

# Package ‘fastRanges’

May 7, 2026

**Type** Package

**Title** Deterministic Multithreaded Genomic Interval Operations

**Version** 1.0.0

**Description** High-performance interval overlap and join operations for 'IRanges' and 'GenomicRanges'. The package provides deterministic multithreaded overlap computation, reusable subject indexes for repeated queries, and join helpers that keep range metadata in a consistent output grammar.

**URL** <https://github.com/cparsania/fastRanges>,  
<https://cparsania.github.io/fastRanges/>

**BugReports** <https://github.com/cparsania/fastRanges/issues>

**License** Artistic-2.0

**Encoding** UTF-8

**Depends** R (>= 4.5.0)

**Imports** methods, S4Vectors, IRanges, GenomicRanges, GenomeInfoDb, Rcpp

**LinkingTo** Rcpp

**Suggests** BiocStyle, testthat (>= 3.0.0), knitr, rmarkdown, pkgdown, XVector, ggplot2, dplyr, tidyr, scales

**VignetteBuilder** knitr

**SystemRequirements** quarto

**biocViews** Software, Infrastructure, Sequencing

**Config/testthat/edition** 3

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**git\_url** <https://git.bioconductor.org/packages/fastRanges>

**git\_branch** RELEASE\_3\_23

**git\_last\_commit** 36a6c05

**git\_last\_commit\_date** 2026-04-28

**Repository** Bioconductor 3.23

**Date/Publication** 2026-05-06

**Author** Chirag Parsania [aut, cre] (github: cparsania)

**Maintainer** Chirag Parsania <chirag.parsania@gmail.com>

## Contents

fast_anti_overlap_join . . . . .	3
fast_build_index . . . . .	5
fast_cluster_overlaps . . . . .	6
fast_count_overlaps . . . . .	9
fast_count_overlaps_by_group . . . . .	12
fast_coverage . . . . .	15
fast_default_threads . . . . .	16
fast_disjoin . . . . .	16
fast_distance_to_nearest . . . . .	17
fast_find_overlaps . . . . .	18
fast_find_overlaps_iter . . . . .	21
fast_follow . . . . .	24
fast_gaps . . . . .	25
fast_index_stats . . . . .	26
fast_inner_overlap_join . . . . .	27
fast_iter_collect . . . . .	29
fast_iter_has_next . . . . .	30
fast_iter_next . . . . .	31
fast_iter_reset . . . . .	31
fast_left_overlap_join . . . . .	32
fast_load_index . . . . .	35
fast_nearest . . . . .	35
fast_overlaps_any . . . . .	36
fast_overlap_aggregate . . . . .	39
fast_overlap_join . . . . .	42
fast_precede . . . . .	45
fast_ranges_example . . . . .	46
fast_range_intersect . . . . .	47
fast_range_setdiff . . . . .	47
fast_range_union . . . . .	48
fast_reduce . . . . .	49
fast_save_index . . . . .	49
fast_self_overlaps . . . . .	50
fast_semi_overlap_join . . . . .	53
fast_tile_coverage . . . . .	56
fast_window_count_overlaps . . . . .	57

**Index**

**60**

---

fast\_anti\_overlap\_join

*Anti Overlap Join*


---

### Description

Return query rows that do not overlap any subject ranges.

### Usage

```
fast_anti_overlap_join(
  query,
  subject,
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE,
  query_prefix = "query_"
)
```

### Arguments

query	An IRanges or GRanges query object.
subject	An IRanges/GRanges object or a fast_ranges_index. Use fast_build_index(subject) when the same subject is reused across many overlap queries.
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap. Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes. Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with IRanges::findOverlaps() / GenomicRanges::findOverlaps().
min_overlap	Integer scalar minimum overlap width, in bases. 0 is the least strict setting. Larger values require wider shared overlap and therefore return fewer hits. This argument matters only when the selected type allows an actual overlap width to be measured.
type	Character scalar describing what "match" means. "any" matches any overlap that satisfies max_gap / min_overlap. "start" matches ranges with compatible start coordinates.

	"end" matches ranges with compatible end coordinates.
	"within" matches queries contained inside subjects.
	"equal" matches queries and subjects with the same interval, or with start/end differences no larger than <code>max_gap</code> when tolerance is allowed.
<code>ignore_strand</code>	Logical scalar controlling strand handling for genomic ranges. For GRanges, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules. TRUE means strand is ignored and only genomic coordinates are compared. For IRanges, this argument has no effect because there is no strand.
<code>threads</code>	Integer scalar number of worker threads to use. Use 1 for the most conservative behavior and easiest debugging. Use larger values on multicore machines when throughput matters. For repeated-query workloads, combine a prebuilt index from <code>fast_build_index(subject)</code> with a thread count chosen empirically on your hardware. <code>fastRanges</code> is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.
<code>deterministic</code>	Logical scalar controlling output order. TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts. FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.
<code>query_prefix</code>	Prefix applied to query columns.

### Details

This is similar to a SQL ANTI JOIN.

It keeps only query rows with zero overlap hits.

`overlap_count` is always 0 in the returned table and is included to keep the result grammar parallel to `fast_semi_overlap_join()`.

### Value

A data.frame containing query rows with zero overlaps.

### Overlap semantics

`query` is the range set you ask about. `subject` is the range set you compare it against.

Core interval semantics (ASCII schematic):

```
type = "any"
query  :  |-----|
subject: |-----|
```

```
type = "within"
```

```

subject: |-----|
query  :   |-----|

type = "start"
query  :   |-----|
subject: |-----|

type = "end"
query  :       |-----|
subject: |-----|

type = "equal"
query  :   |-----|
subject:   |-----|

gap / min_overlap controls
query  : |-----|   |-----| : subject
          < gap >

```

The middle distance is the gap. A hit is allowed when this distance is  $\leq$  max\_gap (for max\_gap  $\geq$  0), and overlap width is  $\geq$  min\_overlap.

Beginner-friendly interpretation:

type = "any" asks "do these ranges touch or overlap closely enough to count?"

type = "start" and type = "end" are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

type = "within" asks whether each query lies inside a subject interval.

type = "equal" asks whether query and subject describe the same interval, optionally with endpoint tolerance when max\_gap  $\geq$  0.

This argument grammar is intentionally aligned with Bioconductor overlap APIs (IRanges / GenomicRanges).

## Examples

```

q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
fast_anti_overlap_join(q, s)

```

---

fast\_build\_index      *Build a Reusable Subject Index*

---

## Description

Build a sorted subject index that can be reused across repeated overlap queries.

## Usage

```
fast_build_index(subject)
```

**Arguments**

subject            An IRanges or GRanges object.

**Details**

The index stores a sorted representation of subject that is optimized for repeated overlap queries.

Build the index once, then pass it as subject to fast\_find\_overlaps(), fast\_count\_overlaps(), or other overlap-summary functions.

Use the raw subject object, not the index, when you need subject metadata in the output table, for example with overlap joins.

**Value**

A fast\_ranges\_index object.

**Examples**

```
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
idx <- fast_build_index(s)
idx
```

---

fast\_cluster\_overlaps    *Cluster Overlapping Ranges*

---

**Description**

Assign each range to an overlap-connected component.

**Usage**

```
fast_cluster_overlaps(  
  x,  
  max_gap = -1L,  
  min_overlap = 0L,  
  type = c("any", "start", "end", "within", "equal"),  
  ignore_strand = FALSE,  
  threads = fast_default_threads(),  
  deterministic = TRUE,  
  return = c("vector", "data.frame")  
)
```

**Arguments**

x	An IRanges or GRanges object.
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap. Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes. Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with IRanges::findOverlaps() / GenomicRanges::findOverlaps().
min_overlap	Integer scalar minimum overlap width, in bases. 0 is the least strict setting. Larger values require wider shared overlap and therefore return fewer hits. This argument matters only when the selected type allows an actual overlap width to be measured.
type	Character scalar describing what "match" means. "any" matches any overlap that satisfies max_gap / min_overlap. "start" matches ranges with compatible start coordinates. "end" matches ranges with compatible end coordinates. "within" matches queries contained inside subjects. "equal" matches queries and subjects with the same interval, or with start/end differences no larger than max_gap when tolerance is allowed.
ignore_strand	Logical scalar controlling strand handling for genomic ranges. For GRanges, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules. TRUE means strand is ignored and only genomic coordinates are compared. For IRanges, this argument has no effect because there is no strand.
threads	Integer scalar number of worker threads to use. Use 1 for the most conservative behavior and easiest debugging. Use larger values on multicore machines when throughput matters. For repeated-query workloads, combine a prebuilt index from fast_build_index(subject) with a thread count chosen empirically on your hardware. fastRanges is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.
deterministic	Logical scalar controlling output order. TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts. FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.
return	One of "vector" or "data.frame".

## Details

Two ranges are assigned to the same cluster when they are connected by a chain of overlaps under the provided overlap settings.

Two ranges can end up in the same cluster even if they do not overlap directly, as long as they are linked by intermediate overlaps.

Example: if range 1 overlaps range 2, and range 2 overlaps range 3, then all three belong to one cluster.

## Value

If return = "vector", an integer vector of cluster IDs with one element per range in x. If return = "data.frame", a data.frame with range\_id, cluster\_id, and cluster\_size.

