Package 'GenomicFeatures'

October 13, 2022

Title Conveniently import and query gene models

Description A set of tools and methods for making and manipulating transcript centric annotations. With these tools the user can easily download the genomic locations of the transcripts, exons and cds of a given organism, from either the UCSC Genome Browser or a BioMart database (more sources will be supported in the future). This information is then stored in a local database that keeps track of the relationship between transcripts, exons, cds and genes. Flexible methods are provided for extracting the desired features in a convenient format.

biocViews Genetics, Infrastructure, Annotation, Sequencing, GenomeAnnotation

URL https://bioconductor.org/packages/GenomicFeatures

BugReports https://github.com/Bioconductor/GenomicFeatures/issues

Version 1.48.4

License Artistic-2.0

Encoding UTF-8

Author M. Carlson, H. Pagès, P. Aboyoun, S. Falcon, M. Morgan, D. Sarkar, M. Lawrence, V. Obenchain

Maintainer Bioconductor Package Maintainer <maintainer@bioconductor.org>

Depends R (>= 3.5.0), BiocGenerics (>= 0.1.0), S4Vectors (>= 0.17.29), IRanges (>= 2.13.23), GenomeInfoDb (>= 1.25.7), GenomicRanges (>= 1.31.17), AnnotationDbi (>= 1.41.4)

Imports methods, utils, stats, tools, DBI, RSQLite (>= 2.0), RCurl, XVector (>= 0.19.7), Biostrings (>= 2.47.6), BiocIO, rtracklayer (>= 1.51.5), biomaRt (>= 2.17.1), Biobase (>= 2.15.1)

Suggests RMariaDB, org.Mm.eg.db, org.Hs.eg.db, BSgenome, BSgenome.Hsapiens.UCSC.hg19 (>= 1.3.17), BSgenome.Celegans.UCSC.ce11, BSgenome.Dmelanogaster.UCSC.dm3 (>= 1.3.17), mirbase.db, FDb.UCSC.tRNAs,

TxDb.Hsapiens.UCSC.hg19.knownGene,
TxDb.Celegans.UCSC.ce11.ensGene,
TxDb.Dmelanogaster.UCSC.dm3.ensGene (>= 2.7.1),
TxDb.Mmusculus.UCSC.mm10.knownGene (>= 3.4.7),
TxDb.Hsapiens.UCSC.hg19.lincRNAsTranscripts,
TxDb.Hsapiens.UCSC.hg38.knownGene (>= 3.4.6),
SNPlocs.Hsapiens.dbSNP144.GRCh38, Rsamtools, pasillaBamSubset
(>= 0.0.5), GenomicAlignments (>= 1.15.7), ensembldb,
AnnotationFilter, RUnit, BiocStyle, knitr

VignetteBuilder knitr

Collate utils.R TxDb-schema.R TxDb-SELECT-helpers.R UCSC-utils.R
Ensembl-utils.R findCompatibleMarts.R TxDb-class.R
FeatureDb-class.R makeTxDb.R makeTxDbFromUCSC.R
makeTxDbFromBiomart.R makeTxDbFromEnsembl.R
makeTxDbFromGRanges.R makeTxDbFromGFF.R makeFeatureDbFromUCSC.R
mapIdsToRanges.R id2name.R transcripts.R transcriptsBy.R
transcriptsByOverlaps.R transcriptLengths.R exonicParts.R
disjointExons.R extendExonsIntoIntrons.R features.R microRNAs.R
extractTranscriptSeqs.R extractUpstreamSeqs.R
getPromoterSeq-methods.R makeTxDbPackage.R select-methods.R
nearest-methods.R transcriptLocs2refLocs.R
coordinate-mapping-methods.R proteinToGenome.R
coverageByTranscript.R zzz.R

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

git_branch RELEASE_3_15

git_last_commit 06e37dc

git_last_commit_date 2022-09-19

Date/Publication 2022-10-13

R topics documented:

`as-format-methods` *Coerce to file format structures*

Description

These functions coerce a `TxDb` object to a `GRanges` object with metadata columns encoding transcript structures according to the model of a standard file format. Currently, BED and GFF models are supported. If a `TxDb` is passed to `export`, when targeting a BED or GFF file, this coercion occurs automatically.

Usage

```
## S4 method for signature 'TxDb'
asBED(x)
## S4 method for signature 'TxDb'
asGFF(x)
```

Arguments

`x`	A `TxDb` object to coerce to a `GRanges`, structured as BED or GFF.

Value

For `asBED`, a `GRanges`, with the columns `name`, `thickStart`, `thickEnd`, `blockStarts`, `blockSizes` added. The thick regions correspond to the CDS regions, and the blocks represent the exons. The transcript IDs are stored in the `name` column. The ranges are the transcript bounds.

For `asGFF`, a `GRanges`, with columns `type`, `Name`, `ID`,, and `Parent`. The gene structures are expressed according to the conventions defined by the GFF3 spec. There are elements of each `type`

of feature: "gene", "mRNA" "exon" and "cds". The Name column contains the gene_id for genes, tx_name for transcripts, and exons and cds regions are NA. The ID column uses gene_id and tx_id, with the prefixes "GeneID" and "TxID" to ensure uniqueness across types. The exons and cds regions have NA for ID. The Parent column contains the IDs of the parent features. A feature may have multiple parents (the column is a CharacterList). Each exon belongs to one or more mRNAs, and mRNAs belong to a gene.

Author(s)

Michael Lawrence

Examples

```
txdb_file <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                         package="GenomicFeatures")
txdb <- loadDb(txdb_file)

asBED(txdb)
asGFF(txdb)
```

---

coverageByTranscript *Compute coverage by transcript (or CDS) of a set of ranges*

Description

coverageByTranscript computes the transcript (or CDS) coverage of a set of ranges.

pcoverageByTranscript is a version of coverageByTranscript that operates element-wise.

Usage

```
coverageByTranscript(x, transcripts, ignore.strand=FALSE)

pcoverageByTranscript(x, transcripts, ignore.strand=FALSE, ...)
```

Arguments

x	An object representing a set of ranges (typically aligned reads). GRanges, GRangesList, GAlignments, GAlignmentPairs, and GAlignmentsList objects are supported. More generally, for coverageByTranscript x can be any object for which seqinfo() and coverage() are supported (e.g. a BamFile object). Note that, for such objects, coverage() is expected to return an RleList object whose names are seqlevels(x)). More generally, for pcoverageByTranscript x can be any object for which grglist() is supported. It should have the length of transcripts or length 1. If the latter, it is recycled to the length of transcripts.

`transcripts` A GRangesList object representing the exons of each transcript for which to compute coverage. For each transcript, the exons must be ordered by *ascending rank*, that is, by their position in the transcript. This means that, for a transcript located on the minus strand, the exons should typically be ordered by descending position on the reference genome. If `transcripts` was obtained with `exonsBy`, then the exons are guaranteed to be ordered by ascending rank. See `?exonsBy` for more information.

Alternatively, `transcripts` can be a TxDb object, or any TxDb-like object that supports the `exonsBy()` extractor (e.g. an EnsDb object). In this case it is replaced with the GRangesList object returned by `exonsBy(transcripts, by="tx", use.names=TRUE)`.

For `pcoverageByTranscript`, `transcripts` should have the length of `x` or length 1. If the latter, it is recycled to the length of `x`.

`ignore.strand` TRUE or FALSE. If FALSE (the default) then the strand of a range in `x` and exon in `transcripts` must be the same in order for the range to contribute coverage to the exon. If TRUE then the strand is ignored.

`...` Additional arguments passed to the internal call to `grglist()`. More precisely, when `x` is not a GRanges or GRangesList object, `pcoverageByTranscript` replace it with the GRangesList object returned by `grglist(x, ...)`.

Value

An RleList object *parallel* to `transcripts`, that is, the i-th element in it is an integer-Rle representing the coverage of the i-th transcript in `transcripts`. Its `lengths()` is guaranteed to be identical to `sum(width(transcripts))`. The names and metadata columns on `transcripts` are propagated to it.

Author(s)

Hervé Pagès

See Also

- `transcripts`, `transcriptsBy`, and `transcriptsByOverlaps`, for extracting genomic feature locations from a TxDb-like object.
- `transcriptLengths` for extracting the transcript lengths (and other metrics) from a TxDb object.
- `extractTranscriptSeqs` for extracting transcript (or CDS) sequences from chromosome sequences.
- The RleList class defined and documented in the **IRanges** package.
- The GRangesList class defined and documented in the **GenomicRanges** package.
- The `coverage` methods defined in the **GenomicRanges** package.
- The `exonsBy` function for extracting exon ranges grouped by transcript.
- `findCompatibleOverlaps` in the **GenomicAlignments** package for finding which reads are *compatible* with the splicing of which transcript.

Examples

```
## ---------------------------------------------------------------------
## 1. A SIMPLE ARTIFICIAL EXAMPLE WITH ONLY ONE TRANSCRIPT
## ---------------------------------------------------------------------

## Get some transcripts:
library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
txdb <- TxDb.Dmelanogaster.UCSC.dm3.ensGene
dm3_transcripts <- exonsBy(txdb, by="tx", use.names=TRUE)
dm3_transcripts

## Let's pick up the 1st transcript: FBtr0300689. It as 2 exons and 1
## intron:
my_transcript <- dm3_transcripts["FBtr0300689"]

## Let's create 3 artificial aligned reads. We represent them as a
## GRanges object of length 3 that contains the genomic positions of
## the 3 reads. Note that these reads are simple alignments i.e. each
## of them can be represented with a single range. This would not be
## the case if they were junction reads.
my_reads <- GRanges(c("chr2L:7531-7630",
                      "chr2L:8101-8200",
                      "chr2L:8141-8240"))

## The coverage of the 3 reads on the reference genome is:
coverage(my_reads)

## As you can see, all the genomic positions in the 3 ranges participate
## to the coverage. This can be confirmed by comparing:
sum(coverage(my_reads))
## with:
sum(width(my_reads))
## They should always be the same.

## When computing the coverage on a transcript, only the part of the
## read that overlaps with the transcript participates to the coverage.
## Let's look at the individual coverage of each read on transcript
## FBtr0300689:

## The 1st read is fully contained within the 1st exon:
coverageByTranscript(my_reads[1], my_transcript)

## Note that the length of the Rle (1880) is the length of the transcript.

## The 2nd and 3rd reads overlap the 2 exons and the intron. Only the
## parts that overlap the exons participate to coverage:
coverageByTranscript(my_reads[2], my_transcript)
coverageByTranscript(my_reads[3], my_transcript)

## The coverage of the 3 reads together is:
coverageByTranscript(my_reads, my_transcript)
```

```
## Note that this is the sum of the individual coverages. This can be
## checked with:
stopifnot(all(
  coverageByTranscript(my_reads, my_transcript)
  ==
  Reduce("+", lapply(seq_along(my_reads),
      function(i) coverageByTranscript(my_reads[i], my_transcript)), 0L)
))

## ---------------------------------------------------------------------
## 2. COMPUTE THE FULL TRANSCRIPTOME COVERAGE OF A SET OF ALIGNED READS
## ---------------------------------------------------------------------

## Load the aligned reads:
library(pasillaBamSubset)
library(GenomicAlignments)
reads <- readGAlignments(untreated1_chr4())

## Compute the full transcriptome coverage by calling
## coverageByTranscript() on 'dm3_transcripts':
tx_cvg <- coverageByTranscript(reads, dm3_transcripts, ignore.strand=TRUE)
tx_cvg

## A sanity check:
stopifnot(identical(lengths(tx_cvg), sum(width(dm3_transcripts))))

## We can also use pcoverageByTranscript() to compute 'tx_cvg'.
## For this we first create a GAlignmentsList object "parallel" to
## 'dm3_transcripts' where the i-th list element contains the aligned
## reads that overlap with the i-th transcript:
hits <- findOverlaps(reads, dm3_transcripts, ignore.strand=TRUE)
tx2reads <- setNames(as(t(hits), "List"), names(dm3_transcripts))
reads_by_tx <- extractList(reads, tx2reads)  # GAlignmentsList object
reads_by_tx

## Call pcoverageByTranscript():
tx_cvg2 <- pcoverageByTranscript(reads_by_tx, dm3_transcripts,
                                 ignore.strand=TRUE)
stopifnot(identical(tx_cvg, tx_cvg2))

## A more meaningful coverage is obtained by counting for each
## transcript only the reads that are *compatible* with its splicing:
compat_hits <- findCompatibleOverlaps(reads, dm3_transcripts)
tx2reads <- setNames(as(t(compat_hits), "List"), names(dm3_transcripts))
compat_reads_by_tx <- extractList(reads, tx2reads)

tx_compat_cvg <- pcoverageByTranscript(compat_reads_by_tx,
                                       dm3_transcripts,
                                       ignore.strand=TRUE)
## A sanity check:
stopifnot(all(all(tx_compat_cvg <= tx_cvg)))

## ---------------------------------------------------------------------
```

```
## 3. COMPUTE CDS COVERAGE OF A SET OF ALIGNED READS
## ---------------------------------------------------------------------

## coverageByTranscript() can also be used to compute CDS coverage:
cds <- cdsBy(txdb, by="tx", use.names=TRUE)
cds_cvg <- coverageByTranscript(reads, cds, ignore.strand=TRUE)
cds_cvg

## A sanity check:
stopifnot(identical(lengths(cds_cvg), sum(width(cds))))

## ---------------------------------------------------------------------
## 4. ALTERNATIVELY, THE CDS COVERAGE CAN BE OBTAINED FROM THE
##    TRANSCRIPT COVERAGE BY TRIMMING THE 5' AND 3' UTRS
## ---------------------------------------------------------------------

tx_lens <- transcriptLengths(txdb, with.utr5_len=TRUE, with.utr3_len=TRUE)
stopifnot(identical(tx_lens$tx_name, names(tx_cvg)))  # sanity

## Keep the rows in 'tx_lens' that correspond to a list element in
## 'cds_cvg' and put them in the same order as in 'cds_cvg':
m <- match(names(cds_cvg), names(tx_cvg))
tx_lens <- tx_lens[m, ]
utr5_width <- tx_lens$utr5_len
utr3_width <- tx_lens$utr3_len
cds_cvg2 <- windows(tx_cvg[m], start=1L+utr5_width, end=-1L-utr3_width)

## A sanity check:
stopifnot(identical(cds_cvg2, cds_cvg))
```

`disjointExons` *Extract non-overlapping exon parts from an object*

Description

`disjointExons` extracts the non-overlapping exon parts from a TxDb object or any other supported object.

WARNING: `disjointExons` is defunct in BioC 3.15. Please use `exonicParts` instead.

