_12, NA))

# Set na_rm = FALSE to propagate NA values
phint_squash(c(jan_1_to_5, jan_3_to_9, jan_11_to_12, NA), na_rm = FALSE)

# empty_to determines the result of empty inputs
phint_squash(phinterval(), empty_to = "hole")
phint_squash(phinterval(), empty_to = "na")

# phint_squash_by: squash within groups, returning a tibble
phint_squash_by(
  c(jan_1_to_5, jan_3_to_9, jan_11_to_12),
  by = c("A", "A", "B")
)

# datetime_squash_by: squash from start/end vectors within groups
datetime_squash_by(
  start = as.Date(c("2000-01-01", "2000-01-03", "2000-01-11")),
  end = as.Date(c("2000-01-05", "2000-01-09", "2000-01-12")),
  by = c("A", "A", "B")
)

# Control output order with order_by
phint_squash_by(
  c(jan_1_to_5, jan_3_to_9, jan_11_to_12),
  by = c(2, 2, 1),
  order_by = TRUE
)
```

Index

∗ **datasets**
    phinterval_options, 33

as.POSIXct(), *23*
as_duration, 2
as_phinterval, 3

datetime_flatten *(*flatten*)*, 4
datetime_flatten(), *35*
datetime_squash *(*squash*)*, 34
datetime_squash(), *5*
datetime_squash_by *(*squash*)*, 34
dplyr::group_by(), *35*
dplyr::summarize(), *35*

flatten, 4

getOption(), *33*

hole, 6
hole(), *15*, *17*, *26*

is_disjoint
        *(*phinterval-span-predicates*)*, 32
is_hole *(*phinterval-span-predicates*)*, 32
is_phinterval, 7
is_phintish, 8
is_recognized_tzone, 8
is_recognized_tzone(), *22*
is_span *(*phinterval-span-predicates*)*, 32

lubridate::as.duration(), *2*
lubridate::duration(), *2*
lubridate::int_standardize(), *4*, *23*
lubridate::interval(), *2*, *3*, *11*, *20*, *22*, *23*
lubridate::ymd(), *23*

n_spans, 9

options(), *33*

phint_any_overlaps
        *(*phinterval-overlap-predicates*)*, 27
phint_complement
        *(*phinterval-set-operations*)*, 30
phint_cumintersect
        *(*phinterval-cumset-operations*)*, 25
phint_cumunion
        *(*phinterval-cumset-operations*)*, 25
phint_discard_instants, 10
phint_discard_instants(), *14*, *15*
phint_end *(*phinterval-accessors*)*, 24
phint_ends *(*phinterval-accessors*)*, 24
phint_flatten *(*flatten*)*, 4
phint_flatten(), *35*
phint_has_overlaps
        *(*phinterval-overlap-predicates*)*, 27
phint_has_overlaps(), *19*
phint_intersect
        *(*phinterval-set-operations*)*, 30
phint_intersect(), *26*
phint_invert, 11
phint_length, 12
phint_lengths *(*phint_length*)*, 12
phint_lengths(), *14*, *15*
phint_overlaps, 13
phint_setdiff
        *(*phinterval-set-operations*)*, 30
phint_sift, 14
phint_squash *(*squash*)*, 34
phint_squash(), *5*
phint_squash_by *(*squash*)*, 34
phint_start *(*phinterval-accessors*)*, 24
phint_starts *(*phinterval-accessors*)*, 24
phint_symmetric_setdiff
        *(*phinterval-set-operations*)*, 30

phint_union
(phinterval-set-operations), 30
phint_union(), *26*
phint_unnest, 16
phint_unoverlap, 18
phint_unoverlap(), *27*, *29*
phint_within, 20
phinterval, 21
phinterval(), *2*, *4*, *20*
phinterval-accessors, 24
phinterval-cumset-operations, 25
phinterval-overlap-predicates, 27
phinterval-set-operations, 30
phinterval-span-predicates, 32
phinterval_options, 33

squash, 34

tibble::tibble(), *16*, *17*, *35*
tzdb::tzdb_names(), *8*

vctrs::obj_is_vector(), *17*, *19*, *22*, *28*, *35*