## Overlap semantics

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```
type = "any"
query  :  |-----|
subject: |-----|
```

```
type = "within"
subject: |-----|
query  :  |-----|
```

```
type = "start"
query  :  |-----|
subject: |-----|
```

```
type = "end"
query  :          |-----|
subject: |-----|
```

```
type = "equal"
query  :  |-----|
subject:  |-----|
```

```
gap / min_overlap controls
query  : |-----|      |-----| : subject
          < gap >
```

The middle distance is the gap. A hit is allowed when this distance is  $\leq$  max\_gap (for max\_gap  $\geq$  0), and overlap width is  $\geq$  min\_overlap.

Beginner-friendly interpretation:

type = "any" asks "do these ranges touch or overlap closely enough to count?"

type = "start" and type = "end" are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

type = "within" asks whether each query lies inside a subject interval.

type = "equal" asks whether query and subject describe the same interval, optionally with endpoint tolerance when max\_gap >= 0.

This argument grammar is intentionally aligned with Bioconductor overlap APIs (IRanges / GenomicRanges).

## Examples

```
x <- IRanges::IRanges(start = c(1, 3, 10, 11, 20), end = c(5, 8, 12, 14, 22))
fast_cluster_overlaps(x)
```

---

fast\_count\_overlaps     *Count Overlaps*

---

## Description

Count subject overlaps per query range.

## Usage

```
fast_count_overlaps(
  query,
  subject,
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE
)
```

## Arguments

query	An IRanges or GRanges query object.
subject	An IRanges/GRanges object or a fast_ranges_index. Use fast_build_index(subject) when the same subject is reused across many overlap queries.
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap. Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes. Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with IRanges::findOverlaps() / GenomicRanges::findOverlaps().

min_overlap	<p>Integer scalar minimum overlap width, in bases.  <math>\emptyset</math> is the least strict setting.  Larger values require wider shared overlap and therefore return fewer hits.  This argument matters only when the selected type allows an actual overlap width to be measured.</p>
type	<p>Character scalar describing what "match" means.  "any" matches any overlap that satisfies max_gap / min_overlap.  "start" matches ranges with compatible start coordinates.  "end" matches ranges with compatible end coordinates.  "within" matches queries contained inside subjects.  "equal" matches queries and subjects with the same interval, or with start/end differences no larger than max_gap when tolerance is allowed.</p>
ignore_strand	<p>Logical scalar controlling strand handling for genomic ranges.  For GRanges, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules.  TRUE means strand is ignored and only genomic coordinates are compared.  For IRanges, this argument has no effect because there is no strand.</p>
threads	<p>Integer scalar number of worker threads to use.  Use 1 for the most conservative behavior and easiest debugging.  Use larger values on multicore machines when throughput matters.  For repeated-query workloads, combine a prebuilt index from fast_build_index(subject) with a thread count chosen empirically on your hardware.  fastRanges is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.</p>
deterministic	<p>Logical scalar controlling output order.  TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts.  FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.</p>

### Details

deterministic does not change returned counts for this summary output.

Returns one integer per query range.

A value of  $\emptyset$  means that query had no matching subjects.

A value of 5 means that query matched five subject ranges under the chosen overlap rule.

### Value

Integer vector with one element per query range.

## Overlap semantics

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```

type = "any"
query  :  |-----|
subject: |-----|

type = "within"
subject: |-----|
query   :  |-----|

type = "start"
query   :  |-----|
subject: |-----|

type = "end"
query   :          |-----|
subject: |-----|

type = "equal"
query   :  |-----|
subject:  |-----|

gap / min_overlap controls
query  : |-----|      |-----| : subject
          < gap >

```

The middle distance is the gap. A hit is allowed when this distance is  $\leq \text{max\_gap}$  (for  $\text{max\_gap} \geq 0$ ), and overlap width is  $\geq \text{min\_overlap}$ .

Beginner-friendly interpretation:

type = "any" asks "do these ranges touch or overlap closely enough to count?"

type = "start" and type = "end" are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

type = "within" asks whether each query lies inside a subject interval.

type = "equal" asks whether query and subject describe the same interval, optionally with endpoint tolerance when  $\text{max\_gap} \geq 0$ .

This argument grammar is intentionally aligned with Bioconductor overlap APIs (IRanges / GenomicRanges).

## Examples

```

q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
fast_count_overlaps(q, s, threads = 1)

```

---

fast\_count\_overlaps\_by\_group

*Count Overlaps by Subject Group*


---

## Description

Count overlaps per query and per subject metadata group.

## Usage

```
fast_count_overlaps_by_group(
  query,
  subject,
  group_col,
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE,
  include_na_group = FALSE
)
```

## Arguments

query	An IRanges or GRanges query object.
subject	An IRanges/GRanges object or a fast_ranges_index. Use fast_build_index(subject) when the same subject is reused across many overlap queries.
group_col	Subject metadata column name used for grouping. This must be present in mcols(subject).
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap. Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes. Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with IRanges::findOverlaps() / GenomicRanges::findOverlaps().
min_overlap	Integer scalar minimum overlap width, in bases. 0 is the least strict setting. Larger values require wider shared overlap and therefore return fewer hits. This argument matters only when the selected type allows an actual overlap width to be measured.

type	<p>Character scalar describing what "match" means.</p> <p>"any" matches any overlap that satisfies max_gap / min_overlap.</p> <p>"start" matches ranges with compatible start coordinates.</p> <p>"end" matches ranges with compatible end coordinates.</p> <p>"within" matches queries contained inside subjects.</p> <p>"equal" matches queries and subjects with the same interval, or with start/end differences no larger than max_gap when tolerance is allowed.</p>
ignore_strand	<p>Logical scalar controlling strand handling for genomic ranges.</p> <p>For GRanges, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules.</p> <p>TRUE means strand is ignored and only genomic coordinates are compared.</p> <p>For IRanges, this argument has no effect because there is no strand.</p>
threads	<p>Integer scalar number of worker threads to use.</p> <p>Use 1 for the most conservative behavior and easiest debugging.</p> <p>Use larger values on multicore machines when throughput matters.</p> <p>For repeated-query workloads, combine a prebuilt index from fast_build_index(subject) with a thread count chosen empirically on your hardware.</p> <p>fastRanges is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.</p>
deterministic	<p>Logical scalar controlling output order.</p> <p>TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts.</p> <p>FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.</p>
include_na_group	<p>Logical scalar. If TRUE, missing group values are counted as "&lt;NA&gt;" instead of being dropped.</p>

## Details

deterministic does not change returned group counts for this summary output.

This function answers the question: "for each query, how many overlaps came from each subject group?"

Rows correspond to query ranges.

Columns correspond to distinct values in mcols(subject)[[group\_col]].

## Value

Integer matrix with one row per query and one column per group.

## Overlap semantics

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```
type = "any"
query  :  |-----|
subject: |-----|
```

```
type = "within"
subject: |-----|
query  :  |-----|
```

```
type = "start"
query  :  |-----|
subject: |-----|
```

```
type = "end"
query  :  |-----|
subject: |-----|
```

```
type = "equal"
query  :  |-----|
subject: |-----|
```

```
gap / min_overlap controls
query : |-----| |-----| : subject
      < gap >
```

The middle distance is the gap. A hit is allowed when this distance is  $\leq$  max\_gap (for max\_gap  $\geq$  0), and overlap width is  $\geq$  min\_overlap.

Beginner-friendly interpretation:

type = "any" asks "do these ranges touch or overlap closely enough to count?"

type = "start" and type = "end" are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

type = "within" asks whether each query lies inside a subject interval.

type = "equal" asks whether query and subject describe the same interval, optionally with endpoint tolerance when max\_gap  $\geq$  0.

This argument grammar is intentionally aligned with Bioconductor overlap APIs (IRanges / GenomicRanges).

## Examples

```
q <- GenomicRanges::GRanges("chr1", IRanges::IRanges(c(1, 10), width = 5))
s <- GenomicRanges::GRanges("chr1", IRanges::IRanges(c(2, 9), width = 5))
S4Vectors::mcols(s)$grp <- c("A", "B")
fast_count_overlaps_by_group(q, s, group_col = "grp")
```

---

fast_coverage	<i>Coverage Across Ranges</i>
---------------	-------------------------------

---

### Description

Compute per-position coverage for input ranges.

### Usage

```
fast_coverage(  
  x,  
  shift = 0L,  
  width = NULL,  
  weight = 1L,  
  method = c("auto", "sort", "hash"),  
  threads = fast_default_threads()  
)
```

### Arguments

x	An IRanges or GRanges object.
shift	Passed to coverage(). Use this to shift intervals before computing coverage.
width	Passed to coverage(). Use this to force the output length.
weight	Passed to coverage(). This can be a scalar or vector and is useful when intervals should contribute weighted counts instead of 1.
method	Coverage method.
threads	Integer scalar thread count. Reserved for API consistency.

### Details

Coverage answers the question: "how many ranges cover each position?"

For IRanges, the result is one Rle.

For GRanges, the result is an RleList, usually one coverage track per sequence level.

### Value

Rle (for IRanges) or RleList (for GRanges).

### Examples