Usage

```
disjointExons(x, ...)

## S4 method for signature 'TxDb'
disjointExons(x, aggregateGenes=FALSE,
                 includeTranscripts=TRUE, ...)
```

Arguments

x	A TxDb object or any other supported object.
...	Arguments to be passed to methods.
aggregateGenes	For disjointExons : A logical. When FALSE (default) exon fragments that overlap multiple genes are dropped. When TRUE, all fragments are kept and the gene_id metadata column includes all gene ids that overlap the exon fragment.
includeTranscripts	For disjointExons : A logical. When TRUE (default) a tx_name metadata column is included that lists all transcript names that overlap the exon fragment.

Details

disjointExons creates a GRanges of non-overlapping exon parts with metadata columns of gene_id and exonic_part. Exon parts that overlap more than 1 gene can be dropped with aggregateGenes=FALSE. When includeTranscripts=TRUE a tx_name metadata column is included that lists all transcript names that overlap the exon fragment. This function replaces prepareAnnotationForDEXSeq in the **DEXSeq** package.

Value

A GRanges object.

Author(s)

disjointExons was originally implemented by Mike Love and Alejandro Reyes and then moved (and adapted) to **GenomicFeatures** by Valerie Obenchain.

See Also

exonicParts for an improved version of disjointExons.

---

exonicParts *Extract non-overlapping exonic or intronic parts from a TxDb-like object*

Description

exonicParts and intronicParts extract the non-overlapping (a.k.a. disjoint) exonic or intronic parts from a TxDb-like object.

Usage

```
exonicParts(txdb, linked.to.single.gene.only=FALSE)
intronicParts(txdb, linked.to.single.gene.only=FALSE)

## 3 helper functions used internally by exonicParts() and intronicParts():
tidyTranscripts(txdb, drop.geneless=FALSE)
tidyExons(txdb, drop.geneless=FALSE)
tidyIntrons(txdb, drop.geneless=FALSE)
```

Arguments

txdb	A TxDb object, or any TxDb-like object that supports the `transcripts()` and `exonsBy()` extractors (e.g. an EnsDb object).
linked.to.single.gene.only	TRUE or FALSE. If FALSE (the default), then the disjoint parts are obtained by calling `disjoin()` on all the exons (or introns) in txdb, including on exons (or introns) not linked to a gene or linked to more than one gene. If TRUE, then the disjoint parts are obtained in 2 steps: 1. call `disjoin()` on the exons (or introns) linked to *at least one gene*, 2. then drop the parts linked to more than one gene from the set of exonic (or intronic) parts obtained previously.
drop.geneless	If FALSE (the default), then all the transcripts (or exons, or introns) get extracted from the TxDb object. If TRUE, then only the transcripts (or exons, or introns) that are linked to a gene get extracted from the TxDb object. Note that drop.geneless also impacts the order in which the features are returned: • Transcripts: If drop.geneless is FALSE then transcripts are returned in the same order as with `transcripts`, which is expected to be by internal transcript id (tx_id). Otherwise they are ordered first by gene id (gene_id), then by internal transcript id. • Exons: If drop.geneless is FALSE then exons are ordered first by internal transcript id (tx_id), then by exon rank (exon_rank). Otherwise they are ordered first by gene id (gene_id), then by internal transcript id, and then by exon rank. • Introns: If drop.geneless is FALSE then introns are ordered by internal transcript id (tx_id). Otherwise they are ordered first by gene id (gene_id), then by internal transcript id.

Value

exonicParts returns a disjoint and strictly sorted GRanges object with 1 range per exonic part and with metadata columns tx_id, tx_name, gene_id, exon_id, exon_name, and exon_rank. If linked.to.single.gene.only was set to TRUE, an additional exonic_part metadata column is added that indicates the rank of each exonic part within all the exonic parts linked to the same gene.

`intronicParts` returns a disjoint and strictly sorted GRanges object with 1 range per intronic part and with metadata columns `tx_id`, `tx_name`, and `gene_id`. If `linked.to.single.gene.only` was set to `TRUE`, an additional `intronic_part` metadata column is added that indicates the rank of each intronic part within all the intronic parts linked to the same gene.

`tidyTranscripts` returns a GRanges object with 1 range per transcript and with metadata columns `tx_id`, `tx_name`, and `gene_id`.

`tidyExons` returns a GRanges object with 1 range per exon and with metadata columns `tx_id`, `tx_name`, `gene_id`, `exon_id`, `exon_name`, and `exon_rank`.

`tidyIntrons` returns a GRanges object with 1 range per intron and with metadata columns `tx_id`, `tx_name`, and `gene_id`.

Note

`exonicParts` is a replacement for `disjointExons` with the following differences/improvements:

- Argument `linked.to.single.gene.only` in `exonicParts` replaces argument `aggregateGenes` in `disjointExons`, but has opposite meaning i.e. `exonicParts(txdb, linked.to.single.gene.only=TRUE)` returns the same exonic parts as `disjointExons(txdb, aggregateGenes=FALSE)`.
- Unlike `disjointExons(txdb, aggregateGenes=TRUE)`, `exonicParts(txdb, linked.to.single.gene.only=FALS` does NOT discard exon parts that are not linked to a gene.
- `exonicParts` is almost 2x more efficient than `disjointExons`.
- `exonicParts` works out-of-the-box on any TxDb-like object that supports the `transcripts()` and `exonsBy()` extractors (e.g. on an EnsDb object).

Author(s)

Hervé Pagès

See Also

- `disjoin` in the **IRanges** package.
- `transcripts`, `transcriptsBy`, and `transcriptsByOverlaps`, for extracting genomic feature locations from a TxDb-like object.
- `transcriptLengths` for extracting the transcript lengths (and other metrics) from a TxDb object.
- `extendExonsIntoIntrons` for extending exons into their adjacent introns.
- `extractTranscriptSeqs` for extracting transcript (or CDS) sequences from chromosome sequences.
- `coverageByTranscript` for computing coverage by transcript (or CDS) of a set of ranges.
- The TxDb class.

Examples

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

## ---------------------------------------------------------------------
## exonicParts()
## ---------------------------------------------------------------------

exonic_parts1 <- exonicParts(txdb)
exonic_parts1

## Mapping from exonic parts to genes is many-to-many:
gene_id1 <- mcols(exonic_parts1)$gene_id
gene_id1  # CharacterList object
table(lengths(gene_id1))
## The number of known genes a Human exonic part can be linked to
## varies from 0 to 22!

exonic_parts2 <- exonicParts(txdb, linked.to.single.gene.only=TRUE)
exonic_parts2

## Mapping from exonic parts to genes now is many-to-one:
gene_id2 <- mcols(exonic_parts2)$gene_id
gene_id2[1:20]  # character vector

## Select exonic parts for a given gene:
exonic_parts2[gene_id2 %in% "643837"]

## Sanity checks:
stopifnot(isDisjoint(exonic_parts1), isStrictlySorted(exonic_parts1))
stopifnot(isDisjoint(exonic_parts2), isStrictlySorted(exonic_parts2))
stopifnot(all(exonic_parts2 %within% reduce(exonic_parts1)))
stopifnot(identical(
    lengths(gene_id1) == 1L,
    exonic_parts1 %within% exonic_parts2
))

## ---------------------------------------------------------------------
## intronicParts()
## ---------------------------------------------------------------------

intronic_parts1 <- intronicParts(txdb)
intronic_parts1

## Mapping from intronic parts to genes is many-to-many:
mcols(intronic_parts1)$gene_id
table(lengths(mcols(intronic_parts1)$gene_id))
## A Human intronic part can be linked to 0 to 22 known genes!

intronic_parts2 <- intronicParts(txdb, linked.to.single.gene.only=TRUE)
intronic_parts2
```

```
## Mapping from intronic parts to genes now is many-to-one:
class(mcols(intronic_parts2)$gene_id)  # character vector

## Sanity checks:
stopifnot(isDisjoint(intronic_parts1), isStrictlySorted(intronic_parts1))
stopifnot(isDisjoint(intronic_parts2), isStrictlySorted(intronic_parts2))
stopifnot(all(intronic_parts2 %within% reduce(intronic_parts1)))
stopifnot(identical(
    lengths(mcols(intronic_parts1)$gene_id) == 1L,
    intronic_parts1 %within% intronic_parts2
))

## ---------------------------------------------------------------------
## Helper functions
## ---------------------------------------------------------------------

tidyTranscripts(txdb)                      # Ordered by 'tx_id'.
tidyTranscripts(txdb, drop.geneless=TRUE)  # Ordered first by 'gene_id',
                                           # then by 'tx_id'.

tidyExons(txdb)                            # Ordered first by 'tx_id',
                                           # then by 'exon_rank'.
tidyExons(txdb, drop.geneless=TRUE)        # Ordered first by 'gene_id',
                                           # then by 'tx_id',
                                           # then by 'exon_rank'.

tidyIntrons(txdb)                          # Ordered by 'tx_id'.
tidyIntrons(txdb, drop.geneless=TRUE)      # Ordered first by 'gene_id',
                                           # then by 'tx_id'.
```

`extendExonsIntoIntrons` *Extend exons by a given number of bases into their adjacent introns*

Description

`extendExonsIntoIntrons` extends the supplied exons by a given number of bases into their adjacent introns.

Usage

```
extendExonsIntoIntrons(ex_by_tx, extent=2)
```

Arguments

`ex_by_tx`	A GRangesList object containing exons grouped by transcript. This must be an object as returned by `exonsBy(txdb, by="tx")`, that is: • each list element in `ex_by_tx` must be a GRanges object representing the exons of a given transcript;

- the exons in each list element must be ordered by ascending rank with respect to their transcript.

`extent` Size of the extent in number of bases. 2 by default.

The first exon in a transcript will be extended by that amount on its 3' side only. The last exon in a transcript will be extended by that amount on its 5' side only. All other exons (i.e. intermediate exons) will be extended by that amount on *each* side.

Note that exons that belong to a single-exon transcript don't get extended.

The default value of 2 corresponds to inclusion of the donor/acceptor intronic regions (typically GT/AG).

Value

A copy of GRangesList object `ex_by_tx` where the original exon ranges have been extended.

Names and metadata columns on `ex_by_tx` are propagated to the result.

Author(s)

Hervé Pagès

See Also

- `transcripts`, `transcriptsBy`, and `transcriptsByOverlaps`, for extracting genomic feature locations from a TxDb-like object.
- `exonicParts` and `intronicParts` for extracting non-overlapping exonic or intronic parts from a TxDb-like object.
- `extractTranscriptSeqs` for extracting transcript (or CDS) sequences from chromosome sequences.
- The TxDb class.

Examples

```
## With toy transcripts:
ex_by_tx <- GRangesList(
    TX1="chr1:10-20:+",
    TX2=c("chr1:10-20:+", "chr1:50-75:+"),
    TX3=c("chr1:10-20:+", "chr1:50-75:+", "chr1:100-120:+"),
    TX4="chr1:10-20:-",
    TX5=c("chr1:10-20:-", "chr1:50-75:-"),
    TX6=c("chr1:10-20:-", "chr1:50-75:-", "chr1:100-120:-")
)

extended <- extendExonsIntoIntrons(ex_by_tx, extent=2)
extended[1:3]
extended[4:6]

## With real-world transcripts:
library(TxDb.Celegans.UCSC.ce11.ensGene)
txdb <- TxDb.Celegans.UCSC.ce11.ensGene
```

```
ex_by_tx <- exonsBy(txdb, by="tx")
ex_by_tx

extendExonsIntoIntrons(ex_by_tx, extent=2)

## Sanity check:
stopifnot(identical(extendExonsIntoIntrons(ex_by_tx, extent=0), ex_by_tx))
```

`extractTranscriptSeqs` *Extract transcript (or CDS) sequences from chromosome sequences*

Description

`extractTranscriptSeqs` extracts transcript (or CDS) sequences from an object representing a single chromosome or a collection of chromosomes.

Usage

```
extractTranscriptSeqs(x, transcripts, ...)

## S4 method for signature 'DNAString'
extractTranscriptSeqs(x, transcripts, strand="+")

## S4 method for signature 'ANY'
extractTranscriptSeqs(x, transcripts, ...)
```

Arguments

`x`	An object representing a single chromosome or a collection of chromosomes. More precisely, `x` can be a DNAString object (single chromosome), or a BSgenome object (collection of chromosomes). Other objects representing a collection of chromosomes are supported (e.g. FaFile objects in the **Rsamtools** package) as long as `seqinfo` and `getSeq` work on them.
`transcripts`	An object representing the exon ranges of each transcript to extract. More precisely: • If `x` is a DNAString object, then `transcripts` must be an IntegerRangesList object. • If `x` is a BSgenome object or any object representing a collection of chromosomes, then `transcripts` must be a GRangesList object or any object for which `exonsBy` is implemented (e.g. a TxDb or EnsDb object). If the latter, then it's first turned into a GRangesList object with `exonsBy(transcripts, by="tx", ...)`. Note that, for each transcript, the exons must be ordered by ascending *rank*, that is, by ascending position *in the transcript* (when going in the 5' to 3' direction). This generally means (but not always) that they are also ordered from 5' to 3' on the reference genome. More precisely:

- For a transcript located on the plus strand, the exons will typically (but not necessarily) be ordered by ascending position on the reference genome.
- For a transcript located on the minus strand, the exons will typically (but not necessarily) be ordered by descending position on the reference genome.

If `transcripts` was obtained with `exonsBy` (see above), then the exons are guaranteed to be ordered by ascending rank. See `?exonsBy` for more information.

`...` Additional arguments, for use in specific methods.

For the default method, additional arguments are allowed only when `transcripts` is not a GRangesList object, in which case they are passed to the internal call to `exonsBy` (see above).

`strand` Only supported when `x` is a DNAString object.

Can be an atomic vector, a factor, or an Rle object, in which case it indicates the strand of each transcript (i.e. all the exons in a transcript are considered to be on the same strand). More precisely: it's turned into a factor (or factor-Rle) that has the "standard strand levels" (this is done by calling the `strand` function on it). Then it's recycled to the length of IntegerRangesList object `transcripts` if needed. In the resulting object, the i-th element is interpreted as the strand of all the exons in the i-th transcript.

`strand` can also be a list-like object, in which case it indicates the strand of each exon, individually. Thus it must have the same *shape* as IntegerRangesList object `transcripts` (i.e. same length plus `strand[[i]]` must have the same length as `transcripts[[i]]` for all `i`).

`strand` can only contain `"+"` and/or `"-"` values. `"*"` is not allowed.

Value

A DNAStringSet object *parallel* to `transcripts`, that is, the i-th element in it is the sequence of the i-th transcript in `transcripts`.

Author(s)

Hervé Pagès

See Also

- `coverageByTranscript` for computing coverage by transcript (or CDS) of a set of ranges.
- `transcriptLengths` for extracting the transcript lengths (and other metrics) from a TxDb object.
- `extendExonsIntoIntrons` for extending exons into their adjacent introns.
- The `transcriptLocs2refLocs` function for converting transcript-based locations into reference-based locations.
- The `available.genomes` function in the **BSgenome** package for checking avaibility of BSgenome data packages (and installing the desired one).
- The DNAString and DNAStringSet classes defined and documented in the **Biostrings** package.

- The translate function in the **Biostrings** package for translating DNA or RNA sequences into amino acid sequences.
- The GRangesList class defined and documented in the **GenomicRanges** package.
- The IntegerRangesList class defined and documented in the **IRanges** package.
- The exonsBy function for extracting exon ranges grouped by transcript.
- The TxDb class.

Examples

```
## ---------------------------------------------------------------------
## 1. A TOY EXAMPLE
## ---------------------------------------------------------------------

library(Biostrings)

## A chromosome of length 30:
x <- DNAString("ATTTAGGACACTCCCTGAGGACAAGACCCC")

## 2 transcripts on 'x':
tx1 <- IRanges(1, 8)            # 1 exon
tx2 <- c(tx1, IRanges(12, 30))  # 2 exons
transcripts <- IRangesList(tx1=tx1, tx2=tx2)
extractTranscriptSeqs(x, transcripts)

## By default, all the exons are considered to be on the plus strand.
## We can use the 'strand' argument to tell extractTranscriptSeqs()
## to extract them from the minus strand.

