Package 'genomeIntervals'

November 2, 2025

Version 1.67.0

Date 2025-07-22

Type Package

Title Operations on genomic intervals

Author Julien Gagneur <gagneur@in.tum.de>, Joern Toedling, Richard Bourgon, Nicolas Delhomme

Maintainer Julien Gagneur <gagneur@in.tum.de>

Depends R (>= 2.15.0), methods, intervals (>= 0.14.0), BiocGenerics (>= 0.15.2)

Imports Seqinfo, GenomicRanges (>= 1.21.16), IRanges(>= 2.3.14), S4Vectors (>= 0.7.10)

Description This package defines classes for representing genomic intervals and provides functions and methods for working with these. Note: The package provides the basic infrastructure for and is enhanced by the package 'girafe'.

License Artistic-2.0

Collate AllGenerics.R Genome_intervals-class.R Genome_intervals-coercion-methods.R Genome_intervals-package.R Genome_intervals-ordering.R show-methods.R size-methods.R c.R core_annotated.R distance_to_nearest-methods.R interval_complement-methods.R interval_overlap-methods.R interval_intersection-methods.R interval_union-methods.R which_nearest-methods.R parseGffAttributes.R readGff3.R writeGff3.R

biocViews DataImport, Infrastructure, Genetics

LazyLoad yes

RoxygenNote 7.1.0

Encoding UTF-8

git_url https://git.bioconductor.org/packages/genomeIntervals

git_branch devel

git_last_commit 8406dc4

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-11-02

Contents

`genomeIntervals-package`

Operations on genomic intervals

Description

Tools for operation on genomic intervals.

Details

Package:	genomeIntervals
Version:	1.25.3
Date:	2015-07-15
Type:	Package
Depends:	R (>= 2.15.0), intervals (>= 0.14.0), BiocGenerics, methods
Imports:	Seqinfo
Suggests:	
License:	Artistic 2.0
BiocViews:	DataImport, Infrastructure, Genetics
LazyLoad:	yes

Index:

Author(s)

Julien Gagneur <gagneur@embl.de>, Richard Bourgon, Joern Toedling, Nicolas Delhomme.

Maintainer: Julien Gagneur <gagneur@embl.de>

See Also

intervals

---

c *c extension for the* genomeIntervals package

Description

This function combines several genome intervals (stranded or not) objects into a single one.

Usage

```
## S4 method for signature 'Genome_intervals'
c(x, ..., recursive = FALSE)
```

Arguments

x	a Genome_intervals or Genome_intervals_stranded object - not mandatory.
...	two (one if x is defined) or more Genome_intervals or Genome_intervals_stranded objects.
recursive	inherited from the base c function definition and not used.

Details

If the arguments have mixed classes (both Genome_intervals or Genome_intervals_stranded), then they are coerced to Genome_intervals before combination. Otherwise, the common class is used. Ifa list is provided with NULL entries, these are discarded. If a vector of object is provided with non genomeIntervals classes, then a list, ordered as the input vector, is returned.

Value

- A single Genome_intervals or Genome_intervals_stranded object. Input objects are combined in their order of appearance in the the argument list.
- If any input argument is not a Genome_intervals, list(...) is returned instead.

##'

Examples

```
##' load toy examples
data("gen_ints")

##' combine i and j returns a Genome_intervals_stranded object
c( i, j )

##' combine a not-stranded and a stranded returns a not-stranded object
c( as(i, "Genome_intervals"), j )
```

---

core_annotated *Genome intervals with minimal annotation*

Description

returns a copy of the input (stranded) genome intervals object with annotations restricted to the minimally required ones.

Usage

```
core_annotated(x)
```

Arguments

x	A Genome_intervals or Genome_intervals_stranded object.

Value

A copy of x with the annotation slot restricted to seq_name, inter_base and strand (the latter only if x is a Genome_intervals_stranded object).