```
x <- IRanges::IRanges(start = c(1, 4, 10), end = c(5, 8, 12))  
fast_coverage(x)
```

fast\_default\_threads    *Default Thread Count*

---

**Description**

Returns the default thread count used by fastRanges overlap routines.

**Usage**

```
fast_default_threads()
```

**Details**

The default is controlled by `getOption("fastRanges.threads")` and falls back to 1L.

This helper is mainly useful when you want package-wide thread control without passing `threads =` to every call.

Example:

```
options(fastRanges.threads = 8L) sets the default thread count for later calls that do not specify threads explicitly.
```

**Value**

Integer scalar thread count.

**Examples**

```
fast_default_threads()
old_threads <- getOption("fastRanges.threads")
options(fastRanges.threads = 3L)
fast_default_threads()
options(fastRanges.threads = old_threads)
```

---

fast\_disjoin            *Disjoin Ranges*

---

**Description**

Return non-overlapping segments induced by input ranges.

**Usage**

```
fast_disjoin(x, ignore_strand = FALSE, with_revmap = FALSE)
```

**Arguments**

x	An IRanges or GRanges object.
ignore_strand	Logical scalar. Ignored for non-genomic ranges.
with_revmap	Logical scalar. If TRUE, include reverse mapping from each output range back to contributing input ranges.

**Details**

fast\_disjoin() cuts the covered span of x into the smallest non-overlapping pieces.

**Value**

An object of the same range class as x.

**Examples**

```
x <- IRanges::IRanges(start = c(1, 4, 10), end = c(5, 8, 12))
fast_disjoin(x)
```

---

fast\_distance\_to\_nearest

*Distance to Nearest Subject Range*

---

**Description**

Compute nearest-neighbor mapping from query ranges to subject ranges.

**Usage**

```
fast_distance_to_nearest(
  query,
  subject,
  ignore_strand = FALSE,
  threads = fast_default_threads()
)
```

**Arguments**

query	An IRanges or GRanges query object.
subject	An IRanges or GRanges subject object.
ignore_strand	Logical scalar. Ignored for non-genomic ranges.
threads	Integer scalar thread count. Included for API consistency.

**Details**

This function currently returns the same object shape as fast\_nearest(). It is provided so users can choose the more explicit name when they care about the distance column.

**Value**

A `S4Vectors::DataFrame` with `query_id`, `subject_id`, and `distance` columns.

**Examples**

```
q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
fast_distance_to_nearest(q, s)
```

---

fast\_find\_overlaps      *Find Overlaps with Deterministic Multithreading*

---

**Description**

Compute overlap pairs between query and subject using a multithreaded C++ backend. The result is a `Hits` object compatible with Bioconductor workflows.

**Usage**

```
fast_find_overlaps(
  query,
  subject,
  select = c("all", "first", "last", "arbitrary"),
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE
)
```

**Arguments**

<code>query</code>	An <code>IRanges</code> or <code>GRanges</code> query object.
<code>subject</code>	An <code>IRanges/GRanges</code> object or a <code>fast_ranges_index</code> . Use <code>fast_build_index(subject)</code> when the same subject is reused across many overlap queries.
<code>select</code>	Character scalar controlling whether all hits are returned or a single subject match is selected per query. Use "all" to return a <code>Hits</code> object. Use "first", "last", or "arbitrary" to return an integer vector with one selected subject index per query and <code>NA</code> for queries with no hit.
<code>max_gap</code>	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap. Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes.

	Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with <code>IRanges::findOverlaps()</code> / <code>GenomicRanges::findOverlaps()</code> .
<code>min_overlap</code>	Integer scalar minimum overlap width, in bases. $\emptyset$ is the least strict setting. Larger values require wider shared overlap and therefore return fewer hits. This argument matters only when the selected <code>type</code> allows an actual overlap width to be measured.
<code>type</code>	Character scalar describing what "match" means. "any" matches any overlap that satisfies <code>max_gap / min_overlap</code> . "start" matches ranges with compatible start coordinates. "end" matches ranges with compatible end coordinates. "within" matches queries contained inside subjects. "equal" matches queries and subjects with the same interval, or with start/end differences no larger than <code>max_gap</code> when tolerance is allowed.
<code>ignore_strand</code>	Logical scalar controlling strand handling for genomic ranges. For <code>GRanges</code> , <code>FALSE</code> means "+", "-", and "*" are interpreted using standard Bioconductor strand rules. <code>TRUE</code> means strand is ignored and only genomic coordinates are compared. For <code>IRanges</code> , this argument has no effect because there is no strand.
<code>threads</code>	Integer scalar number of worker threads to use. Use 1 for the most conservative behavior and easiest debugging. Use larger values on multicore machines when throughput matters. For repeated-query workloads, combine a prebuilt index from <code>fast_build_index(subject)</code> with a thread count chosen empirically on your hardware. <code>fastRanges</code> is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.
<code>deterministic</code>	Logical scalar controlling output order. <code>TRUE</code> returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts. <code>FALSE</code> allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.

## Details

This is the core matching function in `fastRanges`.

Think of it as answering the question: "for each query range, which subject ranges satisfy my overlap rule?"

The return value is a `Hits` object. The important pieces are:

`queryHits(h)` gives the row numbers from query.

`subjectHits(h)` gives the matching row numbers from subject.

If you need only counts or a yes/no answer, prefer `fast_count_overlaps()` or `fast_overlaps_any()`, because they return simpler summaries.

Compatibility notes:

`fastRanges` aims to stay close to Bioconductor overlap semantics for supported inputs, and its outputs are routinely validated against `IRanges::findOverlaps()` / `GenomicRanges::findOverlaps()`.

Currently supported core input types are `IRanges` and `GRanges`.

Empty-range semantics are delegated to Bioconductor-compatible reference behavior.

Circular genomic sequences are not currently supported and will raise an explicit error.

`GRangesList` inputs are not currently supported and will raise an explicit error.

Performance notes:

For one-off overlap calls, use the raw `subject`.

For repeated-query or throughput-oriented workloads, build a reusable index once with `fast_build_index(subject)` and pass that index as `subject`.

For maximum multithreaded throughput, consider `deterministic = FALSE` when output order is not important.

## Value

If `select = "all"`, a `S4Vectors::Hits` object.

Otherwise, an integer vector of length `length(query)` containing one selected subject index per query and `NA` when no subject matched.

## Overlap semantics

`query` is the range set you ask about. `subject` is the range set you compare it against.

Core interval semantics (ASCII schematic):

```
type = "any"
query  :  |-----|
subject: |-----|
```

```
type = "within"
subject: |-----|
query   :  |-----|
```

```
type = "start"
query  :  |-----|
subject: |-----|
```

```
type = "end"
query  :  |-----|
subject: |-----|
```

```
type = "equal"
query  :  |-----|
```

```

subject:  |-----|

gap / min_overlap controls
query  : |-----|      |-----| : subject
          < gap >

```

The middle distance is the gap. A hit is allowed when this distance is  $\leq$  `max_gap` (for `max_gap >= 0`), and overlap width is  $\geq$  `min_overlap`.

Beginner-friendly interpretation:

`type = "any"` asks "do these ranges touch or overlap closely enough to count?"

`type = "start"` and `type = "end"` are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

`type = "within"` asks whether each query lies inside a subject interval.

`type = "equal"` asks whether query and subject describe the same interval, optionally with endpoint tolerance when `max_gap >= 0`.

This argument grammar is intentionally aligned with Bioconductor overlap APIs (`IRanges` / `GenomicRanges`).

## Examples

```

q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
h <- fast_find_overlaps(q, s, threads = 1)
length(h)

```

---

fast\_find\_overlaps\_iter

*Create an Overlap Iterator*

---

## Description

Create an iterator that computes overlaps in query chunks.

## Usage

```

fast_find_overlaps_iter(
  query,
  subject,
  chunk_size = 50000L,
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE
)

```

**Arguments**

query	An IRanges or GRanges query object.
subject	An IRanges/GRanges object or a fast_ranges_index.
chunk_size	Integer scalar number of query ranges to process in one iterator step. Smaller values use less memory and give more frequent progress points. Larger values usually improve throughput but each call to fast_iter_next() does more work.
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap. Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes. Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with IRanges::findOverlaps() / GenomicRanges::findOverlaps().
min_overlap	Integer scalar minimum overlap width, in bases. 0 is the least strict setting. Larger values require wider shared overlap and therefore return fewer hits. This argument matters only when the selected type allows an actual overlap width to be measured.
type	Character scalar describing what "match" means. "any" matches any overlap that satisfies max_gap / min_overlap. "start" matches ranges with compatible start coordinates. "end" matches ranges with compatible end coordinates. "within" matches queries contained inside subjects. "equal" matches queries and subjects with the same interval, or with start/end differences no larger than max_gap when tolerance is allowed.
ignore_strand	Logical scalar controlling strand handling for genomic ranges. For GRanges, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules. TRUE means strand is ignored and only genomic coordinates are compared. For IRanges, this argument has no effect because there is no strand.
threads	Integer scalar number of worker threads to use. Use 1 for the most conservative behavior and easiest debugging. Use larger values on multicore machines when throughput matters. For repeated-query workloads, combine a prebuilt index from fast_build_index(subject) with a thread count chosen empirically on your hardware. fastRanges is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.

deterministic Logical scalar controlling output order.  
 TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts.  
 FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.

## Details

Use the iterator API when query is large and you do not want to materialize all overlap hits in one call.

Typical workflow:

```
create the iterator with fast_find_overlaps_iter()
inspect progress with fast_iter_has_next()
pull one chunk with fast_iter_next()
or collect all remaining chunks with fast_iter_collect()
```

## Value

A fast\_ranges\_iter iterator object.

## Overlap semantics

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```
type = "any"
query  :  |-----|
subject: |-----|
```

```
type = "within"
subject: |-----|
query  :  |-----|
```

```
type = "start"
query  :  |-----|
subject: |-----|
```

```
type = "end"
query  :  |-----|
subject: |-----|
```

```
type = "equal"
query  :  |-----|
subject: |-----|
```

gap / min\_overlap controls

```
query  : |-----|      |-----| : subject
        < gap >
```

The middle distance is the gap. A hit is allowed when this distance is  $\leq \text{max\_gap}$  (for  $\text{max\_gap} \geq 0$ ), and overlap width is  $\geq \text{min\_overlap}$ .