## Extract all the exons from the minus strand:
extractTranscriptSeqs(x, transcripts, strand="-")

## Note that, for a transcript located on the minus strand, the exons
## should typically be ordered by descending position on the reference
## genome in order to reflect their rank in the transcript:
extractTranscriptSeqs(x, IRangesList(tx1=tx1, tx2=rev(tx2)), strand="-")

## Extract the exon of the 1st transcript from the minus strand:
extractTranscriptSeqs(x, transcripts, strand=c("-", "+"))

## Extract the 2nd exon of the 2nd transcript from the minus strand:
extractTranscriptSeqs(x, transcripts, strand=list("-", c("+", "-")))

## ---------------------------------------------------------------------
## 2. A REAL EXAMPLE
## ---------------------------------------------------------------------

## Load a genome:
library(BSgenome.Hsapiens.UCSC.hg19)
genome <- BSgenome.Hsapiens.UCSC.hg19

## Load a TxDb object:
```

```
txdb_file <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                         package="GenomicFeatures")
txdb <- loadDb(txdb_file)

## Check that 'txdb' is based on the hg19 assembly:
txdb

## Extract the exon ranges grouped by transcript from 'txdb':
transcripts <- exonsBy(txdb, by="tx", use.names=TRUE)

## Extract the transcript sequences from the genome:
tx_seqs <- extractTranscriptSeqs(genome, transcripts)
tx_seqs

## A sanity check:
stopifnot(identical(width(tx_seqs), unname(sum(width(transcripts)))))

## Note that 'tx_seqs' can also be obtained with:
extractTranscriptSeqs(genome, txdb, use.names=TRUE)

## ---------------------------------------------------------------------
## 3. USING extractTranscriptSeqs() TO EXTRACT CDS SEQUENCES
## ---------------------------------------------------------------------

cds <- cdsBy(txdb, by="tx", use.names=TRUE)
cds_seqs <- extractTranscriptSeqs(genome, cds)
cds_seqs

## A sanity check:
stopifnot(identical(width(cds_seqs), unname(sum(width(cds)))))

## Note that, alternatively, the CDS sequences can be obtained from the
## transcript sequences by removing the 5' and 3' UTRs:
tx_lens <- transcriptLengths(txdb, with.utr5_len=TRUE, with.utr3_len=TRUE)
stopifnot(identical(tx_lens$tx_name, names(tx_seqs)))  # sanity
## Keep the rows in 'tx_lens' that correspond to a sequence in 'cds_seqs'
## and put them in the same order as in 'cds_seqs':
m <- match(names(cds_seqs), names(tx_seqs))
tx_lens <- tx_lens[m, ]
utr5_width <- tx_lens$utr5_len
utr3_width <- tx_lens$utr3_len
cds_seqs2 <- narrow(tx_seqs[m],
                    start=utr5_width+1L, end=-(utr3_width+1L))
stopifnot(identical(as.character(cds_seqs2), as.character(cds_seqs)))

## ---------------------------------------------------------------------
## 4. TRANSLATE THE CDS SEQUENCES
## ---------------------------------------------------------------------

prot_seqs <- translate(cds_seqs, if.fuzzy.codon="solve")

## Note that, by default, translate() uses The Standard Genetic Code to
## translate codons into amino acids. However, depending on the organism,
```

```
## a different genetic code might be needed to translate CDS sequences
## located on the mitochodrial chromosome. For example, for vertebrates,
## the following code could be used to correct 'prot_seqs':
SGC1 <- getGeneticCode("SGC1")
chrM_idx <- which(all(seqnames(cds) == "chrM"))
prot_seqs[chrM_idx] <- translate(cds_seqs[chrM_idx], genetic.code=SGC1,
                                 if.fuzzy.codon="solve")
```

`extractUpstreamSeqs` *Extract sequences upstream of a set of genes or transcripts*

Description

`extractUpstreamSeqs` is a generic function for extracting sequences upstream of a supplied set of genes or transcripts.

Usage

```
extractUpstreamSeqs(x, genes, width=1000, ...)

## Dispatch is on the 2nd argument!

## S4 method for signature 'GenomicRanges'
extractUpstreamSeqs(x, genes, width=1000)

## S4 method for signature 'TxDb'
extractUpstreamSeqs(x, genes, width=1000, exclude.seqlevels=NULL)
```

Arguments

`x`	An object containing the chromosome sequences from which to extract the upstream sequences. It can be a BSgenome, TwoBitFile, or FaFile object, or any *genome sequence container*. More formally, `x` must be an object for which `seqinfo` and `getSeq` are defined.
`genes`	An object containing the locations (i.e. chromosome name, start, end, and strand) of the genes or transcripts with respect to the reference genome. Only GenomicRanges and TxDb objects are supported at the moment. If the latter, the gene locations are obtained by calling the `genes` function on the TxDb object internally.
`width`	How many bases to extract upstream of each TSS (transcription start site).
`...`	Additional arguments, for use in specific methods.
`exclude.seqlevels`	A character vector containing the chromosome names (a.k.a. sequence levels) to exclude when the genes are obtained from a TxDb object.

Value

A DNAStringSet object containing one upstream sequence per gene (or per transcript if genes is a GenomicRanges object containing transcript ranges).

More precisely, if genes is a GenomicRanges object, the returned object is *parallel* to it, that is, the i-th element in the returned object is the upstream sequence corresponding to the i-th gene (or transcript) in genes. Also the names on the GenomicRanges object are propagated to the returned object.

If genes is a TxDb object, the names on the returned object are the gene IDs found in the TxDb object. To see the type of gene IDs (i.e. Entrez gene ID or Ensembl gene ID or ...), you can display genes with show(genes).

In addition, the returned object has the following metadata columns (accessible with mcols) that provide some information about the gene (or transcript) corresponding to each upstream sequence:

- gene_seqnames: the chromosome name of the gene (or transcript);
- gene_strand: the strand of the gene (or transcript);
- gene_TSS: the transcription start site of the gene (or transcript).

Note

IMPORTANT: Always make sure to use a TxDb package (or TxDb object) that contains a gene model compatible with the *genome sequence container* x, that is, a gene model based on the exact same reference genome as x.

See http://bioconductor.org/packages/release/BiocViews.html#___TxDb for the list of TxDb packages available in the current release of Bioconductor. Note that you can make your own custom TxDb object from various annotation resources by using one of the makeTxDbFrom*() functions listed in the "See also" section below.

Author(s)

Hervé Pagès

See Also

- makeTxDbFromUCSC, makeTxDbFromBiomart, and makeTxDbFromEnsembl, for making a TxDb object from online resources.
- makeTxDbFromGRanges and makeTxDbFromGFF for making a TxDb object from a GRanges object, or from a GFF or GTF file.
- The available.genomes function in the **BSgenome** package for checking avaibility of BSgenome data packages (and installing the desired one).
- The BSgenome, TwoBitFile, and FaFile classes, defined and documented in the **BSgenome**, **rtracklayer**, and **Rsamtools** packages, respectively.
- The TxDb class.
- The genes function for extracting gene ranges from a TxDb object.
- The GenomicRanges class defined and documented in the **GenomicRanges** package.
- The DNAStringSet class defined and documented in the **Biostrings** package.

- The seqinfo getter defined and documented in the **GenomeInfoDb** package.
- The getSeq function for extracting subsequences from a sequence container.

Examples

```
## Load a genome:
library(BSgenome.Dmelanogaster.UCSC.dm3)
genome <- BSgenome.Dmelanogaster.UCSC.dm3
genome

## Use a TxDb object:
library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
txdb <- TxDb.Dmelanogaster.UCSC.dm3.ensGene
txdb  # contains Ensembl gene IDs

## Because the chrU and chrUextra sequences are made of concatenated
## scaffolds (see http://genome.ucsc.edu/cgi-bin/hgGateway?db=dm3),
## extracting the upstream sequences for genes located on these
## scaffolds is not reliable. So we exclude them:
exclude <- c("chrU", "chrUextra")
up1000seqs <- extractUpstreamSeqs(genome, txdb, width=1000,
                                  exclude.seqlevels=exclude)
up1000seqs  # the names are Ensembl gene IDs
mcols(up1000seqs)

## Upstream sequences for genes close to the chromosome bounds can be
## shorter than 1000 (note that this does not happen for circular
## chromosomes like chrM):
table(width(up1000seqs))
mcols(up1000seqs)[width(up1000seqs) != 1000, ]
```

FeatureDb-class *FeatureDb objects*

Description

WARNING: The FeatureDb/makeFeatureDbFromUCSC/features code base is no longer actively maintained and FeatureDb-related functionalities might get deprecated in the near future. Please use makeFeatureDbFromUCSC for a convenient way to import transcript annotations from UCSC online resources into Bioconductor.

The FeatureDb class is a generic container for storing genomic locations of an arbitrary type of genomic features.

See ?TxDb for a container for storing transcript annotations.

See ?makeFeatureDbFromUCSC for a convenient way to make FeatureDb objects from BioMart online resources.

Methods

In the code snippets below, x is a FeatureDb object.

`metadata(x)`: Return x's metadata in a data frame.

Author(s)

Marc Carlson

See Also

- The TxDb class for storing transcript annotations.
- `makeFeatureDbFromUCSC` for a convenient way to make a FeatureDb object from UCSC online resources.
- `saveDb` and `loadDb` for saving and loading the database content of a FeatureDb object.
- `features` for how to extract genomic features from a FeatureDb object.

Examples

```
fdb_file <- system.file("extdata", "FeatureDb.sqlite",
                        package="GenomicFeatures")
fdb <- loadDb(fdb_file)
fdb
```

---

features *Extract simple features from a FeatureDb object*

Description

WARNING: The FeatureDb/makeFeatureDbFromUCSC/features code base is no longer actively maintained and FeatureDb-related functionalities might get deprecated in the near future. Please use `makeFeatureDbFromUCSC` for a convenient way to import transcript annotations from UCSC online resources into Bioconductor.

Generic function to extract genomic features from a FeatureDb object.

Usage

```
features(x)
## S4 method for signature 'FeatureDb'
features(x)
```

Arguments

x	A FeatureDb object.

Value

a GRanges object

Author(s)

M. Carlson

See Also

FeatureDb

Examples

```
fdb <- loadDb(system.file("extdata", "FeatureDb.sqlite",
                                  package="GenomicFeatures"))
features(fdb)
```

`getPromoterSeq` *Get gene promoter sequences*

Description

Extract sequences for the genes or transcripts specified in the query (aGRanges or GRangesList object) from a BSgenome object or an FaFile.

Usage

```
  ## S4 method for signature 'GRangesList'
getPromoterSeq(query, subject, upstream=2000, downstream=200, ...)
  ## S4 method for signature 'GRangesList'
getPromoterSeq(query, subject, upstream=2000, downstream=200, ...)
  ## S4 method for signature 'GRanges'
getPromoterSeq(query, subject, upstream=2000, downstream=200, ...)
```

Arguments

`query`	A GRanges or GRangesList object containing genes grouped by transcript.
`subject`	A BSgenome object or a FaFile from which the sequences will be taken.
`upstream`	The number of DNA bases to include upstream of the TSS (transcription start site)
`downstream`	The number of DNA bases to include downstream of the TSS (transcription start site)
`...`	Additional arguments

Details

getPromoterSeq is an overloaded method dispatching on query, which is either a GRanges or a GRangesList. It is a wrapper for the promoters and getSeq functions. The purpose is to allow sequence extraction from either a BSgenome or FaFile.

Default values for upstream and downstream were chosen based on our current understanding of gene regulation. On average, promoter regions in the mammalian genome are 5000 bp upstream and downstream of the transcription start site.

Value

A DNAStringSet or DNAStringSetList instance corresponding to the GRanges or GRangesList supplied in the query.

Author(s)

Paul Shannon

See Also

intra-range-methods ## promoters method for IntegerRanges objects intra-range-methods ## promoters method for GenomicRanges objects getSeq

Examples

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(BSgenome.Hsapiens.UCSC.hg19)

e2f3 <- "1871"  # entrez geneID for a cell cycle control transcription
                # factor, chr6 on the plus strand

transcriptCoordsByGene.GRangesList <-
   transcriptsBy (TxDb.Hsapiens.UCSC.hg19.knownGene, by = "gene") [e2f3]
  # a GrangesList of length one, describing three transcripts

promoter.seqs <- getPromoterSeq (transcriptCoordsByGene.GRangesList,
                                 Hsapiens, upstream=10, downstream=0)
  # DNAStringSetList of length 1
  # [["1871"]] GCTTCCTGGA GCTTCCTGGA CGGAGCCAGG
```

---

id2name *Map internal ids to external names for a given feature type*

Description

Utility function for retrieving the mapping from the internal ids to the external names of a given feature type.

Usage

```
id2name(txdb, feature.type=c("tx", "exon", "cds"))
```

Arguments

txdb	A TxDb object.
feature.type	The feature type for which the mapping must be retrieved.