Examples

```
# load toy examples
data("gen_ints")

# add some non-core annotations to i
annotation(i)$comment = "some non-core annotation"

# i with all annotations
i

# core annotations only
core_annotated(i)

## Not run:
# with different annotation columns, i and j cannot be combined
c( i, j )

## End(Not run)

# core annotated versions can
c( core_annotated(i), core_annotated(j) )
```

`distance_to_nearest` *Distance in bases to the closest interval(s)*

Description

Given two objects, `from` and `to`, compute the distance in bases of each `from` interval to the nearest `to` interval(s). The distance between a base and the next inter-bases on either side values 0.5. Thus, base - base and inter-base - inter-base intervals distances are integer, whereas base - inter-base intervals distances are half-integers.

Usage

```
## S4 method for signature 'Genome_intervals,Genome_intervals'
distance_to_nearest(from, to)
## S4 method for signature
## 'Genome_intervals_stranded,Genome_intervals_stranded'
distance_to_nearest(from, to)
```

Arguments

`from`	A `Genome_intervals` or `Genome_intervals_stranded` object.
`to`	A `Genome_intervals` or `Genome_intervals` object.

Details

A wrapper calling intervals::distance_to_nearest by seqnames and by strand (if both from and to are Genome_intervals_stranded objects). Thus, if both are stranded, distances are computed over each strand separately. One object must be coerced to Genome_intervals if this is not wished.

Value

A numeric vector of distances with one element for each row of from.

See Also

intervals::distance_to_nearest

Examples

```
## load toy examples
data(gen_ints)

## i in close_intervals notation
close_intervals(i)

## j in close_intervals notation
close_intervals(j)

## distances from i to j
dn = distance_to_nearest(i,j)
dn

## distance == 0 if and only if the interval overlaps another one:
io = interval_overlap(i,j)
if( any( ( sapply(io, length) >0 )  != (!is.na(dn) & dn ==0) ) )
 stop("The property 'distance == 0 if and only if the interval overlaps another one' is not followed for at least one

## distances without strand-specificity
distance_to_nearest(
   as(i,"Genome_intervals"),
   as(j,"Genome_intervals")
)
```

GenomeIntervals *Constructor function for genomeIntervals objects*

Description

A user-friendly constructor function for creating both Genome_intervals and Genome_intervals_stranded objects.

Usage

```
GenomeIntervals(chromosome, start, end, strand = NULL,
                inter.base = NULL, leftOpen = NULL,
                rightOpen = NULL, ...)
```

Arguments

chromosome	character vector of chromosome names of the intervals; will become the seqnames of the resulting object
start	numeric or integer; start (left-most) coordinate of the intervals
end	numeric or integer; end (right-most) coordinate of the intervals
strand	chacter; specifies which strand the intervals are located on; if specified an object of class Genome_intervals_stranded is created; if NULL an object of class Genome_intervals is created
inter.base	logical; if TRUE an interval is located between the specified coordinates, instead of spanning them; useful for restriction-enzym cutting sites, for example.
leftOpen	logical; if TRUE an interval is left-open; if NULL all intervals are assumed to be left-closed.
rightOpen	logical; if TRUE an interval is right-open; if NULL all intervals are assumed to be right-closed.
...	any additional annotation for supplied intervals

Details

The arguments chromosome, start, and end need to be of the same length, with the first element of each vector corresponding to the first interval, the second element to the second interval, and so on.

The same applies to strand, inter.base, leftOpen, rightOpen and any additional vectors in '...', if they are specified.

Value

An object of class Genome_intervals or Genome_intervals_stranded depending on whether strand has been specified.

Author(s)

J. Toedling

See Also

Genome_intervals-class, Genome_intervals_stranded-class

Examples

```
## constructing a Genome_intervals object
G <- GenomeIntervals(start=c(1,3,4,5,8,10), end=c(5,5,6,8,9,11),
                     chromosome=rep(c("chr2","chrX","chr1"), each=2),
                     leftOpen=rep(c(FALSE, FALSE, TRUE), 2))
show(G)

## constructing a Genome_intervals_stranded object with
##  additional interval annotation
GS <- GenomeIntervals(start=c(1,3,4,5,8,10), end=c(5,5,6,8,9,11),
                      chromosome=rep(c("chr2","chrX","chr1"), each=2),
                      strand=c("-","-","+","+","+","+"),
                      GC.content=round(runif(6), digits=2))
show(GS)
```

---

genomeIntervals coercion methods

Coercion methods of the genomeIntervals package

Description

coerce This method allows to coerce a genomeIntervals object into a GRangesList object.