Beginner-friendly interpretation:

`type = "any"` asks "do these ranges touch or overlap closely enough to count?"

`type = "start"` and `type = "end"` are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

`type = "within"` asks whether each query lies inside a subject interval.

`type = "equal"` asks whether query and subject describe the same interval, optionally with endpoint tolerance when  $\text{max\_gap} \geq 0$ .

This argument grammar is intentionally aligned with Bioconductor overlap APIs (`IRanges` / `GenomicRanges`).

## Examples

```
q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
it <- fast_find_overlaps_iter(q, s, chunk_size = 2L)
fast_iter_has_next(it)
```

---

fast\_follow

*Follow Query Ranges*

---

## Description

Return the index of the first subject range that is strictly before each query range.

## Usage

```
fast_follow(
  query,
  subject,
  ignore_strand = FALSE,
  threads = fast_default_threads()
)
```

## Arguments

<code>query</code>	An <code>IRanges</code> or <code>GRanges</code> query object.
<code>subject</code>	An <code>IRanges</code> or <code>GRanges</code> subject object.
<code>ignore_strand</code>	Logical scalar. Ignored for non-genomic ranges.
<code>threads</code>	Integer scalar thread count. Included for API consistency.

**Details**

For each query range, return the index of the first subject range that comes strictly before the query.

**Value**

Integer vector of subject indices, with NA for unmatched queries.

**Examples**

```
q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
fast_follow(q, s)
```

---

fast\_gaps

*Gaps Between Ranges*

---

**Description**

Compute uncovered regions between ranges.

**Usage**

```
fast_gaps(x, start = NULL, end = NULL, ignore_strand = FALSE)
```

**Arguments**

**x** An IRanges or GRanges object.

**start, end** Optional integer bounds for non-genomic ranges. These are most useful for IRanges. For GRanges, sequence lengths are usually taken from seqinfo(x).

**ignore\_strand** Logical scalar. Ignored for non-genomic ranges.

**Details**

fast\_gaps() returns the regions not covered by x inside the requested bounds.

**Value**

An object of the same range class as x.

**Examples**

```
x <- IRanges::IRanges(start = c(1, 4, 10), end = c(5, 8, 12))
fast_gaps(x, start = 1L, end = 15L)
```

---

fast_index_stats	<i>Index Summary Statistics</i>
------------------	---------------------------------

---

### Description

Summarize index size and partition structure.

### Usage

```
fast_index_stats(index, detailed = FALSE)
```

### Arguments

index	A fast_ranges_index object.
detailed	Logical scalar. If TRUE, returns partition-level details.

### Details

Use this function to inspect how the subject was partitioned internally and to get a rough sense of memory footprint.

subject\_n is the number of indexed ranges.

partition\_n is the number of internal partitions, usually driven by sequence structure.

index\_size\_mb is the in-memory object size, not the serialized file size.

### Value

By default, a one-row `S4Vectors::DataFrame` with summary fields. If `detailed = TRUE`, returns a list with summary and partitions.

### Examples

```
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
idx <- fast_build_index(s)
fast_index_stats(idx)
```

---

```
fast_inner_overlap_join
      Inner Overlap Join
```

---

## Description

Convenience wrapper for `fast_overlap_join(..., join = "inner")`.

## Usage

```
fast_inner_overlap_join(
  query,
  subject,
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE,
  query_prefix = "query_",
  subject_prefix = "subject_"
)
```

## Arguments

query	An IRanges or GRanges query object.
subject	An IRanges/GRanges object or a fast_ranges_index. Use <code>fast_build_index(subject)</code> when the same subject is reused across many overlap queries.
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use <code>-1</code> to require a true overlap. Use <code>0</code> to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes. Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with <code>IRanges::findOverlaps()</code> / <code>GenomicRanges::findOverlaps()</code> .
min_overlap	Integer scalar minimum overlap width, in bases. <code>0</code> is the least strict setting. Larger values require wider shared overlap and therefore return fewer hits. This argument matters only when the selected type allows an actual overlap width to be measured.
type	Character scalar describing what "match" means. "any" matches any overlap that satisfies <code>max_gap / min_overlap</code> .

	"start" matches ranges with compatible start coordinates.
	"end" matches ranges with compatible end coordinates.
	"within" matches queries contained inside subjects.
	"equal" matches queries and subjects with the same interval, or with start/end differences no larger than max_gap when tolerance is allowed.
ignore_strand	Logical scalar controlling strand handling for genomic ranges. For GRanges, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules. TRUE means strand is ignored and only genomic coordinates are compared. For IRanges, this argument has no effect because there is no strand.
threads	Integer scalar number of worker threads to use. Use 1 for the most conservative behavior and easiest debugging. Use larger values on multicore machines when throughput matters. For repeated-query workloads, combine a prebuilt index from fast_build_index(subject) with a thread count chosen empirically on your hardware. fastRanges is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.
deterministic	Logical scalar controlling output order. TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts. FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.
query_prefix	Prefix added to columns derived from query. This helps you see which output columns came from the query object.
subject_prefix	Prefix added to columns derived from subject. This helps you see which output columns came from the subject object.

**Value**

A data.frame overlap join result.

**Overlap semantics**

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```

type = "any"
query  :  |-----|
subject: |-----|

type = "within"
subject: |-----|
query   :  |-----|

```

```

type = "start"
query  :   |-----|
subject: |-----|

type = "end"
query  :   |-----|
subject: |-----|

type = "equal"
query  :   |-----|
subject: |-----|

gap / min_overlap controls
query  : |-----|   |-----| : subject
          < gap >

```

The middle distance is the gap. A hit is allowed when this distance is  $\leq$  `max_gap` (for `max_gap`  $\geq$  0), and overlap width is  $\geq$  `min_overlap`.

Beginner-friendly interpretation:

`type = "any"` asks "do these ranges touch or overlap closely enough to count?"

`type = "start"` and `type = "end"` are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

`type = "within"` asks whether each query lies inside a subject interval.

`type = "equal"` asks whether query and subject describe the same interval, optionally with endpoint tolerance when `max_gap`  $\geq$  0.

This argument grammar is intentionally aligned with Bioconductor overlap APIs (`IRanges` / `GenomicRanges`).

## Examples

```

q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
fast_inner_overlap_join(q, s)

```

---

fast\_iter\_collect      *Collect All Iterator Chunks*

---

## Description

Materialize all overlap chunks from an iterator into a single `Hits` object.

## Usage

```
fast_iter_collect(iterator)
```

## Arguments

`iter`                    A `fast_ranges_iter` object.

**Details**

This collects only the chunks that have not yet been consumed.

If you want all hits from the beginning after some chunks were already read, call `fast_iter_reset(iter)` first and then `fast_iter_collect(iter)`.

**Value**

A `S4Vectors::Hits` object.

**Examples**

```
q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
it <- fast_find_overlaps_iter(q, s, chunk_size = 2L)
fast_iter_collect(it)
```

---

`fast_iter_has_next`      *Check if an Iterator Has Remaining Chunks*

---

**Description**

Check if an Iterator Has Remaining Chunks

**Usage**

```
fast_iter_has_next(iter)
```

**Arguments**

`iter`                    A `fast_ranges_iter` object.

**Value**

Logical scalar indicating whether more query chunks remain.

**Examples**

```
q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
it <- fast_find_overlaps_iter(q, s, chunk_size = 2L)
fast_iter_has_next(it)
```

---

fast_iter_next	<i>Advance an Overlap Iterator</i>
----------------	------------------------------------

---

**Description**

Compute overlaps for the next query chunk.

**Usage**

```
fast_iter_next(iter)
```

**Arguments**

iter            A fast\_ranges\_iter object.

**Details**

Each call advances the iterator state.

The returned Hits object uses global query indices, not chunk-local indices, so you can combine chunk outputs safely.

**Value**

A S4Vectors::Hits object for the next chunk, with global query indices.

**Examples**

```
q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
it <- fast_find_overlaps_iter(q, s, chunk_size = 2L)
fast_iter_next(it)
```

---

fast_iter_reset	<i>Reset an Overlap Iterator</i>
-----------------	----------------------------------

---

**Description**

Reset iterator position to the first query chunk.

**Usage**