Details

Transcripts, exons and CDS in a TxDb object are stored in seperate tables where the primary key is an integer called *feature internal id*. This id is stored in the "tx_id" column for transcripts, in the "exon_id" column for exons, and in the "cds_id" column for CDS. Unlike other commonly used ids like Entrez Gene IDs or Ensembl IDs, this internal id was generated at the time the TxDb object was created and has no meaning outside the scope of this object.

The id2name function can be used to translate this internal id into a more informative id or name called *feature external name*. This name is stored in the "tx_name" column for transcripts, in the "exon_name" column for exons, and in the "cds_name" column for CDS.

Note that, unlike the feature internal id, the feature external name is not guaranteed to be unique or even defined (the column can contain NAs).

Value

A named character vector where the names are the internal ids and the values the external names.

Author(s)

Hervé Pagès

See Also

- transcripts, transcriptsBy, and transcriptsByOverlaps, for how to extract genomic features from a TxDb object.
- The TxDb class.

Examples

```
txdb1_file <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                          package="GenomicFeatures")
txdb1 <- loadDb(txdb1_file)
id2name(txdb1, feature.type="tx")[1:4]
id2name(txdb1, feature.type="exon")[1:4]
id2name(txdb1, feature.type="cds")[1:4]

txdb2_file <- system.file("extdata", "Biomart_Ensembl_sample.sqlite",
                          package="GenomicFeatures")
txdb2 <- loadDb(txdb2_file)
id2name(txdb2, feature.type="tx")[1:4]
id2name(txdb2, feature.type="exon")[1:4]
id2name(txdb2, feature.type="cds")[1:4]
```

makeFeatureDbFromUCSC *Making a FeatureDb object from annotations available at the UCSC Genome Browser*

Description

WARNING: The FeatureDb/makeFeatureDbFromUCSC/features code base is no longer actively maintained and FeatureDb-related functionalities might get deprecated in the near future. Please use makeFeatureDbFromUCSC for a convenient way to import transcript annotations from UCSC online resources into Bioconductor.

The makeFeatureDbFromUCSC function allows the user to make a FeatureDb object from simple annotation tracks at UCSC. The tracks in question must (at a minimum) have a start, end and a chromosome affiliation in order to be made into a FeatureDb. This function requires a precise declaration of its first three arguments to indicate which genome, track and table wish to be imported. There are discovery functions provided to make this process go smoothly.

Usage

```
supportedUCSCFeatureDbTracks(genome)

supportedUCSCFeatureDbTables(genome, track)

UCSCFeatureDbTableSchema(genome,
                         track,
                         tablename)

makeFeatureDbFromUCSC(
        genome,
        track,
        tablename,
        columns = UCSCFeatureDbTableSchema(genome,track,tablename),
        url="http://genome.ucsc.edu/cgi-bin/",
        goldenPath.url=getOption("UCSC.goldenPath.url"),
        chromCol,
        chromStartCol,
        chromEndCol,
        taxonomyId=NA)
```

Arguments

genome	genome abbreviation used by UCSC and obtained by ucscGenomes()[, "db"]. For example: "hg18".
track	name of the UCSC track. Use supportedUCSCFeatureDbTracks to get the list of available tracks for a particular genome
tablename	name of the UCSC table containing the annotations to retrieve. Use the supportedUCSCFeatureDbTables utility function to get the list of supported tables for a track.

columns	a named character vector to list out the names and types of the other columns that the downloaded track should have. Use UCSCFeatureDbTableSchema to retrieve this information for a particular table.
url,goldenPath.url	use to specify the location of an alternate UCSC Genome Browser.
chromCol	If the schema comes back and the 'chrom' column has been labeled something other than 'chrom', use this argument to indicate what that column has been labeled as so we can properly designate it. This could happen (for example) with the knownGene track tables, which has no 'chromStart' or 'chromEnd' columns, but which DOES have columns that could reasonably substitute for these columns under particular circumstances. Therefore we allow these three columns to have arguments so that their definition can be re-specified
chromStartCol	Same thing as chromCol, but for renames of 'chromStart'
chromEndCol	Same thing as chromCol, but for renames of 'chromEnd'
taxonomyId	By default this value is NA and the organism inferred will be used to look up the correct value for this. But you can use this argument to override that and supply your own valid taxId here.

Details

makeFeatureDbFromUCSC is a convenience function that builds a tiny database from one of the UCSC track tables. supportedUCSCFeatureDbTracks a convenience function that returns potential track names that could be used to make FeatureDb objects supportedUCSCFeatureDbTables a convenience function that returns potential table names for FeatureDb objects (table names go with a track name) UCSCFeatureDbTableSchema A convenience function that creates a named vector of types for all the fields that can potentially be supported for a given track. By default, this will be called on your specified tablename to include all of the fields in a track.

Value

A FeatureDb object for makeFeatureDbFromUCSC. Or in the case of supportedUCSCFeatureDbTracks and UCSCFeatureDbTableSchema a named character vector

Author(s)

M. Carlson

See Also

ucscGenomes,

Examples

```
## Display the list of genomes available at UCSC:
library(GenomicFeatures)
library(rtracklayer)
ucscGenomes()[ , "db"]

## Display the list of Tracks supported by makeFeatureDbFromUCSC():
```

```
# supportedUCSCFeatureDbTracks("mm10")

## Display the list of tables supported by your track:
supportedUCSCFeatureDbTables(genome="mm10",
                             track="qPCR Primers")

## Display fields that could be passed in to colnames:
UCSCFeatureDbTableSchema(genome="mm10",
                         track="qPCR Primers",
                         tablename="qPcrPrimers")

## Retrieving a full transcript dataset for Mouse from UCSC:
fdb <- makeFeatureDbFromUCSC(genome="mm10",
                             track="qPCR Primers",
                             tablename="qPcrPrimers")
fdb
```

---

makeTxDb *Making a TxDb object from user supplied annotations*

Description

makeTxDb is a low-level constructor for making a TxDb object from user supplied transcript annotations.

Note that the end user will rarely need to use makeTxDb directly but will typically use one of the high-level constructors makeTxDbFromUCSC, makeTxDbFromEnsembl, or makeTxDbFromGFF.

Usage

```
makeTxDb(transcripts, splicings, genes=NULL,
         chrominfo=NULL, metadata=NULL,
         reassign.ids=FALSE, on.foreign.transcripts=c("error", "drop"))
```

Arguments

transcripts	Data frame containing the genomic locations of a set of transcripts.
splicings	Data frame containing the exon and CDS locations of a set of transcripts.
genes	Data frame containing the genes associated to a set of transcripts.
chrominfo	Data frame containing information about the chromosomes hosting the set of transcripts.
metadata	2-column data frame containing meta information about this set of transcripts like organism, genome, UCSC table, etc... The names of the columns must be "name" and "value" and their type must be character.
reassign.ids	TRUE or FALSE. Controls how internal ids should be assigned for each type of feature i.e. for transcripts, exons, and CDS. For each type, if reassign.ids is FALSE (the default) and if the ids are supplied, then they are used as the internal

ids, otherwise the internal ids are assigned in a way that is compatible with the order defined by ordering the features first by chromosome, then by strand, then by start, and finally by end.

`on.foreign.transcripts`
: Controls what to do when the input contains *foreign transcripts* i.e. transcripts that are on sequences not in `chrominfo`. If set to `"error"` (the default)

Details

The `transcripts` (required), `splicings` (required) and `genes` (optional) arguments must be data frames that describe a set of transcripts and the genomic features related to them (exons, CDS and genes at the moment). The `chrominfo` (optional) argument must be a data frame containing chromosome information like the length of each chromosome.

`transcripts` must have 1 row per transcript and the following columns:

- `tx_id`: Transcript ID. Integer vector. No NAs. No duplicates.
- `tx_chrom`: Transcript chromosome. Character vector (or factor) with no NAs.
- `tx_strand`: Transcript strand. Character vector (or factor) with no NAs where each element is either `"+"` or `"-"`.
- `tx_start`, `tx_end`: Transcript start and end. Integer vectors with no NAs.
- `tx_name`: [optional] Transcript name. Character vector (or factor). NAs and/or duplicates are ok.
- `tx_type`: [optional] Transcript type (e.g. mRNA, ncRNA, snoRNA, etc...). Character vector (or factor). NAs and/or duplicates are ok.
- `gene_id`: [optional] Associated gene. Character vector (or factor). NAs and/or duplicates are ok.

Other columns, if any, are ignored (with a warning).

`splicings` must have N rows per transcript, where N is the nb of exons in the transcript. Each row describes an exon plus, optionally, the CDS contained in this exon. Its columns must be:

- `tx_id`: Foreign key that links each row in the `splicings` data frame to a unique row in the `transcripts` data frame. Note that more than 1 row in `splicings` can be linked to the same row in `transcripts` (many-to-one relationship). Same type as `transcripts$tx_id` (integer vector). No NAs. All the values in this column must be present in `transcripts$tx_id`.
- `exon_rank`: The rank of the exon in the transcript. Integer vector with no NAs. (`tx_id`, `exon_rank`) pairs must be unique.
- `exon_id`: [optional] Exon ID. Integer vector with no NAs.
- `exon_name`: [optional] Exon name. Character vector (or factor). NAs and/or duplicates are ok.
- `exon_chrom`: [optional] Exon chromosome. Character vector (or factor) with no NAs. If missing then `transcripts$tx_chrom` is used. If present then `exon_strand` must also be present.
- `exon_strand`: [optional] Exon strand. Character vector (or factor) with no NAs. If missing then `transcripts$tx_strand` is used and `exon_chrom` must also be missing.

- exon_start, exon_end: Exon start and end. Integer vectors with no NAs.
- cds_id: [optional] CDS ID. Integer vector. If present then cds_start and cds_end must also be present. NAs are allowed and must match those in cds_start and cds_end.
- cds_name: [optional] CDS name. Character vector (or factor). If present then cds_start and cds_end must also be present. NAs and/or duplicates are ok. Must contain NAs at least where cds_start and cds_end contain them.
- cds_start, cds_end: [optional] CDS start and end. Integer vectors. If one of the 2 columns is missing then all cds_* columns must be missing. NAs are allowed and must occur at the same positions in cds_start and cds_end.
- cds_phase: [optional] CDS phase. Integer vector. If present then cds_start and cds_end must also be present. NAs are allowed and must match those in cds_start and cds_end.

Other columns, if any, are ignored (with a warning).

genes should not be supplied if transcripts has a gene_id column. If supplied, it must have N rows per transcript, where N is the nb of genes linked to the transcript (N will be 1 most of the time). Its columns must be:

- tx_id: [optional] genes must have either a tx_id or a tx_name column but not both. Like splicings$tx_id, this is a foreign key that links each row in the genes data frame to a unique row in the transcripts data frame.
- tx_name: [optional] Can be used as an alternative to the genes$tx_id foreign key.
- gene_id: Gene ID. Character vector (or factor). No NAs.

Other columns, if any, are ignored (with a warning).

chrominfo must have 1 row per chromosome and the following columns:

- chrom: Chromosome name. Character vector (or factor) with no NAs and no duplicates.
- length: Chromosome length. Integer vector with either all NAs or no NAs.
- is_circular: [optional] Chromosome circularity flag. Logical vector. NAs are ok.

Other columns, if any, are ignored (with a warning).

Value

A TxDb object.

Author(s)

Hervé Pagès

See Also

- makeTxDbFromUCSC, makeTxDbFromBiomart, and makeTxDbFromEnsembl, for making a TxDb object from online resources.
- makeTxDbFromGRanges and makeTxDbFromGFF for making a TxDb object from a GRanges object, or from a GFF or GTF file.
- The TxDb class.
- saveDb and loadDb in the **AnnotationDbi** package for saving and loading a TxDb object as an SQLite file.

Examples

```
transcripts <- data.frame(
                  tx_id=1:3,
                  tx_chrom="chr1",
                  tx_strand=c("-", "+", "+"),
                  tx_start=c(1, 2001, 2001),
                  tx_end=c(999, 2199, 2199))
splicings <-  data.frame(
                  tx_id=c(1L, 2L, 2L, 2L, 3L, 3L),
                  exon_rank=c(1, 1, 2, 3, 1, 2),
                  exon_start=c(1, 2001, 2101, 2131, 2001, 2131),
                  exon_end=c(999, 2085, 2144, 2199, 2085, 2199),
                  cds_start=c(1, 2022, 2101, 2131, NA, NA),
                  cds_end=c(999, 2085, 2144, 2193, NA, NA),
                  cds_phase=c(0, 0, 2, 0, NA, NA))

txdb <- makeTxDb(transcripts, splicings)
```

---

makeTxDbFromBiomart *Make a TxDb object from annotations available on a BioMart database*

Description

The `makeTxDbFromBiomart` function allows the user to make a TxDb object from transcript annotations available on a BioMart database.

Note that `makeTxDbFromBiomart` is being phased out in favor of `makeTxDbFromEnsembl`.

Usage

```
makeTxDbFromBiomart(biomart="ENSEMBL_MART_ENSEMBL",
                    dataset="hsapiens_gene_ensembl",
                    transcript_ids=NULL,
                    circ_seqs=NULL,
                    filter=NULL,
                    id_prefix="ensembl_",
                    host="https://www.ensembl.org",
                    port,
                    taxonomyId=NA,
                    miRBaseBuild=NA)

getChromInfoFromBiomart(biomart="ENSEMBL_MART_ENSEMBL",
                        dataset="hsapiens_gene_ensembl",
                        id_prefix="ensembl_",
                        host="https://www.ensembl.org",
                        port)
```

Arguments

biomart	which BioMart database to use. Get the list of all available BioMart databases with the listMarts function from the biomaRt package. See the details section below for a list of BioMart databases with compatible transcript annotations.
dataset	which dataset from BioMart. For example: "hsapiens_gene_ensembl", "mmusculus_gene_ensembl", "dmelanogaster_gene_ensembl", "celegans_gene_ensembl", "scerevisiae_gene_ensembl", etc in the ensembl database. See the examples section below for how to discover which datasets are available in a given BioMart database.
transcript_ids	optionally, only retrieve transcript annotation data for the specified set of transcript ids. If this is used, then the meta information displayed for the resulting TxDb object will say 'Full dataset: no'. Otherwise it will say 'Full dataset: yes'.
circ_seqs	a character vector to list out which chromosomes should be marked as circular.
filter	Additional filters to use in the BioMart query. Must be a named list. An example is filter=list(source="entrez")
id_prefix	Specifies the prefix used in BioMart attributes. For example, some BioMarts may have an attribute specified as "ensembl_transcript_id" whereas others have the same attribute specified as "transcript_id". Defaults to "ensembl_".
host	The host URL of the BioMart. Defaults to www.ensembl.org.
port	The port to use in the HTTP communication with the host. This argument has been deprecated. It is handled by useEnsembl depending on the host input.
taxonomyId	By default this value is NA and the dataset selected will be used to look up the correct value for this. But you can use this argument to override that and supply your own taxId here (which will be independently checked to make sure its a real taxonomy id). Normally you should never need to use this.
miRBaseBuild	specify the string for the appropriate build Information from mirbase.db to use for microRNAs. This can be learned by calling supportedMiRBaseBuildValues. By default, this value will be set to NA, which will inactivate the microRNAs accessor.