Usage

```
## S4 method for signature 'Genome_intervals'
as(from,to)
```

Arguments

from	An object of class Genome_intervals
to	a character string: GRanges or GRangesList

Value

coerce A GRanges or GRangesList containing the result of the coercion.

Author(s)

Nicolas Delhomme

See Also

- genomeIntervals object
- readGff3 function

Examples

```
## Not run:
annot<-readGff3(system.file("extdata",
                            "Dmel-mRNA-exon-r5.52.gff3",
                            package="RnaSeqTutorial")
gAnnot<-as(annot,"GRangesList") type(annot)

## End(Not run)
```

Genome_intervals deprecated functions

The following function have been deprecated:

- seq_name
- seq_name<-

Description

- The seq_name and seq_name<- accessor functions have been replaced by the more generic seqnames and seqnames<- accessor functions, respectively.

Genome_intervals-class

Class "Genome_intervals"

Description

A set of genomic intervals without specified strand. Genomic intervals are intervals over the integers with two further annotations: seqnames (a chromosome or more generally a sequence of origin) and inter_base (logical) that states whether the interval is to be understood as an interval over bases (such as coding-sequence) or inter-bases (such as restriction sites or insertion positions).

Slots

.Data: See Intervals_full

annotation: A "data.frame" with the same number of rows as .Data. It has a column named seq_name that is a factor and does not contain missing values. seq_name is used to represent the chromosome or more generally the sequence of origin of the intervals. annotation has a column named inter_base that is logical and does not contain missing values. inter_base is FALSE if the interval is to be understood as an interval over bases (such as coding-sequence) and TRUE if it is over inter-bases (such as restriction site or an insertion position). Like base intervals, inter-base interval are encoded over the integers. An inter-base at position n indicates the space between base n and n+1.

closed: See Intervals_full

type: See Intervals_full

Extends

Class "Intervals_full", directly. Class "Intervals_virtual", by class "Intervals_full", distance 2. Class "matrix", by class "Intervals_full", distance 3. Class "array", by class "Intervals_full", distance 4. Class "structure", by class "Intervals_full", distance 5. Class "vector", by class "Intervals_full", distance 6, with explicit coerce.

Methods

[signature(x = "Genome_intervals"): ...

[[signature(x = "Genome_intervals"): ...

[[<- signature(x = "Genome_intervals"): ...

\$ signature(x = "Genome_intervals"): ...

\$<- signature(x = "Genome_intervals"): ...

annotation signature(object = "Genome_intervals"): ...

annotation<- signature(object = "Genome_intervals"): ...

coerce signature(from = "Genome_intervals", to = "Intervals_full"): ...

coerce signature(from = "Genome_intervals", to = "character"): ...

coerce signature(from = "Genome_intervals", to = "data.frame"): ...

distance_to_nearest signature(from = "Genome_intervals", to = "Genome_intervals"): ...

inter_base signature(x = "Genome_intervals"): ...

inter_base<- signature(x = "Genome_intervals"): ...

interval_complement signature(x = "Genome_intervals"): ...

interval_intersection signature(x = "Genome_intervals"): ...

interval_overlap signature(from = "Genome_intervals", to = "Genome_intervals"): ...

interval_union signature(x = "Genome_intervals"): ...

seqnames signature(x = "Genome_intervals"): ...

seqnames<- signature(x = "Genome_intervals"): ...

size signature(x = "Genome_intervals"): ...

type<- signature(x = "Genome_intervals"): ...

which_nearest For each interval in Set1, finds nearest (least distant) interval in Set2. Intervals on different chromsomes are never considered 'near' to each other. The returned value is a data.frame with the number of rows equal to the number of intervals in Set1. Each row specifies the distance to the nearest interval in Set2 (a 0 means that the interval overlaps one or more intervals in Set2), and the indices of near and overlapping intervals in Set2. See Intervals_full for further details.

width Returns the interval length as the number of bp covered (base interval) or spanned(inter-base interval). Similar to the IRanges package widthfunction.