```
fast_iter_reset(iter)
```

**Arguments**

iter            A fast\_ranges\_iter object.

**Details**

After reset, the next call to `fast_iter_next()` starts again from the first query chunk.

**Value**

Invisibly returns `iter`.

**Examples**

```
q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
it <- fast_find_overlaps_iter(q, s, chunk_size = 2L)
fast_iter_reset(it)
```

---

`fast_left_overlap_join`

*Left Overlap Join*

---

**Description**

Convenience wrapper for `fast_overlap_join(..., join = "left")`.

**Usage**

```
fast_left_overlap_join(
  query,
  subject,
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE,
  query_prefix = "query_",
  subject_prefix = "subject_"
)
```

**Arguments**

<code>query</code>	An IRanges or GRanges query object.
<code>subject</code>	An IRanges/GRanges object or a <code>fast_ranges_index</code> . Use <code>fast_build_index(subject)</code> when the same subject is reused across many overlap queries.
<code>max_gap</code>	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use <code>-1</code> to require a true overlap. Use <code>0</code> to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes.

	Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with <code>IRanges::findOverlaps()</code> / <code>GenomicRanges::findOverlaps()</code> .
<code>min_overlap</code>	Integer scalar minimum overlap width, in bases. <code>0</code> is the least strict setting. Larger values require wider shared overlap and therefore return fewer hits. This argument matters only when the selected type allows an actual overlap width to be measured.
<code>type</code>	Character scalar describing what "match" means. "any" matches any overlap that satisfies <code>max_gap / min_overlap</code> . "start" matches ranges with compatible start coordinates. "end" matches ranges with compatible end coordinates. "within" matches queries contained inside subjects. "equal" matches queries and subjects with the same interval, or with start/end differences no larger than <code>max_gap</code> when tolerance is allowed.
<code>ignore_strand</code>	Logical scalar controlling strand handling for genomic ranges. For <code>GRanges</code> , <code>FALSE</code> means "+", "-", and "*" are interpreted using standard Bioconductor strand rules. <code>TRUE</code> means strand is ignored and only genomic coordinates are compared. For <code>IRanges</code> , this argument has no effect because there is no strand.
<code>threads</code>	Integer scalar number of worker threads to use. Use 1 for the most conservative behavior and easiest debugging. Use larger values on multicore machines when throughput matters. For repeated-query workloads, combine a prebuilt index from <code>fast_build_index(subject)</code> with a thread count chosen empirically on your hardware. <code>fastRanges</code> is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.
<code>deterministic</code>	Logical scalar controlling output order. <code>TRUE</code> returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts. <code>FALSE</code> allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.
<code>query_prefix</code>	Prefix added to columns derived from query. This helps you see which output columns came from the query object.
<code>subject_prefix</code>	Prefix added to columns derived from subject. This helps you see which output columns came from the subject object.

**Value**

A data.frame overlap join result.

## Overlap semantics

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```

type = "any"
query  :  |-----|
subject: |-----|

type = "within"
subject: |-----|
query   :  |-----|

type = "start"
query   :  |-----|
subject: |-----|

type = "end"
query   :          |-----|
subject: |-----|

type = "equal"
query   :  |-----|
subject:  |-----|

gap / min_overlap controls
query  : |-----|      |-----| : subject
          < gap >

```

The middle distance is the gap. A hit is allowed when this distance is  $\leq \text{max\_gap}$  (for  $\text{max\_gap} \geq 0$ ), and overlap width is  $\geq \text{min\_overlap}$ .

Beginner-friendly interpretation:

type = "any" asks "do these ranges touch or overlap closely enough to count?"

type = "start" and type = "end" are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

type = "within" asks whether each query lies inside a subject interval.

type = "equal" asks whether query and subject describe the same interval, optionally with endpoint tolerance when  $\text{max\_gap} \geq 0$ .

This argument grammar is intentionally aligned with Bioconductor overlap APIs (IRanges / GenomicRanges).

## Examples

```

q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
fast_left_overlap_join(q, s)

```

---

fast_load_index	<i>Load a Reusable Subject Index</i>
-----------------	--------------------------------------

---

**Description**

Load a fast\_ranges\_index object saved with fast\_save\_index().

**Usage**

```
fast_load_index(path)
```

**Arguments**

path                    File path to a serialized index.

**Details**

This function validates that the file contains the fields required by fastRanges before returning the object.

**Value**

A fast\_ranges\_index object.

**Examples**

```
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
idx <- fast_build_index(s)
f <- tempfile(fileext = ".rds")
fast_save_index(idx, f)
idx2 <- fast_load_index(f)
unlink(f)
print(idx2)
```

---

fast_nearest	<i>Nearest Subject Range per Query</i>
--------------	--

---

**Description**

Compute nearest-neighbor mapping from query ranges to subject ranges.

**Usage**

```
fast_nearest(
  query,
  subject,
  ignore_strand = FALSE,
  threads = fast_default_threads()
)
```

**Arguments**

query	An IRanges or GRanges query object.
subject	An IRanges or GRanges subject object.
ignore_strand	Logical scalar. Ignored for non-genomic ranges.
threads	Integer scalar thread count. Included for API consistency.

**Details**

These functions answer nearest-neighbor questions rather than overlap questions.

fast\_nearest() and fast\_distance\_to\_nearest() return one row per matched query.

query\_id is the row index in query.

subject\_id is the row index of the nearest subject.

distance is 0 when the query overlaps the subject and positive when the nearest subject is separated by a gap.

**Value**

A S4Vectors::DataFrame with query\_id, subject\_id, and distance columns.

**Examples**

```
q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
fast_nearest(q, s)
```

---

fast_overlaps_any	<i>Overlap Existence per Query</i>
-------------------	------------------------------------

---

**Description**

Return TRUE for queries that overlap at least one subject range.

**Usage**

```
fast_overlaps_any(
  query,
  subject,
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE
)
```

**Arguments**

query	An IRanges or GRanges query object.
subject	An IRanges/GRanges object or a fast_ranges_index. Use fast_build_index(subject) when the same subject is reused across many overlap queries.
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap. Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes. Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with IRanges::findOverlaps() / GenomicRanges::findOverlaps().
min_overlap	Integer scalar minimum overlap width, in bases. 0 is the least strict setting. Larger values require wider shared overlap and therefore return fewer hits. This argument matters only when the selected type allows an actual overlap width to be measured.
type	Character scalar describing what "match" means. "any" matches any overlap that satisfies max_gap / min_overlap. "start" matches ranges with compatible start coordinates. "end" matches ranges with compatible end coordinates. "within" matches queries contained inside subjects. "equal" matches queries and subjects with the same interval, or with start/end differences no larger than max_gap when tolerance is allowed.
ignore_strand	Logical scalar controlling strand handling for genomic ranges. For GRanges, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules. TRUE means strand is ignored and only genomic coordinates are compared. For IRanges, this argument has no effect because there is no strand.
threads	Integer scalar number of worker threads to use. Use 1 for the most conservative behavior and easiest debugging. Use larger values on multicore machines when throughput matters. For repeated-query workloads, combine a prebuilt index from fast_build_index(subject) with a thread count chosen empirically on your hardware. fastRanges is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.
deterministic	Logical scalar controlling output order. TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts. FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.

**Details**

deterministic does not change returned logical values for this summary output.

Returns one logical value per query range.

TRUE means at least one subject range matched.

FALSE means no subject range matched.

**Value**

Logical vector with one element per query range.

**Overlap semantics**

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```
type = "any"
query  :  |-----|
subject: |-----|
```

```
type = "within"
subject: |-----|
query  :  |-----|
```

```
type = "start"
query  :  |-----|
subject: |-----|
```

```
type = "end"
query  :          |-----|
subject: |-----|
```

```
type = "equal"
query  :  |-----|
subject:  |-----|
```

```
gap / min_overlap controls
query  : |-----|      |-----| : subject
          < gap >
```

The middle distance is the gap. A hit is allowed when this distance is  $\leq$  max\_gap (for max\_gap  $\geq$  0), and overlap width is  $\geq$  min\_overlap.

Beginner-friendly interpretation:

type = "any" asks "do these ranges touch or overlap closely enough to count?"

type = "start" and type = "end" are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

type = "within" asks whether each query lies inside a subject interval.

type = "equal" asks whether query and subject describe the same interval, optionally with endpoint tolerance when max\_gap >= 0.

This argument grammar is intentionally aligned with Bioconductor overlap APIs (IRanges / GenomicRanges).

## Examples

```
q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
fast_overlaps_any(q, s, threads = 1)
```

---

fast\_overlap\_aggregate

*Aggregate Subject Metadata Over Overlaps*

---

## Description

Aggregate a numeric subject metadata column across overlaps for each query.

## Usage