Details

makeTxDbFromBiomart is a convenience function that feeds data from a BioMart database to the lower level makeTxDb function. See ?makeTxDbFromUCSC for a similar function that feeds data from the UCSC source.

Here is a list of datasets known to be compatible with makeTxDbFromBiomart (list updated on September 18, 2017):

1. All the datasets in the main Ensembl database. Get the list with:

```
mart <- biomaRt::useEnsembl(biomart="ENSEMBL_MART_ENSEMBL",
                            host="https://www.ensembl.org")
biomaRt::listDatasets(mart)
```

2. All the datasets in the Ensembl Fungi database. Get the list with:

```
mart <- biomaRt::useEnsemblGenomes(biomart="fungi_mart")
biomaRt::listDatasets(mart)
```

3. All the datasets in the Ensembl Metazoa database. Get the list with:

```
mart <- biomaRt::useEnsemblGenomes(biomart="metazoa_mart")
biomaRt::listDatasets(mart)
```

4. All the datasets in the Ensembl Plants database. Get the list with:

```
mart <- biomaRt::useEnsemblGenomes(biomart="plants_mart")
biomaRt::listDatasets(mart)
```

5. All the datasets in the Ensembl Protists database. Get the list with:

```
mart <- biomaRt::useEnsemblGenomes(biomart="protists_mart")
biomaRt::listDatasets(mart)
```

6. All the datasets in the Gramene Mart. Get the list with:

```
mart <- biomaRt::useEnsembl(biomart="ENSEMBL_MART_PLANT",
                            host="https://ensembl.gramene.org")
biomaRt::listDatasets(mart)
```

Note that BioMart is not currently available for Ensembl Bacteria.

Also please note that not all these datasets have CDS information.

Value

A TxDb object for `makeTxDbFromBiomart`.

A data frame with 1 row per chromosome (or scaffold) and with columns `chrom` and `length` for `getChromInfoFromBiomart`.

Author(s)

M. Carlson and H. Pagès

See Also

- `makeTxDbFromUCSC` and `makeTxDbFromEnsembl` for making a TxDb object from other online resources.
- `makeTxDbFromGRanges` and `makeTxDbFromGFF` for making a TxDb object from a GRanges object, or from a GFF or GTF file.
- The `listMarts`, `useEnsembl`, `listDatasets`, and `listFilters` functions in the **biomaRt** package.
- The `supportedMiRBaseBuildValues` function for listing all the possible values for the `miRBaseBuild` argument.
- The TxDb class.
- `makeTxDb` for the low-level function used by the `makeTxDbFrom*` functions to make the TxDb object returned to the user.

Examples

```
## ---------------------------------------------------------------------
## A. BASIC USAGE
## ---------------------------------------------------------------------

## We can use listDatasets() from the biomaRt package to list the
## datasets available in the "ENSEMBL_MART_ENSEMBL" BioMart database:
library(biomaRt)
listMarts(host="https://www.ensembl.org")
mart <- useEnsembl(biomart="ENSEMBL_MART_ENSEMBL", host="https://www.ensembl.org")
datasets <- listDatasets(mart)
head(datasets)
subset(datasets, grepl("elegans", dataset, ignore.case=TRUE))

## Retrieve the full transcript dataset for Worm:
txdb1 <- makeTxDbFromBiomart(dataset="celegans_gene_ensembl")
txdb1

## Retrieve an incomplete transcript dataset for Human:
transcript_ids <- c(
    "ENST00000013894",
    "ENST00000268655",
    "ENST00000313243",
    "ENST00000435657",
    "ENST00000384428",
    "ENST00000478783"
)

if (interactive()) {
  txdb2 <- makeTxDbFromBiomart(dataset="hsapiens_gene_ensembl",
                               transcript_ids=transcript_ids)
  txdb2  # note that these annotations match the GRCh38 genome assembly
}

## ---------------------------------------------------------------------
## B. ACCESSING THE EnsemblGenomes MARTS
## ---------------------------------------------------------------------

library(biomaRt)

## Note that BioMart is not currently available for Ensembl Bacteria.

## ---------------------
## --- Ensembl Fungi ---

mart <- useEnsemblGenomes(biomart="fungi_mart")
datasets <- listDatasets(mart)
datasets$dataset
yeast_txdb <- makeTxDbFromBiomart(biomart="fungi_mart",
                                  dataset="scerevisiae_eg_gene",
                                  host="https://fungi.ensembl.org")
yeast_txdb
```

```
## Note that the dataset for Yeast on Ensembl Fungi is not necessarily
## the same as on the main Ensembl database:
yeast_txdb0 <- makeTxDbFromBiomart(dataset="scerevisiae_gene_ensembl")
all(transcripts(yeast_txdb0) %in% transcripts(yeast_txdb))
all(transcripts(yeast_txdb) %in% transcripts(yeast_txdb0))

## -----------------------
## --- Ensembl Metazoa ---

## The metazoa mart is slow and at the same time it doesn't seem to
## support requests that take more than 1 min at the moment. So a call to
## biomaRt::getBM() will fail with a "Timeout was reached" error if the
## requested data takes more than 1 min to download. This unfortunately
## happens with the example below so we don't try to run it for now.

## Not run:
  mart <- useEnsemblGenomes(biomart="metazoa_mart")
  datasets <- listDatasets(mart)
  datasets$dataset
  worm_txdb <- makeTxDbFromBiomart(biomart="metazoa_mart",
                                   dataset="celegans_eg_gene",
                                   host="https://metazoa.ensembl.org")
  worm_txdb

  ## Note that even if the dataset for Worm on Ensembl Metazoa contains
  ## the same transcript as on the main Ensembl database, the transcript
  ## type might be annotated with slightly different terms (e.g. antisense
  ## vs antisense_RNA):
  filter <- list(tx_name="Y71G12B.44")
  transcripts(worm_txdb, filter=filter, columns=c("tx_name", "tx_type"))
  transcripts(txdb1, filter=filter, columns=c("tx_name", "tx_type"))

## End(Not run)
## ----------------------
## --- Ensembl Plants ---

## Like the metazoa mart (see above), the plants mart is also slow and
## doesn't seem to support requests that take more than 1 min either.
## So we don't try to run the example below for now.

## Not run:
mart <- useEnsemblGenomes(biomart="plants_mart")
datasets <- listDatasets(mart)
datasets[ , 1:2]
athaliana_txdb <- makeTxDbFromBiomart(biomart="plants_mart",
                                      dataset="athaliana_eg_gene",
                                      host="https://plants.ensembl.org")
athaliana_txdb

## End(Not run)
## ------------------------
## --- Ensembl Protists ---
```

```
mart <- useEnsemblGenomes(biomart="protists_mart")
datasets <- listDatasets(mart)
datasets$dataset
tgondii_txdb <- makeTxDbFromBiomart(biomart="protists_mart",
                                    dataset="tgondii_eg_gene",
                                    host="https://protists.ensembl.org")
tgondii_txdb

## ---------------------------------------------------------------------
## C. USING AN Ensembl MIRROR
## ---------------------------------------------------------------------

## You can use the 'host' argument to access the "ENSEMBL_MART_ENSEMBL"
## BioMart database at a mirror (e.g. at uswest.ensembl.org). A gotcha
## when doing this is that the name of the database on the mirror might
## be different! We can check this with listMarts() from the biomaRt
## package:
if (interactive()) {

  listMarts(host="https://useast.ensembl.org")

  txdb3 <- makeTxDbFromBiomart(biomart="ENSEMBL_MART_ENSEMBL",
                               dataset="hsapiens_gene_ensembl",
                               transcript_ids=transcript_ids,
                               host="https://useast.ensembl.org")
  txdb3
}
## Therefore in addition to setting 'host' to "uswest.ensembl.org", we
## might also need to specify the 'biomart' argument.


## ---------------------------------------------------------------------
## D. USING FILTERS
## ---------------------------------------------------------------------

## We can use listFilters() from the biomaRt package to get valid filter
## names:
mart <- useEnsembl(biomart="ENSEMBL_MART_ENSEMBL",
                dataset="hsapiens_gene_ensembl",
                host="https://www.ensembl.org")
head(listFilters(mart))

## Retrieve transcript dataset for Ensembl gene ENSG00000011198:
my_filter <- list(ensembl_gene_id="ENSG00000011198")

if (interactive()) {
  txdb4 <- makeTxDbFromBiomart(dataset="hsapiens_gene_ensembl",
                               filter=my_filter)
  txdb4
  transcripts(txdb4, columns=c("tx_id", "tx_name", "gene_id"))
  transcriptLengths(txdb4)
}
```

```
## ---------------------------------------------------------------------
## E. RETRIEVING CHROMOSOME INFORMATION ONLY
## ---------------------------------------------------------------------

chrominfo <- getChromInfoFromBiomart(dataset="celegans_gene_ensembl")
chrominfo
```

`makeTxDbFromEnsembl` *Make a TxDb object from an Ensembl database*

Description

The `makeTxDbFromEnsembl` function creates a TxDb object for a given organism by importing the genomic locations of its transcripts, exons, CDS, and genes from an Ensembl database.

Note that it uses the **RMariaDB** package internally so make sure that this package is installed.

Usage

```
makeTxDbFromEnsembl(organism="Homo sapiens",
                    release=NA,
                    circ_seqs=NULL,
                    server="ensembldb.ensembl.org",
                    username="anonymous", password=NULL, port=0L,
                    tx_attrib=NULL)
```

Arguments

`organism`	The *scientific name* (i.e. genus and species, or genus and species and subspecies) of the organism for which to import the data. Case is not sensitive. Underscores can be used instead of white spaces e.g. `"homo_sapiens"` is accepted.
`release`	The Ensembl release to query e.g. 89. If set to `NA` (the default), the current release is used.
`circ_seqs`	A character vector to list out which chromosomes should be marked as circular.
`server`	The name of the MySQL server to query. See `https://www.ensembl.org/info/data/mysql.html` for the list of Ensembl public MySQL servers. Make sure to use the server nearest to you. It can make a big difference!
`username`	Login username for the MySQL server.
`password`	Login password for the MySQL server.
`port`	Port of the MySQL server.
`tx_attrib`	If not `NULL`, only select transcripts with an attribute of the given code, a string, like `"gencode_basic"`.

Value

A TxDb object.

Note

makeTxDbFromEnsembl tends to be faster and more reliable than makeTxDbFromBiomart.

Author(s)

H. Pagès

See Also

- makeTxDbFromUCSC and makeTxDbFromBiomart for making a TxDb object from other online resources.
- makeTxDbFromGRanges and makeTxDbFromGFF for making a TxDb object from a GRanges object, or from a GFF or GTF file.
- The TxDb class.
- makeTxDb for the low-level function used by the makeTxDbFrom* functions to make the TxDb object returned to the user.

Examples

```
## Not run:
txdb <- makeTxDbFromEnsembl("Saccharomyces cerevisiae",
                            server="useastdb.ensembl.org")
txdb

## End(Not run)
```

---

makeTxDbFromGFF *Make a TxDb object from annotations available as a GFF3 or GTF file*

Description

The makeTxDbFromGFF function allows the user to make a TxDb object from transcript annotations available as a GFF3 or GTF file.

Usage

```
makeTxDbFromGFF(file,
                format=c("auto", "gff3", "gtf"),
                dataSource=NA,
                organism=NA,
                taxonomyId=NA,
                circ_seqs=NULL,
                chrominfo=NULL,
                miRBaseBuild=NA,
                metadata=NULL,
                dbxrefTag)
```

Arguments

file	Input GFF3 or GTF file. Can be a path to a file, or an URL, or a connection object, or a GFF3File or GTFFile object.
format	Format of the input file. Accepted values are: "auto" (the default) for auto-detection of the format, "gff3", or "gtf". Use "gff3" or "gtf" only if auto-detection failed.
dataSource	A single string describing the origin of the data file. Please be as specific as possible.
organism	What is the Genus and species of this organism. Please use proper scientific nomenclature for example: "Homo sapiens" or "Canis familiaris" and not "human" or "my fuzzy buddy". If properly written, this information may be used by the software to help you out later.
taxonomyId	By default this value is NA and the organism provided will be used to look up the correct value for this. But you can use this argument to override that and supply your own taxonomy id here (which will be separately validated). Since providing a valid taxonomy id will not require us to look up one based on your organism: this is one way that you can loosen the restrictions about what is and isn't a valid value for the organism.
circ_seqs	A character vector to list out which chromosomes should be marked as circular.
chrominfo	Data frame containing information about the chromosomes. Will be passed to the internal call to makeTxDb. See ?makeTxDb for more information. Alternatively, can be a Seqinfo object.
miRBaseBuild	Specify the string for the appropriate build Information from mirbase.db to use for microRNAs. This can be learned by calling supportedMiRBaseBuildValues. By default, this value will be set to NA, which will inactivate the microRNAs accessor.
metadata	A 2-column data frame containing meta information to be included in the TxDb object. See ?makeTxDb for more information about the format of metadata.
dbxrefTag	If not missing, the values in the Dbxref attribute with the specified tag (like "GeneID") are used for the feature names.

Details

makeTxDbFromGFF is a convenience function that feeds data from the parsed file to the makeTxDbFromGRanges function.

Value

A TxDb object.

Author(s)

M. Carlson and H. Pagès

See Also

- makeTxDbFromGRanges, which makeTxDbFromGFF is based on, for making a TxDb object from a GRanges object.
- The import function in the **rtracklayer** package (also used by makeTxDbFromGFF internally).
- makeTxDbFromUCSC, makeTxDbFromBiomart, and makeTxDbFromEnsembl, for making a TxDb object from online resources.
- The supportedMiRBaseBuildValues function for listing all the possible values for the miRBaseBuild argument.
- The TxDb class.
- makeTxDb for the low-level function used by the makeTxDbFrom* functions to make the TxDb object returned to the user.