Note

A Genome_intervals is a "Intervals_full" of type Z (i.e. a set of intervals over the integers). The annotation slot can carry further columns that can serve as annotations.

See Also

Genome_intervals_stranded for a derived class that allows stranded genomic intervals.

Examples

```
# The "Genome_intervals" class

i <- new(
  "Genome_intervals",
  matrix(
   c(1,2,
     3,5,
     4,6,
     8,9
     ),
   byrow = TRUE,
                ncol = 2
  ),
  closed = matrix(
     c(
      TRUE, FALSE,
      TRUE, FALSE,
      TRUE, TRUE,
      TRUE, FALSE
      ),
     byrow = TRUE,
        ncol = 2
        ),
  annotation = data.frame(
      seq_name = factor(c("chr01","chr01", "chr02","chr02")),
      inter_base = c(FALSE, FALSE, TRUE, TRUE)
      )
  )

colnames(i) <- c( "start", "end" )

# print
print(i)

# size (number of bases per interval)
size(i)

## convert to a data.frame
as(i,"data.frame")

## simpler way to construct a Genome_intervals object:
G <- GenomeIntervals(start=c(1,3,4,5,10,8), end=c(5,5,6,8,11,9),
                     chromosome=rep(c("chr2","chrX","chr1"), each=2),
                     leftOpen=rep(c(FALSE, FALSE, TRUE), 2))
show(G)
```

Genome_intervals-ordering

Ordering methods for Genome intervals

Description

An order is defined on genome intervals and stranded genome intervals to allow sort(), order() and rank().

Usage

```
## S4 method for signature 'Genome_intervals'
order(..., na.last=TRUE, decreasing=FALSE)
## S4 method for signature 'Genome_intervals_stranded'
order(..., na.last=TRUE, decreasing=FALSE)

## S4 method for signature 'Genome_intervals'
sort(x, decreasing=FALSE, ...)

## S4 method for signature 'Genome_intervals'
rank(x, na.last=TRUE, ties.method=c("average", "first", "last", "random", "max", "min"), ...)
## S4 method for signature 'Genome_intervals'
xtfrm(x)
```

Arguments

x	Objects of class Genome_intervals or Genome_intervals_stranded.
...	Objects of class Genome_intervals, Genome_intervals_stranded or of any other class for order.
na.last	Ignored for ordering Genome_intervals and Genome_intervals_stranded objects
decreasing	TRUE or FALSE.
ties.method	A character string specifying how ties are treated. Only "first" is supported.

Details

An order on Genome_intervals entries is defined by sorting by 1. seqnames 2. start, where closed start & not inter-base < closed start & inter-base < open start & not inter-base < open start & inter-base 3. stop, where open stop & not inter-base < open stop & inter-base < closed stop & not inter-base < closed stop & inter-base 4. strand (for Genome_intervals_stranded object)

The factors seqnames and strand are sorted according to their levels (default R behavior).

The primitive is implemented in xtfrm which is then called by the other methods. Hence, the order, sort and rank methods are consistent.

`order(..., na.last=TRUE, decreasing=TRUE)`: return a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments. See `order` in the base package for more details. `na.last` is ignored for `Genome_intervals` objects.

`rank(x, na.last=TRUE, ties.method=c("average", "first", "last", "random", "max", "min"), ...)`: Return the sample ranks of the (stranded) genome intervals in `x`. See `rank` in the base package for more details.

`sort(x)`: Sort `x`. See `sort` in the base package for more details.

`xtfrm(x)`: Auxiliary function that produces a numeric vector which will sort in the same order as 'x' `x`. See `xtfrm` in the base package for more details. Workhorse for the other methods

See Also

`Genome_intervals` `Genome_intervals_stranded` `order`, `sort`, `rank`, `xtfrm`

Examples

```
## an example with ties
gi = GenomeIntervals(c("chr2", "chr2", "chr1", "chr1"), c(1,1,10,10), c(5,3,12,12) )

sort(gi)
rank(gi)
order(gi)

## Define order on seqnames at your convenience
## by specifying the order of the levels
## compare:
gi = GenomeIntervals(
  c("chr2", "chr2", "chr10", "chr10"),
  c(1,1,10,10),
  c(5,3,12,12)
  )
sort(gi)

## with:
gi2 = GenomeIntervals(
 factor(c("chr2", "chr2", "chr10", "chr10"), levels=c("chr2", "chr10")),
 c(1,1,10,10),
 c(5,3,12,12)
 )
sort(gi2)
```

Genome_intervals_stranded-class

Class "Genome_intervals_stranded"

Description

A set of genomic intervals with a specified strand.