```
fast_overlap_aggregate(
  query,
  subject,
  value_col = NULL,
  fun = c("count", "sum", "mean", "min", "max"),
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE,
  na_rm = TRUE
)
```

## Arguments

query	An IRanges or GRanges query object.
subject	An IRanges/GRanges object or a fast_ranges_index. Use fast_build_index(subject) when the same subject is reused across many overlap queries.
value_col	Subject metadata column name containing numeric values. This is required unless fun = "count".
fun	Aggregation function: one of "count", "sum", "mean", "min", or "max".
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap.

	<p>Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes.</p> <p>Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly.</p> <p>Units are bases. The meaning is intentionally aligned with <code>IRanges::findOverlaps()</code> / <code>GenomicRanges::findOverlaps()</code>.</p>
min_overlap	<p>Integer scalar minimum overlap width, in bases.</p> <p>0 is the least strict setting.</p> <p>Larger values require wider shared overlap and therefore return fewer hits.</p> <p>This argument matters only when the selected type allows an actual overlap width to be measured.</p>
type	<p>Character scalar describing what "match" means.</p> <p>"any" matches any overlap that satisfies <code>max_gap / min_overlap</code>.</p> <p>"start" matches ranges with compatible start coordinates.</p> <p>"end" matches ranges with compatible end coordinates.</p> <p>"within" matches queries contained inside subjects.</p> <p>"equal" matches queries and subjects with the same interval, or with start/end differences no larger than <code>max_gap</code> when tolerance is allowed.</p>
ignore_strand	<p>Logical scalar controlling strand handling for genomic ranges.</p> <p>For <code>GRanges</code>, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules.</p> <p>TRUE means strand is ignored and only genomic coordinates are compared.</p> <p>For <code>IRanges</code>, this argument has no effect because there is no strand.</p>
threads	<p>Integer scalar number of worker threads to use.</p> <p>Use 1 for the most conservative behavior and easiest debugging.</p> <p>Use larger values on multicore machines when throughput matters.</p> <p>For repeated-query workloads, combine a prebuilt index from <code>fast_build_index(subject)</code> with a thread count chosen empirically on your hardware.</p> <p><code>fastRanges</code> is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.</p>
deterministic	<p>Logical scalar controlling output order.</p> <p>TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts.</p> <p>FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.</p>
na_rm	<p>Logical scalar. If TRUE, remove missing values in aggregation.</p>

### Details

`deterministic` does not change returned aggregate values for this summary output.

This function summarizes subject metadata over the overlap hits of each query.

Examples:

use fun = "count" to count overlaps  
 use fun = "sum" to sum a signal column over matched subjects  
 use fun = "mean" to compute average matched score per query

## Value

Numeric vector with one value per query.

## Overlap semantics

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```
type = "any"
query  :  |-----|
subject: |-----|
```

```
type = "within"
subject: |-----|
query  :  |-----|
```

```
type = "start"
query  :  |-----|
subject: |-----|
```

```
type = "end"
query  :  |-----|
subject: |-----|
```

```
type = "equal"
query  :  |-----|
subject: |-----|
```

```
gap / min_overlap controls
query : |-----| |-----| : subject
      < gap >
```

The middle distance is the gap. A hit is allowed when this distance is  $\leq$  max\_gap (for max\_gap  $\geq$  0), and overlap width is  $\geq$  min\_overlap.

Beginner-friendly interpretation:

type = "any" asks "do these ranges touch or overlap closely enough to count?"

type = "start" and type = "end" are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

type = "within" asks whether each query lies inside a subject interval.

type = "equal" asks whether query and subject describe the same interval, optionally with endpoint tolerance when max\_gap  $\geq$  0.

This argument grammar is intentionally aligned with Bioconductor overlap APIs (IRanges / GenomicRanges).

**Examples**

```
q <- GenomicRanges::GRanges("chr1", IRanges::IRanges(c(1, 10), width = 5))
s <- GenomicRanges::GRanges("chr1", IRanges::IRanges(c(2, 9), width = 5))
S4Vectors::mcols(s)$score <- c(2, 5)
fast_overlap_aggregate(q, s, value_col = "score", fun = "sum")
```

---

fast\_overlap\_join      *Join Ranges by Overlap*

---

**Description**

Build a metadata-preserving overlap join with a consistent, tidy tabular output grammar.

**Usage**

```
fast_overlap_join(
  query,
  subject,
  join = c("inner", "left"),
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE,
  query_prefix = "query_",
  subject_prefix = "subject_"
)
```

**Arguments**

query	An IRanges or GRanges query object.
subject	An IRanges/GRanges object or a fast_ranges_index. Use fast_build_index(subject) when the same subject is reused across many overlap queries.
join	Join mode. "inner" returns one row per overlap hit and drops queries with no hit. "left" keeps every query at least once. Queries with no hit get NA values in subject columns.
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap. Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes. Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with IRanges::findOverlaps() / GenomicRanges::findOverlaps().

min_overlap	Integer scalar minimum overlap width, in bases. 0 is the least strict setting. Larger values require wider shared overlap and therefore return fewer hits. This argument matters only when the selected type allows an actual overlap width to be measured.
type	Character scalar describing what "match" means. "any" matches any overlap that satisfies max_gap / min_overlap. "start" matches ranges with compatible start coordinates. "end" matches ranges with compatible end coordinates. "within" matches queries contained inside subjects. "equal" matches queries and subjects with the same interval, or with start/end differences no larger than max_gap when tolerance is allowed.
ignore_strand	Logical scalar controlling strand handling for genomic ranges. For GRanges, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules. TRUE means strand is ignored and only genomic coordinates are compared. For IRanges, this argument has no effect because there is no strand.
threads	Integer scalar number of worker threads to use. Use 1 for the most conservative behavior and easiest debugging. Use larger values on multicore machines when throughput matters. For repeated-query workloads, combine a prebuilt index from fast_build_index(subject) with a thread count chosen empirically on your hardware. fastRanges is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.
deterministic	Logical scalar controlling output order. TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts. FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.
query_prefix	Prefix added to columns derived from query. This helps you see which output columns came from the query object.
subject_prefix	Prefix added to columns derived from subject. This helps you see which output columns came from the subject object.

## Details

The join family turns overlap hits into beginner-friendly tabular output.

Output always starts with query\_id and subject\_id.

query\_id is the 1-based row index from query.

subject\_id is the 1-based row index from subject, or NA for unmatched queries in a left join.

The remaining columns are prefixed copies of the original range columns and metadata columns.

**Value**

A data.frame with overlap ids and prefixed query/subject columns.

**Overlap semantics**

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```

type = "any"
query  :  |-----|
subject: |-----|

type = "within"
subject: |-----|
query   :  |-----|

type = "start"
query   :  |-----|
subject: |-----|

type = "end"
query   :      |-----|
subject: |-----|

type = "equal"
query   :  |-----|
subject:  |-----|

gap / min_overlap controls
query  : |-----|      |-----| : subject
          < gap >

```

The middle distance is the gap. A hit is allowed when this distance is  $\leq \text{max\_gap}$  (for  $\text{max\_gap} \geq 0$ ), and overlap width is  $\geq \text{min\_overlap}$ .

Beginner-friendly interpretation:

type = "any" asks "do these ranges touch or overlap closely enough to count?"

type = "start" and type = "end" are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

type = "within" asks whether each query lies inside a subject interval.

type = "equal" asks whether query and subject describe the same interval, optionally with endpoint tolerance when  $\text{max\_gap} \geq 0$ .

This argument grammar is intentionally aligned with Bioconductor overlap APIs (IRanges / GenomicRanges).

**Examples**

```
q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
fast_overlap_join(q, s, join = "inner", threads = 1)
```

---

fast_precede	<i>Precede Query Ranges</i>
--------------	-----------------------------

---

**Description**

Return the index of the first subject range that is strictly after each query range.

**Usage**

```
fast_precede(  
  query,  
  subject,  
  ignore_strand = FALSE,  
  threads = fast_default_threads()  
)
```

**Arguments**

query	An IRanges or GRanges query object.
subject	An IRanges or GRanges subject object.
ignore_strand	Logical scalar. Ignored for non-genomic ranges.
threads	Integer scalar thread count. Included for API consistency.

**Details**

For each query range, return the index of the first subject range that comes strictly after the query.

**Value**

Integer vector of subject indices, with NA for unmatched queries.

**Examples**

```
q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
fast_precede(q, s)
```

---

fast\_ranges\_example    *Example Genomic Ranges for Documentation, Tests, and Tutorials*

---

## Description

fast\_ranges\_example is a small reproducible dataset shipped with the package. It is intended for documentation examples, teaching, unit tests, and quick smoke checks of overlap workflows.

## Format

A named list with two components:

**query** A GRanges object with 6 genomic intervals and metadata columns query\_id and score.

**subject** A GRanges object with 7 genomic intervals and metadata columns gene\_id and biotype.

## Details

The object is a named list with two GRanges components:

- query: six example genomic intervals that behave like user-supplied peaks, windows, or regions of interest.
- subject: seven example genomic intervals that behave like annotation features such as genes, promoters, enhancers, or other reference ranges.

Metadata columns are included so examples can demonstrate joins, grouped summaries, and overlap aggregation:

- query contains query\_id and score
- subject contains gene\_id and biotype

The same records are also distributed as plain BED files in `inst/extdata/query_peaks.bed` and `inst/extdata/subject_genes.bed`. Use the in-memory dataset when you want a ready-to-run example in R, and use the BED files when you want to demonstrate file import or command-line workflows.

This dataset is intentionally small and synthetic. It is designed for examples and tests, not as a biological reference resource.

## Source

Generated from `data-raw/make_example_data.R`, which also writes matching BED files to `inst/extdata/`.

## Examples