Examples

```
## TESTING GFF3
gffFile <- system.file("extdata","GFF3_files","a.gff3",package="GenomicFeatures")
txdb <- makeTxDbFromGFF(file=gffFile,
            dataSource="partial gtf file for Tomatoes for testing",
            organism="Solanum lycopersicum")

## TESTING GTF, this time specifying the chrominfo
gtfFile <- system.file("extdata","GTF_files","Aedes_aegypti.partial.gtf",
                       package="GenomicFeatures")
chrominfo <- data.frame(chrom = c('supercont1.1','supercont1.2'),
                        length=c(5220442, 5300000),
                        is_circular=c(FALSE, FALSE))
metadata <- data.frame(name="Resource URL",
                       value=paste0("ftp://ftp.ensemblgenomes.org/pub/metazoa/",
                                    "release-13/gtf/aedes_aegypti/"))
txdb2 <- makeTxDbFromGFF(file=gtfFile,
             chrominfo=chrominfo,
             dataSource="ensemblgenomes",
             organism="Aedes aegypti",
             metadata=metadata)
```

---

makeTxDbFromGRanges *Make a TxDb object from a GRanges object*

Description

The makeTxDbFromGRanges function allows the user to extract gene, transcript, exon, and CDS information from a GRanges object structured as GFF3 or GTF, and to return that information in a TxDb object.

Usage

```
makeTxDbFromGRanges(gr, drop.stop.codons=FALSE, metadata=NULL, taxonomyId=NA)
```

Arguments

gr	A GRanges object structured as GFF3 or GTF, typically obtained with BiocIO::import().
drop.stop.codons	TRUE or FALSE. If TRUE, then features of type stop_codon are ignored. Otherwise (the default) the stop codons are considered to be part of the CDS and merged to them.
metadata	A 2-column data frame containing meta information to be included in the TxDb object. This data frame is just passed to makeTxDb, which makeTxDbFromGRanges calls at the end to make the TxDb object from the information extracted from gr. See ?makeTxDb for more information about the format of metadata.
taxonomyId	By default this value is NA which will result in an NA field since there is no reliable way to infer this from a GRanges object. But you can use this argument to supply your own valid taxId here and if you do, then the Organism can be filled in as well

Value

A TxDb object.

Author(s)

Hervé Pagès

See Also

- makeTxDbFromUCSC, makeTxDbFromBiomart, and makeTxDbFromEnsembl, for making a TxDb object from online resources.
- makeTxDbFromGFF for making a TxDb object from a GFF or GTF file.
- The import generic function in the **BiocIO** package.
- The asGFF method for TxDb objects (asGFF,TxDb-method) for the reverse of makeTxDbFromGRanges, that is, for turning a TxDb object into a GRanges object structured as GFF.
- The TxDb class.
- makeTxDb for the low-level function used by the makeTxDbFrom* functions to make the TxDb object returned to the user.

Examples

```
library(BiocIO)  # for the import() function

## ---------------------------------------------------------------------
## WITH A GRanges OBJECT STRUCTURED AS GFF3
## ---------------------------------------------------------------------
GFF3_files <- system.file("extdata", "GFF3_files",
                          package="GenomicFeatures")

path <- file.path(GFF3_files, "a.gff3")
gr <- import(path)
```

```
txdb <- makeTxDbFromGRanges(gr)
txdb

## Reverse operation:
gr2 <- asGFF(txdb)

## Sanity check (asGFF() does not propagate the CDS phase at the moment):
target <- as.list(txdb)
target$splicings$cds_phase <- NULL
stopifnot(identical(target, as.list(makeTxDbFromGRanges(gr2))))

## ---------------------------------------------------------------------
## WITH A GRanges OBJECT STRUCTURED AS GTF
## ---------------------------------------------------------------------
GTF_files <- system.file("extdata", "GTF_files", package="GenomicFeatures")

## test1.gtf was grabbed from http://mblab.wustl.edu/GTF22.html (5 exon
## gene with 3 translated exons):
path <- file.path(GTF_files, "test1.gtf")
gr <- import(path)
txdb <- makeTxDbFromGRanges(gr)
txdb

path <- file.path(GTF_files, "Aedes_aegypti.partial.gtf")
gr <- import(path)
txdb <- makeTxDbFromGRanges(gr)
txdb
```

`makeTxDbFromUCSC` *Make a TxDb object from annotations available at the UCSC Genome Browser*

Description

The `makeTxDbFromUCSC` function allows the user to make a TxDb object from transcript annotations available at the UCSC Genome Browser.

Note that it uses the **RMariaDB** package internally so make sure that this package is installed.

Usage

```
makeTxDbFromUCSC(genome="hg19", tablename="knownGene",
        transcript_ids=NULL,
        circ_seqs=NULL,
        url="http://genome.ucsc.edu/cgi-bin/",
        goldenPath.url=getOption("UCSC.goldenPath.url"),
        taxonomyId=NA,
        miRBaseBuild=NA)

supportedUCSCtables(genome="hg19", url="http://genome.ucsc.edu/cgi-bin/")
```

```
browseUCSCtrack(genome="hg19", tablename="knownGene",
                url="http://genome.ucsc.edu/cgi-bin/")
```

Arguments

genome	The name of a UCSC genome assembly e.g. "hg19" or "panTro6". You can use rtracklayer::ucscGenomes()[, "db"] to obtain the current list of valid UCSC genome assemblies.
tablename	The name of the UCSC table containing the transcript genomic locations to retrieve. Use the supportedUCSCtables utility function to get the list of tables known to work with makeTxDbFromUCSC.
transcript_ids	Optionally, only retrieve transcript locations for the specified set of transcript ids. If this is used, then the meta information displayed for the resulting TxDb object will say 'Full dataset: no'. Otherwise it will say 'Full dataset: yes'.
circ_seqs	Like GRanges objects, SummarizedExperiment objects, and many other objects in Bioconductor, the TxDb object returned by makeTxDbFromUCSC contains a seqinfo component that can be accessed with seqinfo(). This component contains various sequence-level information like the sequence names, lengths, and circularity flag for the genome assembly of the TxDb object. As far as we know the information of which sequences are circular is not available in the UCSC Genome Browser. However, for the most commonly used UCSC genome assemblies makeTxDbFromUCSC will get this information from a knowledge database stored in the **GenomeInfoDb** package (see ?registered_UCSC_genomes). For less commonly used UCSC genome assemblies, makeTxDbFromUCSC will make a guess based on the chromosome names (e.g. chrM or 2micron will be assumed to be circular). Even though this works most of the time, it is not guaranteed to work *all the time*. So in this case a warning is issued. If you think the guess is incorrect then you can supply your own list of circular sequences (as a character vector) via the circ_seqs argument.
url,goldenPath.url	Use to specify the location of an alternate UCSC Genome Browser.
taxonomyId	By default this value is NA and the organism inferred will be used to look up the correct value for this. But you can use this argument to supply your own valid taxId here.
miRBaseBuild	Specify the string for the appropriate build information from **mirbase.db** to use for microRNAs. This can be learned by calling supportedMiRBaseBuildValues. By default, this value will be set to NA, which will inactivate the microRNAs accessor.

Details

makeTxDbFromUCSC is a convenience function that feeds data from the UCSC source to the lower level makeTxDb function. See ?makeTxDbFromEnsembl for a similar function that feeds data from an Ensembl database.

Value

For makeTxDbFromUCSC: A TxDb object.

For supportedUCSCtables: A data frame with 3 columns (tablename, track, and subtrack) and 1 row per table known to work with makeTxDbFromUCSC. IMPORTANT NOTE: In the returned data frame, the set of tables associated with a track with subtracks might contain tables that don't exist for the specified genome.

Author(s)

M. Carlson and H. Pagès

See Also

- makeTxDbFromEnsembl and makeTxDbFromBiomart for making a TxDb object from other online resources.
- makeTxDbFromGRanges and makeTxDbFromGFF for making a TxDb object from a GRanges object, or from a GFF or GTF file.
- ucscGenomes in the **rtracklayer** package.
- The supportedMiRBaseBuildValues function for listing all the possible values for the miRBaseBuild argument.
- The TxDb class.
- makeTxDb for the low-level function used by the makeTxDbFrom* functions to make the TxDb object returned to the user.

Examples

```
## ---------------------------------------------------------------------
## A. BASIC USAGE
## ---------------------------------------------------------------------

## Use ucscGenomes() from the rtracklayer package to display the list of
## genomes available at UCSC:
library(rtracklayer)
ucscGenomes()[ , "db"]

## Display the list of tables known to work with makeTxDbFromUCSC():
supportedUCSCtables("hg38")
supportedUCSCtables("hg19")

## Open the UCSC track page for a given organism/table:
browseUCSCtrack("hg38", tablename="knownGene")
browseUCSCtrack("hg19", tablename="knownGene")

browseUCSCtrack("hg38", tablename="ncbiRefSeqSelect")
browseUCSCtrack("hg19", tablename="ncbiRefSeqSelect")

browseUCSCtrack("hg19", tablename="pseudoYale60")

browseUCSCtrack("sacCer3", tablename="ensGene")
```

```
## Retrieve a full transcript dataset for Yeast from UCSC:
txdb1 <- makeTxDbFromUCSC("sacCer3", tablename="ensGene")
txdb1

## Retrieve an incomplete transcript dataset for Mouse from UCSC (only
## transcripts linked to Entrez Gene ID 22290):
transcript_ids <- c(
    "uc009uzf.1",
    "uc009uzg.1",
    "uc009uzh.1",
    "uc009uzi.1",
    "uc009uzj.1"
)

txdb2 <- makeTxDbFromUCSC("mm10", tablename="knownGene",
                          transcript_ids=transcript_ids)
txdb2

## ---------------------------------------------------------------------
## B. IMPORTANT NOTE ABOUT supportedUCSCtables()
## ---------------------------------------------------------------------

## In the data frame returned by supportedUCSCtables(), the set of
## tables associated with a track with subtracks might contain tables
## that don't exist for the specified genome:
supportedUCSCtables("mm10")
browseUCSCtrack("mm10", tablename="ncbiRefSeqSelect")  # no such table
```

`makeTxDbPackage` *Making a TxDb package from annotations available at the UCSC Genome Browser, biomaRt or from another source.*

Description

A TxDb package is an annotation package containing a TxDb object.

The `makeTxDbPackageFromUCSC` function allows the user to make a TxDb package from transcript annotations available at the UCSC Genome Browser.

The `makeTxDbPackageFromBiomart` function allows the user to do the same thing as `makeTxDbPackageFromUCSC` except that the annotations originate from biomaRt.

Finally, the `makeTxDbPackage` function allows the user to make a TxDb package directly from a TxDb object.

Usage

```
makeTxDbPackageFromUCSC(
    version=,
    maintainer,
```

```
    author,
    destDir=".",
    license="Artistic-2.0",
    genome="hg19",
    tablename="knownGene",
    transcript_ids=NULL,
    circ_seqs=NULL,
    url="http://genome.ucsc.edu/cgi-bin/",
    goldenPath.url=getOption("UCSC.goldenPath.url"),
    taxonomyId=NA,
    miRBaseBuild=NA)

makeFDbPackageFromUCSC(
    version,
    maintainer,
    author,
    destDir=".",
    license="Artistic-2.0",
    genome="hg19",
    track="tRNAs",
    tablename="tRNAs",
    columns = UCSCFeatureDbTableSchema(genome, track, tablename),
    url="http://genome.ucsc.edu/cgi-bin/",
    goldenPath.url=getOption("UCSC.goldenPath.url"),
    chromCol=NULL,
    chromStartCol=NULL,
    chromEndCol=NULL,
    taxonomyId=NA)

makeTxDbPackageFromBiomart(
    version,
    maintainer,
    author,
    destDir=".",
    license="Artistic-2.0",
    biomart="ENSEMBL_MART_ENSEMBL",
    dataset="hsapiens_gene_ensembl",
    transcript_ids=NULL,
    circ_seqs=NULL,
    filter=NULL,
    id_prefix="ensembl_",
    host="https://www.ensembl.org",
    port,
    taxonomyId=NA,
    miRBaseBuild=NA)

makeTxDbPackage(txdb,
                version,
```

```
                maintainer,
                author,
                destDir=".",
                license="Artistic-2.0",
                pkgname=NULL,
                provider=NULL,
                providerVersion=NULL)

supportedMiRBaseBuildValues()

makePackageName(txdb)
```

Arguments

version	What is the version number for this package?
maintainer	Who is the package maintainer? (must include email to be valid). Should be a person object, or something coercible to one, like a string. May be omitted if the author argument is a person containing someone with the maintainer role.
author	Who is the creator of this package? Should be a person object, or something coercible to one, like a character vector of names. The maintainer argument will be merged into this list.
destDir	A path where the package source should be assembled.
license	What is the license (and it's version)
biomart	which BioMart database to use. Get the list of all available BioMart databases with the listMarts function from the biomaRt package. See the details section below for a list of BioMart databases with compatible transcript annotations.
dataset	which dataset from BioMart. For example: "hsapiens_gene_ensembl", "mmusculus_gene_ensembl", "dmelanogaster_gene_ensembl", "celegans_gene_ensembl", "scerevisiae_gene_ensembl", etc in the ensembl database. See the examples section below for how to discover which datasets are available in a given BioMart database.
genome	genome abbreviation used by UCSC and obtained by ucscGenomes()[, "db"]. For example: "hg18".
track	name of the UCSC track. Use supportedUCSCFeatureDbTracks to get the list of available tracks for a particular genome
tablename	name of the UCSC table containing the transcript annotations to retrieve. Use the supportedUCSCtables utility function to get the list of tables known to work with makeTxDbFromUCSC.
transcript_ids	optionally, only retrieve transcript annotation data for the specified set of transcript ids. If this is used, then the meta information displayed for the resulting TxDb object will say 'Full dataset: no'. Otherwise it will say 'Full dataset: yes'.
circ_seqs	a character vector to list out which chromosomes should be marked as circular.
filter	Additional filters to use in the BioMart query. Must be a named list. An example is filter=as.list(c(source="entrez"))
host	The host URL of the BioMart. Defaults to https://www.ensembl.org.

port — The port to use in the HTTP communication with the host. This argument has been deprecated. It is handled by useEnsembl depending on the host input.

id_prefix — Specifies the prefix used in BioMart attributes. For example, some BioMarts may have an attribute specified as "ensembl_transcript_id" whereas others have the same attribute specified as "transcript_id". Defaults to "ensembl_".

columns — a named character vector to list out the names and types of the other columns that the downloaded track should have. Use UCSCFeatureDbTableSchema to retrieve this information for a particular table.

url,goldenPath.url — use to specify the location of an alternate UCSC Genome Browser.

chromCol — If the schema comes back and the 'chrom' column has been labeled something other than 'chrom', use this argument to indicate what that column has been labeled as so we can properly designate it. This could happen (for example) with the knownGene track tables, which has no 'chromStart' or 'chromEnd' columns, but which DOES have columns that could reasonably substitute for these columns under particular circumstances. Therefore we allow these three columns to have arguments so that their definition can be re-specified

chromStartCol — Same thing as chromCol, but for renames of 'chromStart'

chromEndCol — Same thing as chromCol, but for renames of 'chromEnd'

txdb — A TxDb object that represents a handle to a transcript database. This object type is what is returned by makeTxDbFromUCSC, makeTxDbFromUCSC or makeTxDb

taxonomyId — By default this value is NA and the organism provided (or inferred) will be used to look up the correct value for this. But you can use this argument to override that and supply your own valid taxId here

miRBaseBuild — specify the string for the appropriate build Information from mirbase.db to use for microRNAs. This can be learned by calling supportedMiRBaseBuildValues. By default, this value will be set to NA, which will inactivate the microRNAs accessor.

pkgname — By default this value is NULL and does not need to be filled in (a package name will be generated for you). But if you override this value, then the package and it's object will be instead named after this value. Be aware that the standard rules for package names will apply, (so don't include spaces, underscores or dashes)

provider — If not given, a default is taken from the 'Data source' field of the metadata table.

providerVersion — If not given, a default is taken from one of 'UCSC table', 'BioMart version' or 'Data source' fields of the metadata table.