Slots

`.Data`: See `Genome_intervals`

`annotation`: A `data.frame` (see `Genome_intervals` for basic requirements). The `annotation` moreover has a `strand` column that is a factor with exactly two levels(typically `"+"` and `"-"`).

`closed`: See `Genome_intervals`

`type`: See `Genome_intervals`

Extends

Class `"Genome_intervals"`, directly. Class `"Intervals_full"`, by class "Genome_intervals", distance 2. Class `"Intervals_virtual"`, by class "Genome_intervals", distance 3. Class `"matrix"`, by class "Genome_intervals", distance 4. Class `"array"`, by class "Genome_intervals", distance 5. Class `"structure"`, by class "Genome_intervals", distance 6. Class `"vector"`, by class "Genome_intervals", distance 7, with explicit coerce.

Methods

coerce `signature(from = "Genome_intervals_stranded", to = "character")`: ...

distance_to_nearest `signature(from = "Genome_intervals_stranded", to = "Genome_intervals_stranded")`: ...

interval_complement `signature(x = "Genome_intervals_stranded")`: ...

interval_intersection `signature(x = "Genome_intervals_stranded")`: ...

interval_overlap `signature(to = "Genome_intervals_stranded", from = "Genome_intervals_stranded")`: ...

interval_union `signature(x = "Genome_intervals_stranded")`: ...

strand `signature(x = "Genome_intervals_stranded")`: ...

strand<- `signature(x = "Genome_intervals_stranded")`: ...

See Also

`Genome_intervals` the parent class without strand.

Examples

```
# The "Genome_intervals_stranded" class
j <- new(
  "Genome_intervals_stranded",
  matrix(
   c(1,2,
     3,5,
     4,6,
     8,9
     ),
   byrow = TRUE,
                ncol = 2
  ),
  closed = matrix(
```

```
        c(
         FALSE, FALSE,
         TRUE, FALSE,
         TRUE, TRUE,
         TRUE, FALSE
         ),
        byrow = TRUE,
           ncol = 2
           ),
        annotation = data.frame(
            seq_name = factor( c("chr01","chr01", "chr02","chr02") ),
         strand = factor( c("+", "+", "+", "-") ),
         inter_base = c(FALSE,FALSE,FALSE,TRUE)
         )
      )

    ## print
    print(j)

    ## size of each interval as count of included bases
    size(j)

    ## close intervals left and right (canonical representation)
    close_intervals(j)


    ## simpler way to construct a Genome_intervals_stranded object
    GS <- GenomeIntervals(start=c(1,3,4,5,8,10), end=c(5,5,6,8,9,11),
                          chromosome=rep(c("chr2","chrX","chr1"), each=2),
                          strand=c("-","-","+","+","+","+") )
    show(GS)
```

---

gen_ints *Genome Intervals examples*

Description

Toy examples for testing functions and running examples of the package genomeIntervals.

Usage

```
data(gen_ints)
```

Format

Two Genome_intervals_stranded objects, i and j, without inter-base intervals and a third one, k, with.

getGffAttribute *Pull one or more key/value pairs from gffAttributes strings*

Description

GFF files contain a string, with key/value pairs separated by ";", and the key and value separated by "=". This function quickly extracts one or more key/value pairs.

Usage

```
getGffAttribute(gi, attribute)
```

Arguments

gi	A Genome_intervals object.
attribute	A vector of key names.

Value

A matrix with the same number of rows as gi, and one column per element of attribute.

See Also

See parseGffAttributes for more complete parsing. See the function readGff3 for loading a GFF file.