```
data("fast_ranges_example", package = "fastRanges")
query <- fast_ranges_example$query
subject <- fast_ranges_example$subject

query
```

```
subject
fast_find_overlaps(query, subject, threads = 1)
```

---

**fast\_range\_intersect** *Intersection of Two Range Sets*

---

**Description**

Compute range-wise intersection.

**Usage**

```
fast_range_intersect(x, y, ignore_strand = FALSE)
```

**Arguments**

`x, y` IRanges or GRanges objects of compatible class.  
`ignore_strand` Logical scalar. Ignored for non-genomic ranges.

**Details**

`fast_range_intersect()` keeps only the coordinate span shared by `x` and `y`.

**Value**

An object of the same range class as `x` and `y`.

**Examples**

```
x <- IRanges::IRanges(start = c(1, 10), end = c(5, 12))
y <- IRanges::IRanges(start = c(3, 20), end = c(8, 21))
fast_range_intersect(x, y)
```

---

**fast\_range\_setdiff** *Set Difference of Two Range Sets*

---

**Description**

Compute ranges in `x` that are not covered by `y`.

**Usage**

```
fast_range_setdiff(x, y, ignore_strand = FALSE)
```

**Arguments**

`x, y` IRanges or GRanges objects of compatible class.  
`ignore_strand` Logical scalar. Ignored for non-genomic ranges.

**Details**

`fast_range_setdiff()` subtracts the covered span of `y` from `x`.

**Value**

An object of the same range class as `x`.

**Examples**

```
x <- IRanges::IRanges(start = c(1, 10), end = c(5, 12))
y <- IRanges::IRanges(start = c(3, 20), end = c(8, 21))
fast_range_setdiff(x, y)
```

---

<code>fast_range_union</code>	<i>Union of Two Range Sets</i>
-------------------------------	--------------------------------

---

**Description**

Compute range-wise union.

**Usage**

```
fast_range_union(x, y, ignore_strand = FALSE)
```

**Arguments**

`x, y` IRanges or GRanges objects of compatible class.  
`ignore_strand` Logical scalar. Ignored for non-genomic ranges.

**Details**

`fast_range_union()` returns the combined interval coverage of `x` and `y`.

**Value**

An object of the same range class as `x` and `y`.

**Examples**

```
x <- IRanges::IRanges(start = c(1, 10), end = c(5, 12))
y <- IRanges::IRanges(start = c(3, 20), end = c(8, 21))
fast_range_union(x, y)
```

---

fast_reduce	<i>Reduce Overlapping Ranges</i>
-------------	----------------------------------

---

**Description**

Merge overlapping or adjacent ranges.

**Usage**

```
fast_reduce(x, ignore_strand = FALSE, min_gap_width = 1L, with_revmap = FALSE)
```

**Arguments**

x	An IRanges or GRanges object.
ignore_strand	Logical scalar. Ignored for non-genomic ranges.
min_gap_width	Integer scalar controlling when nearby ranges should be merged. 1 merges overlapping or directly adjacent ranges. Larger values require larger gaps before two ranges are kept separate.
with_revmap	Logical scalar. If TRUE, include reverse mapping from each output range back to contributing input ranges.

**Details**

fast\_reduce() simplifies a range set by merging runs of overlapping or near-adjacent intervals.

**Value**

An object of the same range class as x.

**Examples**

```
x <- IRanges::IRanges(start = c(1, 4, 10), end = c(5, 8, 12))
fast_reduce(x)
```

---

fast_save_index	<i>Save a Reusable Subject Index</i>
-----------------	--------------------------------------

---

**Description**

Save a fast\_ranges\_index object to disk for reuse across sessions.

**Usage**

```
fast_save_index(index, path, compress = TRUE)
```

**Arguments**

index	A fast_ranges_index object created by fast_build_index().
path	Output file path for the serialized index.
compress	Logical scalar. If TRUE, uses xz compression.

**Details**

Saving an index is useful when index construction is expensive and the same subject set will be queried again in a later R session.

**Value**

Invisibly returns path.

**Examples**

```
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
idx <- fast_build_index(s)
f <- tempfile(fileext = ".rds")
fast_save_index(idx, f)
unlink(f)
```

---

fast\_self\_overlaps      *Self Overlaps*

---

**Description**

Find overlaps within a single range object.

**Usage**

```
fast_self_overlaps(  
  x,  
  max_gap = -1L,  
  min_overlap = 0L,  
  type = c("any", "start", "end", "within", "equal"),  
  ignore_strand = FALSE,  
  threads = fast_default_threads(),  
  deterministic = TRUE,  
  drop_self = TRUE,  
  drop_redundant = TRUE  
)
```

**Arguments**

x	An IRanges or GRanges object.
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap. Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes. Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with IRanges::findOverlaps() / GenomicRanges::findOverlaps().
min_overlap	Integer scalar minimum overlap width, in bases. 0 is the least strict setting. Larger values require wider shared overlap and therefore return fewer hits. This argument matters only when the selected type allows an actual overlap width to be measured.
type	Character scalar describing what "match" means. "any" matches any overlap that satisfies max_gap / min_overlap. "start" matches ranges with compatible start coordinates. "end" matches ranges with compatible end coordinates. "within" matches queries contained inside subjects. "equal" matches queries and subjects with the same interval, or with start/end differences no larger than max_gap when tolerance is allowed.
ignore_strand	Logical scalar controlling strand handling for genomic ranges. For GRanges, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules. TRUE means strand is ignored and only genomic coordinates are compared. For IRanges, this argument has no effect because there is no strand.
threads	Integer scalar number of worker threads to use. Use 1 for the most conservative behavior and easiest debugging. Use larger values on multicore machines when throughput matters. For repeated-query workloads, combine a prebuilt index from fast_build_index(subject) with a thread count chosen empirically on your hardware. fastRanges is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.
deterministic	Logical scalar controlling output order. TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts. FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.
drop_self	Logical scalar. If TRUE, self-hits (i vs i) are removed.
drop_redundant	Logical scalar. If TRUE, redundant pairs are removed for self-comparisons by keeping only query_id < subject_id.

**Details**

fast\_self\_overlaps() compares a range object to itself.

This is useful when you want to know which ranges in one set overlap each other rather than overlap another object.

**Value**

A S4Vectors::Hits object.

**Overlap semantics**

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```

type = "any"
query  :  |-----|
subject: |-----|

type = "within"
subject: |-----|
query   :  |-----|

type = "start"
query   :  |-----|
subject: |-----|

type = "end"
query   :      |-----|
subject: |-----|

type = "equal"
query   :  |-----|
subject:  |-----|

gap / min_overlap controls
query  : |-----|      |-----| : subject
          < gap >

```

The middle distance is the gap. A hit is allowed when this distance is  $\leq$  max\_gap (for max\_gap  $\geq$  0), and overlap width is  $\geq$  min\_overlap.

Beginner-friendly interpretation:

type = "any" asks "do these ranges touch or overlap closely enough to count?"

type = "start" and type = "end" are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

type = "within" asks whether each query lies inside a subject interval.

type = "equal" asks whether query and subject describe the same interval, optionally with endpoint tolerance when max\_gap >= 0.

This argument grammar is intentionally aligned with Bioconductor overlap APIs (IRanges / GenomicRanges).

## Examples

```
x <- IRanges::IRanges(start = c(1, 3, 10), end = c(5, 8, 12))
fast_self_overlaps(x)
```

---

```
fast_semi_overlap_join
```

*Semi Overlap Join*

---

## Description

Return query rows that overlap at least one subject range.

## Usage

```
fast_semi_overlap_join(
  query,
  subject,
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE,
  query_prefix = "query_"
)
```

## Arguments

query	An IRanges or GRanges query object.
subject	An IRanges/GRanges object or a fast_ranges_index. Use fast_build_index(subject) when the same subject is reused across many overlap queries.
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap. Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes. Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with IRanges::findOverlaps() / GenomicRanges::findOverlaps().

min_overlap	<p>Integer scalar minimum overlap width, in bases.  <math>\emptyset</math> is the least strict setting.</p> <p>Larger values require wider shared overlap and therefore return fewer hits.</p> <p>This argument matters only when the selected type allows an actual overlap width to be measured.</p>
type	<p>Character scalar describing what "match" means.</p> <p>"any" matches any overlap that satisfies max_gap / min_overlap.</p> <p>"start" matches ranges with compatible start coordinates.</p> <p>"end" matches ranges with compatible end coordinates.</p> <p>"within" matches queries contained inside subjects.</p> <p>"equal" matches queries and subjects with the same interval, or with start/end differences no larger than max_gap when tolerance is allowed.</p>
ignore_strand	<p>Logical scalar controlling strand handling for genomic ranges.</p> <p>For GRanges, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules.</p> <p>TRUE means strand is ignored and only genomic coordinates are compared.</p> <p>For IRanges, this argument has no effect because there is no strand.</p>
threads	<p>Integer scalar number of worker threads to use.</p> <p>Use 1 for the most conservative behavior and easiest debugging.</p> <p>Use larger values on multicore machines when throughput matters.</p> <p>For repeated-query workloads, combine a prebuilt index from fast_build_index(subject) with a thread count chosen empirically on your hardware.</p> <p>fastRanges is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.</p>
deterministic	<p>Logical scalar controlling output order.</p> <p>TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts.</p> <p>FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.</p>
query_prefix	<p>Prefix applied to query columns.</p>

### Details

This is similar to a SQL SEMI JOIN.