Details

makeTxDbPackageFromUCSC is a convenience function that calls both the makeTxDbFromUCSC and the makeTxDbPackage functions. The makeTxDbPackageFromBiomart follows a similar pattern and calls the makeTxDbFromBiomart and makeTxDbPackage functions. supportedMiRBaseBuildValues is a convenience function that will list all the possible values for the miRBaseBuild argument. makePackageName creates a package name from a TxDb object. This function is also used by OrganismDbi.

Value

A TxDb object.

Author(s)

M. Carlson

See Also

makeTxDbFromUCSC, makeTxDbFromBiomart, makeTxDb, ucscGenomes

Examples

```
## First consider relevant helper/discovery functions:
## Get the list of tables known to work with makeTxDbPackageFromUCSC():
supportedUCSCtables(genome="hg19")

## Can also list all the possible values for the miRBaseBuild argument:
supportedMiRBaseBuildValues()

## Next are examples of actually building a package:
## Not run:
## Makes a transcript package for Yeast from the ensGene table at UCSC:
makeTxDbPackageFromUCSC(version="0.01",
                        maintainer="Some One <so@someplace.org>",
                        author="Some One <so@someplace.com>",
                        genome="sacCer2",
                        tablename="ensGene")

## Makes a transcript package from Human by using biomaRt and limited to a
## small subset of the transcripts.
transcript_ids <- c(
    "ENST00000400839",
    "ENST00000400840",
    "ENST00000478783",
    "ENST00000435657",
    "ENST00000268655",
    "ENST00000313243",
    "ENST00000341724")

makeTxDbPackageFromBiomart(version="0.01",
                           maintainer="Some One <so@someplace.org>",
                           author="Some One <so@someplace.com>",
                           transcript_ids=transcript_ids)


## End(Not run)
```

mapIdsToRanges *Map IDs to Genomic Ranges*

Description

Map IDs to Genomic Ranges

Usage

```
mapIdsToRanges(x, ...)

## S4 method for signature 'TxDb'
mapIdsToRanges(x, keys, type = c("cds", "exon", "tx",
  "gene"), columns = NULL)
```

Arguments

x	Database to use for mapping
keys	Values to lookup, passed to transcripts et. al.
type	Types of feature to return
columns	Additional metadata columns to include in the output
...	Additional arguments passed to methods

Value

GRangesList corresponding to the keys

Methods (by class)

- TxDb: TxDb method

Examples

```
fl <- system.file(package = "GenomicFeatures", "extdata", "sample_ranges.rds")
txdb <- makeTxDbFromGRanges(readRDS(fl))

keys <- list(tx_name = c("ENST00000371582", "ENST00000371588",
    "ENST00000494752", "ENST00000614008", "ENST00000496771"))
mapIdsToRanges(txdb, keys = keys, type = "tx")
```

mapRangesToIds *Map Genomic Ranges to IDs*

Description

Map Genomic Ranges to IDs

Usage

```
mapRangesToIds(x, ...)

## S4 method for signature 'TxDb'
mapRangesToIds(x, ranges, type = c("cds", "exon", "tx",
  "gene"), columns = NULL, ...)
```

Arguments

x	Database to use for mapping
ranges	range object used to subset
type	of feature to return
columns	additional metadata columns to include in the output.
...	Additional arguments passed to findOverlaps

Value

DataFrame of mcols from the database.

Methods (by class)

- TxDb: TxDb method

Examples

```
fl <- system.file(package = "GenomicFeatures", "extdata", "sample_ranges.rds")
txdb <- makeTxDbFromGRanges(readRDS(fl))

keys <- list(tx_name = c("ENST00000371582", "ENST00000371588",
    "ENST00000494752", "ENST00000614008", "ENST00000496771"))
res <- mapIdsToRanges(txdb, keys = keys, type = "tx")
mapRangesToIds(txdb, res, "tx")
```

mapToTranscripts *Map range coordinates between transcripts and genome space*

Description

Map range coordinates between features in the transcriptome and genome (reference) space.

See ?mapToAlignments in the **GenomicAlignments** package for mapping coordinates between reads (local) and genome (reference) space using a CIGAR alignment.

Usage

```
## mapping to transcripts
## S4 method for signature 'GenomicRanges,GenomicRanges'
mapToTranscripts(x, transcripts,
          ignore.strand = FALSE)
## S4 method for signature 'GenomicRanges,GRangesList'
mapToTranscripts(x, transcripts,
          ignore.strand = FALSE, intronJunctions=FALSE)
## S4 method for signature 'ANY,TxDb'
mapToTranscripts(x, transcripts, ignore.strand = FALSE,
          extractor.fun = GenomicFeatures::transcripts, ...)
## S4 method for signature 'GenomicRanges,GRangesList'
pmapToTranscripts(x, transcripts,
          ignore.strand = FALSE)

## mapping from transcripts
## S4 method for signature 'GenomicRanges,GRangesList'
mapFromTranscripts(x, transcripts,
          ignore.strand = FALSE)
## S4 method for signature 'GenomicRanges,GRangesList'
pmapFromTranscripts(x, transcripts,
          ignore.strand = FALSE)
## S4 method for signature 'IntegerRanges,GRangesList'
pmapFromTranscripts(x, transcripts)
```

Arguments

x	GenomicRanges object of positions to be mapped. The seqnames of x are used in mapFromTranscripts, i.e., when mapping from transcripts to the genome. In the case of pmapFromTranscripts, x can be an IntegerRanges object.
transcripts	A named GenomicRanges or GRangesList object used to map between x and the result. The ranges can be any feature in the transcriptome extracted from a TxDb (e.g., introns, exons, cds regions). See ?transcripts and ?transcriptsBy for a list of extractor functions. The transcripts object must have names. When mapping from transcripts to the genome, they are used to determine mapping pairs; in the reverse direction they become the seqlevels of the output object.

ignore.strand When ignore.strand is TRUE, strand is ignored in overlaps operations (i.e., all strands are considered "+") and the strand in the output is '*'.

When ignore.strand is FALSE strand in the output is taken from the transcripts argument. When transcripts is a GRangesList, all inner list elements of a common list element must have the same strand or an error is thrown.

Mapped position is computed by counting from the transcription start site (TSS) and is not affected by the value of ignore.strand.

intronJunctions

Logical to indicate if intronic ranges in x should be reported.

This argument is only supported in mapToTranscripts when transcripts is a GRangesList. When transcripts is a GRangesList, individual ranges can be thought of as exons and the spaces between the ranges as introns.

When intronJunctions=TRUE, ranges that fall completely "within" an intron are reported as a zero-width range (start and end are taken from the ranges they fall between). A metadata column called "intronic" is returned with the GRanges and marked as TRUE for these ranges. By default, intronJunctions=FALSE and these ranges are not mapped.

Ranges that have either the start or end in an intron are considered "non hits" and are never mapped. Ranges that span introns are always mapped. Neither of these range types are controlled by the intronJunctions argument.

extractor.fun Function to extract genomic features from a TxDb object.

This argument is only applicable to mapToTranscripts when transcripts is a TxDb object. The extractor should be the name of a function (not a character()) described on the ?transcripts, ?transcriptsBy, or ?microRNAs man page.

Valid extractor functions:

- transcripts ## default
- exons
- cds
- genes
- promoters
- exonicParts
- intronicParts
- transcriptsBy
- exonsBy
- cdsBy
- intronsByTranscript
- fiveUTRsByTranscript
- threeUTRsByTranscript
- microRNAs
- tRNAs

... Additional arguments passed to extractor.fun functions.

Details

In GenomicFeatures >= 1.21.10, the default for `ignore.strand` was changed to `FALSE` for consistency with other methods in the **GenomicRanges** and **GenomicAlignments** packages. Additionally, the mapped position is computed from the TSS and does not depend on the `ignore.strand` argument. See the section on `ignore.strand` for details.

- `mapToTranscripts`, `pmapToTranscripts` The genomic range in `x` is mapped to the local position in the `transcripts` ranges. A successful mapping occurs when `x` is completely within the `transcripts` range, equivalent to:

  ```
  findOverlaps(..., type="within")
  ```

  Transcriptome-based coordinates start counting at 1 at the beginning of the `transcripts` range and return positions where `x` was aligned. The seqlevels of the return object are taken from the `transcripts` object and should be transcript names. In this direction, mapping is attempted between all elements of `x` and all elements of `transcripts`.

  `mapToTranscripts` uses `findOverlaps` to map ranges in `x` to ranges in `transcripts`. This method does not return unmapped ranges.

  `pmapToTranscripts` maps the i-th range in `x` to the i-th range in `transcripts`. Recycling is supported for both `x` and `transcripts` when either is length == 1L; otherwise the lengths must match. Ranges in `x` that do not map (out of bounds or strand mismatch) are returned as zero-width ranges starting at 0. These ranges are given the seqname of "UNMAPPED".

- `mapFromTranscripts`, `pmapFromTranscripts` The transcript-based position in `x` is mapped to genomic coordinates using the ranges in `transcripts`. A successful mapping occurs when the following is TRUE:

  ```
  width(transcripts) >= start(x) + width(x)
  ```

  `x` is aligned to `transcripts` by moving in `start(x)` positions in from the beginning of the `transcripts` range. The seqlevels of the return object are chromosome names.

  `mapFromTranscripts` uses the seqname of `x` and the names of `transcripts` to determine mapping pairs (vs attempting to match all possible pairs). Name matching is motivated by use cases such as differentially expressed regions where the expressed regions in `x` would only be related to a subset of regions in `transcripts`. This method does not return unmapped ranges.

  `pmapFromTranscripts` maps the i-th range in x to the i-th range in `transcripts` and therefore does not use name matching. Recycling is supported in `pmapFromTranscripts` when either `x` or `transcripts` is length == 1L; otherwise the lengths must match. Ranges in `x` that do not map (out of bounds or strand mismatch) are returned as zero-width ranges starting at 0. These ranges are given the seqname of "UNMAPPED".

Value

`pmapToTranscripts` returns a `GRanges` the same length as `x`.

`pmapFromTranscripts` returns a `GRanges` when `transcripts` is a `GRanges` and a `GRangesList` when `transcripts` is a `GRangesList`. In both cases the return object is the same length as `x`. The rational for returning the `GRangesList` is to preserve exon structure; ranges in a list element that are not overlapped by `x` are returned as a zero-width range. The `GRangesList` return object will have no seqlevels called "UNMAPPED"; those will only occur when a `GRanges` is returned.

mapToTranscripts and mapFromTranscripts return GRanges objects that vary in length similar to a Hits object. The result contains mapped records only; strand mismatch and out of bound ranges are not returned. xHits and transcriptsHits metadata columns (similar to the queryHits and subjectHits of a Hits object) indicate elements of x and transcripts used in the mapping.

When intronJunctions is TRUE, mapToTranscripts returns an extra metdata column named intronic to identify the intron ranges.

When mapping to transcript coordinates, seqlevels of the output are the names on the transcripts object and most often these will be transcript names. When mapping to the genome, seqlevels of the output are the seqlevels of transcripts which are usually chromosome names.

Author(s)

V. Obenchain, M. Lawrence and H. Pagès

See Also

- ?mapToAlignments in the **GenomicAlignments** package for methods mapping between reads and genome space using a CIGAR alignment.