Examples

```
# Get file path
libPath <- installed.packages()["genomeIntervals", "LibPath"]
filePath <- file.path(
                  libPath,
                  "genomeIntervals",
                  "example_files"
                  )

# Load gff
gff <- readGff3( file.path( filePath, "sgd_simple.gff"), isRightOpen=FALSE)

## head of full gff annotations
head(annotation(gff))

# extract ID and Parent attributes
idpa = getGffAttribute( gff, c( "ID", "Parent" ) )

head(idpa)
```

interval_overlap *Assess overlap from one set of genomic intervals to another*

Description

Given two objects, a 'from' and a 'to', assess which intervals in 'to' overlap which of 'from'.

Usage

```
## S4 method for signature 'Genome_intervals,Genome_intervals'
interval_overlap(
                 from, to,
                 check_valid = TRUE
                 )
## S4 method for signature
## 'Genome_intervals_stranded,Genome_intervals_stranded'
interval_overlap(
                 from, to,
                 check_valid = TRUE
                 )
```

Arguments

from	A Genome_intervals or Genome_intervals_stranded object.
to	A Genome_intervals or Genome_intervals_stranded object.
check_valid	Should validObject be called before passing to compiled code?

Details

A wrapper calling intervals:interval_overlap by seq_name and by strand (if both to and from are "Genome_intervals_stranded" objects).

Value

A list, with one element for each row of from. The elements are vectors of indices, indicating which to rows overlap each from. A list element of length 0 indicates a from with no overlapping to intervals.

Examples

```
data(gen_ints)
# i as entered
i

# i in close_intervals notation
close_intervals(i)
```

```
# j in close_intervals notation
close_intervals(j)

# list of intervals of j overlapping intervals of i
interval_overlap(i,j)
```

`interval_union` *Genome interval set operations*

Description

Compute interval set operations on "Genome_intervals" or "Genome_intervals_stranded" objects.

Usage

```
## S4 method for signature 'Genome_intervals'
interval_union(x, ...)
## S4 method for signature 'Genome_intervals_stranded'
interval_union(x, ...)

## S4 method for signature 'Genome_intervals'
interval_complement(x)
## S4 method for signature 'Genome_intervals_stranded'
interval_complement(x)

## S4 method for signature 'Genome_intervals'
interval_intersection(x,...)
## S4 method for signature 'Genome_intervals_stranded'
interval_intersection(x,...)
```

Arguments

x	A "Genome_intervals" or "Genome_intervals_stranded" object.
...	Optionally, additional objects of the same class as x.

Details

Wrappers calling the corresponding functions of the package intervals by same seq_name, inter_base and if needed strand. Note that the union of single input object x returns the reduced form of x, i.e. the interval representation of the covered set.

Value

A single object of appropriate class, representing the union, complement or intersection of intervals computed over entries with same seq_name, inter_base and also strand if all passed objects are of the class "Genome_intervals_stranded".

See Also

interval_union, interval_complement, interval_intersection and reduce from the package intervals.

Examples

```
## load toy examples
data(gen_ints)
## content of i object
i

## complement
interval_complement(i)

## reduced form (non-overlapping interval representation of the covered set)
interval_union(i)

## union
interval_union(i[1:2,], i[1:4,])

# map to genome intervals and union again
i.nostrand = as(i,"Genome_intervals")
interval_union(i.nostrand)

## intersection with a second object
# print i and j in closed interval notation
close_intervals(i)
close_intervals(j)

# interval_intersection
interval_intersection(i,j)

#interval intersection non-stranded
interval_intersection(i.nostrand, as(j, "Genome_intervals"))
```

---

parseGffAttributes *Parse out the gffAttributes column of a Genome_intervals object*

Description

GFF files contain a string, with key/value pairs separated by ";", and the key and value separated by "=". This function parses such strings into a list of vectors with named elements.

Usage

```
parseGffAttributes(gi)
```

Arguments

gi	A Genome_intervals object.

Value

A list, with one element per row of `gi`. Each element is a character vector with named components. Names correspond to keys, and components correspond to values.

Note

Key/value pairs which are missing the "=" symbol, or which have nothing between it and the ";" delimiter or end of line, will generate a `NA` value, with a warning. Any key/value "pairs" with more than one "=" cause an error.