It keeps only query rows that have at least one overlap hit.

overlap\_count tells you how many subject ranges matched each retained query row.

### Value

A data.frame containing matching query rows and overlap counts.

## Overlap semantics

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```

type = "any"
query  :  |-----|
subject: |-----|

type = "within"
subject: |-----|
query   :  |-----|

type = "start"
query   :  |-----|
subject: |-----|

type = "end"
query   :          |-----|
subject: |-----|

type = "equal"
query   :  |-----|
subject:  |-----|

gap / min_overlap controls
query  : |-----|      |-----| : subject
          < gap >

```

The middle distance is the gap. A hit is allowed when this distance is  $\leq \text{max\_gap}$  (for  $\text{max\_gap} \geq 0$ ), and overlap width is  $\geq \text{min\_overlap}$ .

Beginner-friendly interpretation:

type = "any" asks "do these ranges touch or overlap closely enough to count?"

type = "start" and type = "end" are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

type = "within" asks whether each query lies inside a subject interval.

type = "equal" asks whether query and subject describe the same interval, optionally with endpoint tolerance when  $\text{max\_gap} \geq 0$ .

This argument grammar is intentionally aligned with Bioconductor overlap APIs (IRanges / GenomicRanges).

## Examples

```

q <- IRanges::IRanges(start = c(1, 10, 20), width = c(5, 4, 3))
s <- IRanges::IRanges(start = c(3, 9, 18), width = c(4, 6, 5))
fast_semi_overlap_join(q, s)

```

---

fast\_tile\_coverage      *Tile-Based Coverage Summary*


---

**Description**

Aggregate coverage into fixed-width tiles.

**Usage**

```
fast_tile_coverage(
  x,
  tile_width,
  step_width = tile_width,
  shift = 0L,
  width = NULL,
  weight = 1L,
  method = c("auto", "sort", "hash"),
  threads = fast_default_threads()
)
```

**Arguments**

x	An IRanges or GRanges object.
tile_width	Integer scalar tile width.
step_width	Integer scalar step width. Use a value smaller than tile_width for overlapping tiles and the same value for adjacent tiles.
shift	Passed to coverage().
width	Passed to coverage().
weight	Passed to coverage().
method	Coverage method.
threads	Integer scalar thread count. Reserved for API consistency.

**Details**

fast\_tile\_coverage() converts base-resolution coverage into a simpler table of fixed-width summaries.

Each row in the result is one tile.

coverage\_sum is the sum of coverage values across that tile.

For GRanges, the output also includes seqnames.

**Value**

A data.frame with tile coordinates and coverage\_sum.

**Examples**

```
x <- IRanges::IRanges(start = c(1, 4, 10), end = c(5, 8, 12))
fast_tile_coverage(x, tile_width = 5L)
```

---

```
fast_window_count_overlaps
      Windowed Overlap Counts
```

---

**Description**

Count overlaps in sliding windows across the coordinate span of query.

**Usage**

```
fast_window_count_overlaps(
  query,
  subject,
  window_width,
  step_width = window_width,
  max_gap = -1L,
  min_overlap = 0L,
  type = c("any", "start", "end", "within", "equal"),
  ignore_strand = FALSE,
  threads = fast_default_threads(),
  deterministic = TRUE
)
```

**Arguments**

query	An IRanges or GRanges query object. Windows are generated from $\min(\text{start}(\text{query}))$ to $\max(\text{end}(\text{query}))$ , separately by chromosome for GRanges.
subject	An IRanges or GRanges subject object.
window_width	Integer scalar window width.
step_width	Integer scalar window step. Use a value smaller than window_width for sliding windows and the same value for non-overlapping windows.
max_gap	Integer scalar controlling how far apart two ranges may be and still count as a hit. Use -1 to require a true overlap. Use 0 to allow touching ranges for "any" and to keep Bioconductor's default tolerance behavior for the other overlap modes. Use positive values when you want "nearby" ranges to count as matches even if they do not overlap directly. Units are bases. The meaning is intentionally aligned with IRanges::findOverlaps() / GenomicRanges::findOverlaps().

min_overlap	<p>Integer scalar minimum overlap width, in bases.  <math>\emptyset</math> is the least strict setting.  Larger values require wider shared overlap and therefore return fewer hits.  This argument matters only when the selected type allows an actual overlap width to be measured.</p>
type	<p>Character scalar describing what "match" means.  "any" matches any overlap that satisfies max_gap / min_overlap.  "start" matches ranges with compatible start coordinates.  "end" matches ranges with compatible end coordinates.  "within" matches queries contained inside subjects.  "equal" matches queries and subjects with the same interval, or with start/end differences no larger than max_gap when tolerance is allowed.</p>
ignore_strand	<p>Logical scalar controlling strand handling for genomic ranges.  For GRanges, FALSE means "+", "-", and "*" are interpreted using standard Bioconductor strand rules.  TRUE means strand is ignored and only genomic coordinates are compared.  For IRanges, this argument has no effect because there is no strand.</p>
threads	<p>Integer scalar number of worker threads to use.  Use 1 for the most conservative behavior and easiest debugging.  Use larger values on multicore machines when throughput matters.  For repeated-query workloads, combine a prebuilt index from fast_build_index(subject) with a thread count chosen empirically on your hardware.  fastRanges is optimized for large and throughput-oriented workloads. For one-off or small jobs, Bioconductor's native overlap routines may be competitive.</p>
deterministic	<p>Logical scalar controlling output order.  TRUE returns a stable order, which is useful for testing, reproducible reports, and direct comparison across thread counts.  FALSE allows the implementation to return hits in an unspecified order, which can be noticeably faster for large multithreaded jobs because it avoids extra global ordering work.</p>

### Details

Windows are generated across the span of query, then overlap counts are measured between those windows and subject.

The result is a table rather than a Hits object because the main goal is summary, not individual hit inspection.

### Value

A data.frame containing window coordinates and overlap counts.

**Overlap semantics**

query is the range set you ask about. subject is the range set you compare it against.

Core interval semantics (ASCII schematic):

```
type = "any"
query  :  |-----|
subject: |-----|
```

```
type = "within"
subject: |-----|
query  :  |-----|
```

```
type = "start"
query  :  |-----|
subject: |-----|
```

```
type = "end"
query  :  |-----|
subject: |-----|
```

```
type = "equal"
query  :  |-----|
subject: |-----|
```

```
gap / min_overlap controls
query  : |-----|      |-----| : subject
          < gap >
```

The middle distance is the gap. A hit is allowed when this distance is  $\leq$  max\_gap (for max\_gap  $\geq$  0), and overlap width is  $\geq$  min\_overlap.

Beginner-friendly interpretation:

type = "any" asks "do these ranges touch or overlap closely enough to count?"

type = "start" and type = "end" are useful when interval boundaries are biologically meaningful, for example transcription start or end sites.

type = "within" asks whether each query lies inside a subject interval.

type = "equal" asks whether query and subject describe the same interval, optionally with endpoint tolerance when max\_gap  $\geq$  0.

This argument grammar is intentionally aligned with Bioconductor overlap APIs (IRanges / GenomicRanges).

**Examples**

```
q <- IRanges::IRanges(start = c(1, 12), end = c(10, 20))
s <- IRanges::IRanges(start = c(2, 5, 15), end = c(3, 6, 16))
fast_window_count_overlaps(q, s, window_width = 5L, step_width = 5L)
```

# Index

## \* datasets

- fast\_ranges\_example, [46](#)
  
- fast\_anti\_overlap\_join, [3](#)
- fast\_build\_index, [5](#)
- fast\_cluster\_overlaps, [6](#)
- fast\_count\_overlaps, [9](#)
- fast\_count\_overlaps\_by\_group, [12](#)
- fast\_coverage, [15](#)
- fast\_default\_threads, [16](#)
- fast\_disjoin, [16](#)
- fast\_distance\_to\_nearest, [17](#)
- fast\_find\_overlaps, [18](#)
- fast\_find\_overlaps\_iter, [21](#)
- fast\_follow, [24](#)
- fast\_gaps, [25](#)
- fast\_index\_stats, [26](#)
- fast\_inner\_overlap\_join, [27](#)
- fast\_iter\_collect, [29](#)
- fast\_iter\_has\_next, [30](#)
- fast\_iter\_next, [31](#)
- fast\_iter\_reset, [31](#)
- fast\_left\_overlap\_join, [32](#)
- fast\_load\_index, [35](#)
- fast\_nearest, [35](#)
- fast\_overlap\_aggregate, [39](#)
- fast\_overlap\_join, [42](#)
- fast\_overlaps\_any, [36](#)
- fast\_precede, [45](#)
- fast\_range\_intersect, [47](#)
- fast\_range\_setdiff, [47](#)
- fast\_range\_union, [48](#)
- fast\_ranges\_example, [46](#)
- fast\_reduce, [49](#)
- fast\_save\_index, [49](#)
- fast\_self\_overlaps, [50](#)
- fast\_semi\_overlap\_join, [53](#)
- fast\_tile\_coverage, [56](#)
- fast\_window\_count\_overlaps, [57](#)