See Also

In many cases, `getGffAttribute`, in this package, is easier and faster. See the function `readGff3` for loading a GFF file.

Examples

```
# Get file path
libPath <- installed.packages()["genomeIntervals", "LibPath"]
filePath <- file.path(
                  libPath,
                  "genomeIntervals",
                  "example_files"
                  )

# Load gff and parse attributes
gff <- readGff3( file.path( filePath, "sgd_simple.gff"), isRightOpen = FALSE )
gfatt <- parseGffAttributes(gff)

head( gfatt )
```

---

readGff3,character-method

readGff3

Description

Read (write) a Genome_intervals_stranded object from (to) a GFF3 file

Usage

```
readGff3(file, isRightOpen=FALSE, quiet=FALSE)
readBasePairFeaturesGff3(file, quiet=FALSE)
readZeroLengthFeaturesGff3(file, quiet=FALSE)
writeGff3(object, file)
```

Arguments

file	The name of the gff file to read/write.
isRightOpen	Although it is arguable that a GFF3 file might have a right-open intervals convention - the format description being at best imprecise - most GFF3 file follow a right-closed convention. Hence, as of version 1.25.1, the default has been changed to isRightOpen = FALSE. See the details section on how to restore the older behaviour.
quiet	a boolean to turn verbosity off when reading a Gff3 file
object	a Genome_intervals object

Details

- readGff3 Make a Genome_intervals_stranded object from a gff file in gff3 format.
- readBasePairFeaturesGff3 Same as readGff3 assuming isRightOpen='FALSE', i.e. no zero length intervals are created. This is the default behaviour since v1.25.1.
- readZeroLengthFeaturesGff3 Same as readGff3 assuming isRightOpen='TRUE', i.e. zero length intervals are created when a feature's start is the same as its end. This was the default prior to version 1.25.1.
- writeGff3 Write a Genome_intervals object to a gff file in gff3 format.

The file must follow gff3 format specifications as in http://www.sequenceontology.org/gff3.shtml. Due to the imprecise definition and to allow for zero-length features, the default for reading a Gff3 file has been to assume right open intervals (until v1.25.1). As by then, the community consensus has been to use closed intervals, the default behaviour of readGff3 has been changed accordingly. The readGff3 file is now a wrapper that dispatches to two sub functions - which may be used directly - readBasePairFeaturesGff3 and readZeroLengthFeaturesGff3. The former assumes closed intervals and hence does not create zero-length intervals. The latter does the opposite and uses right-open intervals!

Some more noteworthy details:

The file is read as a table and meta-information (lines starting with ###) are not parsed.

A "." in, for example, the gff file's *score* or *frame* field will be converted to NA.

When the GFF file follows the right-open interval convention (isRightOpen is TRUE), then GFF entries for which end base equals first base are recognized as zero-length features and loaded as inter_base intervals.

Strand entries in the file are expected to be '.', '?', '+' or '-'. The two first are mapped to NA.

It can be that readGff3 is able to construct a Genome_intervals_stranded object from the input file, although not valid. A warning message is then generated and the constructed object is returned to allow inspection of it.

Potential FASTA entries at the end of the file are ignored.

Value

- readGff3 and friendsA Genome_intervals_stranded object image of the gff file. The GFF3 fields seqid, source, type, score, strand, phase and attributes are stored in the annotation slot and renamed as seq_name, source, type, score, strand, phase and gffAttributes respectively.

- writeGff3It dispatches to `write.table` and hence returns similar values.

See Also

The functions `getGffAttribute` and `parseGffAttributes` for parsing GFF attributes.

Examples

```
# Get file path
libPath <- installed.packages()["genomeIntervals", "LibPath"]
filePath <- file.path(
 libPath,
 "genomeIntervals",
 "example_files"
)

# Load SGD gff
# SGD does not comply to the GFF3 right-open interval convention
gff <- readGff3( file.path( filePath, "sgd_simple.gff"), isRightOpen = FALSE)

head(gff,10)

head(annotation(gff),10)

## Not run:
## write the gff3 file
writeGff3(gff,file="sgd_simple.gff")

## End(Not run)
```

Index