Examples

```
## ---------------------------------------------------------------------
## A. Basic Use: Conversion between CDS and Exon coordinates and the
##    genome
## ---------------------------------------------------------------------

## Gene "Dgkb" has ENTREZID "217480":
library(org.Mm.eg.db)
Dgkb_geneid <- get("Dgkb", org.Mm.egSYMBOL2EG)

## The gene is on the positive strand, chromosome 12:
library(TxDb.Mmusculus.UCSC.mm10.knownGene)
txdb <- TxDb.Mmusculus.UCSC.mm10.knownGene
tx_by_gene <- transcriptsBy(txdb, by="gene")
Dgkb_transcripts <- tx_by_gene[[Dgkb_geneid]]
Dgkb_transcripts  # all 7 Dgkb transcripts are on chr12, positive strand

## To map coordinates from local CDS or exon space to genome
## space use mapFromTranscripts().

## When mapping CDS coordinates to genome space the 'transcripts'
## argument is the collection of CDS regions by transcript.
coord <- GRanges("chr12", IRanges(4, width=1))
## Get the names of the transcripts in the gene:
Dgkb_tx_names <- mcols(Dgkb_transcripts)$tx_name
Dgkb_tx_names
## Use these names to isolate the region of interest:
cds_by_tx <- cdsBy(txdb, "tx", use.names=TRUE)
Dgkb_cds_by_tx <- cds_by_tx[intersect(Dgkb_tx_names, names(cds_by_tx))]
## Dgkb CDS grouped by transcript (no-CDS transcripts omitted):
Dgkb_cds_by_tx
```

```
lengths(Dgkb_cds_by_tx)  # nb of CDS per transcript
## A requirement for mapping from transcript space to genome space
## is that seqnames in 'x' match the names in 'transcripts'.
names(Dgkb_cds_by_tx) <- rep(seqnames(coord), length(Dgkb_cds_by_tx))
## There are 6 results, one for each transcript.
mapFromTranscripts(coord, Dgkb_cds_by_tx)

## To map exon coordinates to genome space the 'transcripts'
## argument is the collection of exon regions by transcript.
coord <- GRanges("chr12", IRanges(100, width=1))
ex_by_tx <- exonsBy(txdb, "tx", use.names=TRUE)
Dgkb_ex_by_tx <- ex_by_tx[Dgkb_tx_names]
names(Dgkb_ex_by_tx) <- rep(seqnames(coord), length(Dgkb_ex_by_tx))
## Again the output has 6 results, one for each transcript.
mapFromTranscripts(coord, Dgkb_ex_by_tx)

## To go the reverse direction and map from genome space to
## local CDS or exon space, use mapToTranscripts().

## Genomic position 37981944 maps to CDS position 4:
coord <- GRanges("chr12", IRanges(37981944, width=1))
mapToTranscripts(coord, Dgkb_cds_by_tx)

## Genomic position 37880273 maps to exon position 100:
coord <- GRanges("chr12", IRanges(37880273, width=1))
mapToTranscripts(coord, Dgkb_ex_by_tx)


## The following examples use more than 2GB of memory, which is more
## than what 32-bit Windows can handle:
is_32bit_windows <- .Platform$OS.type == "windows" &&
                    .Platform$r_arch == "i386"
if (!is_32bit_windows) {
## ---------------------------------------------------------------------
## B. Map sequence locations in exons to the genome
## ---------------------------------------------------------------------

## NAGNAG alternative splicing plays an essential role in biological
## processes and represents a highly adaptable system for
## posttranslational regulation of gene function. The majority of
## NAGNAG studies largely focus on messenger RNA. A study by Sun,
## Lin, and Yan (http://www.hindawi.com/journals/bmri/2014/736798/)
## demonstrated that NAGNAG splicing is also operative in large
## intergenic noncoding RNA (lincRNA). One finding of interest was
## that linc-POLR3G-10 exhibited two NAGNAG acceptors located in two
## distinct transcripts: TCONS_00010012 and TCONS_00010010.

## Extract the exon coordinates of TCONS_00010012 and TCONS_00010010:
lincrna <- c("TCONS_00010012", "TCONS_00010010")
library(TxDb.Hsapiens.UCSC.hg19.lincRNAsTranscripts)
txdb <- TxDb.Hsapiens.UCSC.hg19.lincRNAsTranscripts
exons <- exonsBy(txdb, by="tx", use.names=TRUE)[lincrna]
exons
```

```
## The two NAGNAG acceptors were identified in the upstream region of
## the fourth and fifth exons located in TCONS_00010012.
## Extract the sequences for transcript TCONS_00010012:
library(BSgenome.Hsapiens.UCSC.hg19)
genome <- BSgenome.Hsapiens.UCSC.hg19
exons_seq <- getSeq(genome, exons[[1]])

## TCONS_00010012 has 4 exons:
exons_seq

## The most common triplet among the lincRNA sequences was CAG. Identify
## the location of this pattern in all exons.
cag_loc <- vmatchPattern("CAG", exons_seq)

## Convert the first occurance of CAG in each exon back to genome
## coordinates.
first_loc <- do.call(c, sapply(cag_loc, "[", 1, simplify=TRUE))
pmapFromTranscripts(first_loc, exons[[1]])

## ---------------------------------------------------------------------
## C. Map dbSNP variants to CDS or cDNA coordinates
## ---------------------------------------------------------------------

## The GIPR gene encodes a G-protein coupled receptor for gastric
## inhibitory polypeptide (GIP). Originally GIP was identified to
## inhibited gastric acid secretion and gastrin release but was later
## demonstrated to stimulate insulin release in the presence of elevated
## glucose.

## In this example 5 SNPs located in the GIPR gene are mapped to cDNA
## coordinates. A list of SNPs in GIPR can be downloaded from dbSNP or
## NCBI.
rsids <- c("rs4803846", "rs139322374", "rs7250736", "rs7250754",
           "rs9749185")

## Extract genomic coordinates with a SNPlocs package.
library(SNPlocs.Hsapiens.dbSNP144.GRCh38)
snps <- snpsById(SNPlocs.Hsapiens.dbSNP144.GRCh38, rsids)

## Gene regions of GIPR can be extracted from a TxDb package of
## compatible build. The TxDb package uses Entrez gene identifiers
## and GIPR is a gene symbol. Let's first lookup its Entrez gene ID.
library(org.Hs.eg.db)
GIPR_geneid <- get("GIPR", org.Hs.egSYMBOL2EG)

## The transcriptsBy() extractor returns a range for each transcript that
## includes the UTR and exon regions (i.e., cDNA).
library(TxDb.Hsapiens.UCSC.hg38.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg38.knownGene
tx_by_gene <- transcriptsBy(txdb, "gene")
GIPR_transcripts <- tx_by_gene[GIPR_geneid]
GIPR_transcripts  # all 8 GIPR transcripts are on chr19, positive strand
```

```
## Before mapping, the chromosome names (seqlevels) in the two
## objects must be harmonized. The style is NCBI for 'snps' and
## UCSC for 'GIPR_transcripts'.
seqlevelsStyle(snps)
seqlevelsStyle(GIPR_transcripts)

## Modify the style (and genome) in 'snps' to match 'GIPR_transcripts'.
seqlevelsStyle(snps) <- seqlevelsStyle(GIPR_transcripts)

## The 'GIPR_transcripts' object is a GRangesList of length 1. This single
## list element contains the cDNA range for 8 different transcripts. To
## map to each transcript individually 'GIPR_transcripts' must be unlisted
## before mapping.

## Map all 5 SNPS to all 8 transcripts:
mapToTranscripts(snps, unlist(GIPR_transcripts))

## Map the first SNP to transcript "ENST00000590918.5" and the second to
## "ENST00000263281.7".
pmapToTranscripts(snps[1:2], unlist(GIPR_transcripts)[1:2])

## The cdsBy() extractor returns coding regions by gene or by transcript.
## Extract the coding regions for transcript "ENST00000263281.7".
cds <- cdsBy(txdb, "tx", use.names=TRUE)["ENST00000263281.7"]
cds

## The 'cds' object is a GRangesList of length 1 containing all CDS ranges
## for the single transcript "ENST00000263281.7".

## To map to the concatenated group of ranges leave 'cds' as a GRangesList.
mapToTranscripts(snps, cds)

## Only the second SNP could be mapped. Unlisting the 'cds' object maps
## the SNPs to the individual cds ranges (vs the concatenated range).
mapToTranscripts(snps[2], unlist(cds))

## The location is the same because the SNP hit the first CDS range. If
## the transcript were on the "-" strand the difference in concatenated
## vs non-concatenated position would be more obvious.

## Change strand:
strand(cds) <- strand(snps) <- "-"
mapToTranscripts(snps[2], unlist(cds))
}
```

`microRNAs` *Extract microRNA or tRNA genomic ranges from an object*

Description

Generic functions to extract microRNA or tRNA genomic ranges from an object. This page documents the methods for TxDb objects only.

Usage

```
microRNAs(x)
## S4 method for signature 'TxDb'
microRNAs(x)

tRNAs(x)
## S4 method for signature 'TxDb'
tRNAs(x)
```

Arguments

x	A TxDb object.

Value

A GRanges object.

Author(s)

M. Carlson

See Also

- transcripts, transcriptsBy, and transcriptsByOverlaps for the core genomic features extractors.
- The TxDb class.

Examples

```
## Not run: library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(mirbase.db)
microRNAs(TxDb.Hsapiens.UCSC.hg19.knownGene)

## End(Not run)
```

nearest-methods *Finding the nearest genomic range neighbor in a TxDb*

Description

The distance methods for TxDb objects and subclasses.

Usage

```
## S4 method for signature 'GenomicRanges,TxDb'
distance(x, y, ignore.strand=FALSE,
    ..., id, type=c("gene", "tx", "exon", "cds"))
```

Arguments

x	The query GenomicRanges instance.
y	For distance, a TxDb instance. The id is used to extract ranges from the TxDb which are then used to compute the distance from x.
id	A character vector the same length as x. The id must be identifiers in the TxDb object. type indicates what type of identifier id is.
type	A character(1) describing the id. Must be one of 'gene', 'tx', 'exon' or 'cds'.
ignore.strand	A logical indicating if the strand of the ranges should be ignored. When TRUE, strand is set to '+'.
...	Additional arguments for methods.

Details

- distance: Returns the distance for each range in x to the range extracted from the TxDb object y. Values in id are matched to one of 'gene_id', 'tx_id', 'exon_id' or 'cds_id' identifiers in the TxDb and the corresponding ranges are extracted. The type argument specifies which identifier is represented in id. The extracted ranges are used in the distance calculation with the ranges in x.

  The method returns NA values when the genomic region defined by id cannot be collapsed into a single range (e.g., when a gene spans multiple chromosomes) or if the id is not found in y.

  The behavior of distance with respect to zero-width ranges has changed in Bioconductor 2.12. See the man page ?distance in IRanges for details.

Value

For distance, an integer vector of distances between the ranges in x and y.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

- nearest-methods man page in IRanges.
- nearest-methods man page in GenomicRanges.

Examples

```
## ---------------------------------------------------------
## distance()
## ---------------------------------------------------------

library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
txdb <- TxDb.Dmelanogaster.UCSC.dm3.ensGene
gr <- GRanges(c("chr2L", "chr2R"),
              IRanges(c(100000, 200000),  width=100))
distance(gr, txdb, id=c("FBgn0259717", "FBgn0261501"), type="gene")
distance(gr, txdb, id=c("10000", "23000"), type="cds")

## The id's must be in the appropriate order with respect to 'x'.
distance(gr, txdb, id=c("4", "4097"), type="tx")

## 'id' "4" is on chr2L and "4097" is on chr2R.
transcripts(txdb, filter=list(tx_id=c("4", "4097")))

## If we reverse the 'id' the chromosomes are incompatable with gr.
distance(gr, txdb, id=c("4097", "4"), type="tx")

## distance() compares each 'x' to the corresponding 'y'.
## If an 'id' is not found in the TxDb 'y' will not
## be the same lenth as 'x' and an error is thrown.
## Not run:
distance(gr, txdb, id=c("FBgn0000008", "INVALID"), type="gene") ## will fail

## End(Not run)
```

---

proteinToGenome *Map protein-relative coordinates to genomic coordinates*

Description

proteinToGenome is a generic function for mapping ranges of protein-relative positions to the genome.

NOTE: This man page is for the proteinToGenome S4 generic function and methods defined in the **GenomicFeatures** package, which are (loosely) modeled on the proteinToGenome function from the **ensembldb** package. See ?ensembldb::proteinToGenome for the latter.

Usage

```
## S4 generic function:
proteinToGenome(x, txdb, ...)  # dispatch is on 2nd argument 'txdb'

## S4 method for signature 'ANY'
proteinToGenome(x, txdb)

## S4 method for signature 'GRangesList'
proteinToGenome(x, txdb)
```

Arguments

x	A named IRanges object (or derivative) containing ranges of *protein-relative positions* (protein-relative positions are positions relative to a protein sequence).
	The names on x must be transcript names present in txdb. More precisely, for the default proteinToGenome() method, names(x) must be a subset of:
	`mcols(transcripts(txdb, columns="tx_name"))$tx_name`
	And for the method for GRangesList objects, names(x) must be a subset of:
	`names(txdb)`

txdb	For the default proteinToGenome() method: A TxDb object or any object that supports transcripts() and cdsBy() (e.g. an EnsDb object from the **ensembldb** package). For the method for GRangesList objects: A named GRangesList object (or derivative) where each list element is a GRanges object representing a CDS (the ranges in the GRanges object must represent the CDS parts ordered by ascending exon rank).
...	Further arguments to be passed to specific methods.

Details

The proteinToGenome() method for GRangesList objects is the workhorse behind the default method. Note that the latter is a thin wrapper around the former, which simply does the following:

1. Use cdsBy() to extract the CDS from txdb. The CDS are returned in a GRangesList object with transcript names on it.
2. Call proteinToGenome() on x and the GRangesList object returned by cdsBy().

Value

A named GRangesList object *parallel* to x (the transcript names on x are propagated). The i-th list element in the returned object is the result of mapping the range of protein-relative positions x[i] to the genome.

Note that a given range in x can only be mapped to the genome if the name on it is the name of a *coding* transcript. If it's not (i.e. if it's the name of a *non-coding* transcript), then an empty GRanges object is placed in the returned object to indicate the impossible mapping, and a warning is issued.

Otherwise, if a given range in x can be mapped to the genome, then the result of the mapping is represented by a non-empty GRanges object. Note that this object represents the original CDS associated to x, trimmed on its 5' end or 3' end, or on both. Furthermore, this object will have the same metadata columns as the GRanges object representing the original CDS, plus the 2 following ones:

- protein_start: The protein-relative start of the mapping.
- protein_end: The protein-relative end of the mapping.

Note

Unlike ensembldb::proteinToGenome() which can work either with Ensembl protein IDs or Ensembl transcript IDs on x, the default proteinToGenome() method described above only accepts *transcript names* on x.

This means that, if the user is in possession of protein IDs, they must first replace them with the corresponding transcript IDs (referred to as *transcript names* in the context of TxDb objects). How to do this exactly depends on the origin of those IDs (UCSC, Ensembl, GTF/GFF3 file, FlyBase, etc...)

Author(s)

H. Pagès, and the authors of ensembldb::proteinToGenome() for the inspiration and design.

See Also

- The proteinToGenome function in the **ensembldb** package, which the proteinToGenome() generic and methods documented in this man page are (loosely) modeled on.
- TxDb objects.
- EnsDb objects (TxDb-like objects) in the **ensembldb** package.
- transcripts for extracting transcripts from a TxDb-like object.
- cdsBy for extracting CDS from a TxDb-like object.
- IRanges objects in the **IRanges** package.
- GRanges and GRangesList objects in the **GenomicRanges** package.

Examples

```
## ---------------------------------------------------------------------
## USING TOY CDS
## ---------------------------------------------------------------------
CDS1 <- GRanges(c("chrX:11-60:+", "chrX:101-125:+"))
CDS2 <- GRanges(c("chrY:201-230:-", "chrY:101-125:-", "chrY:11-60:-"))
cds_by_tx <- GRangesList(TX1=CDS1, TX2=CDS2)
cds_by_tx

x1 <- IRanges(start=8, end=20, names="TX1")
```

```
proteinToGenome(x1, cds_by_tx)

x2 <- IRanges(start=c(1, 18), end=c(25, 20), names=c("TX1", "TX1"))
x2
proteinToGenome(x2, cds_by_tx)

x3 <- IRanges(start=8, end=15, names="TX2")
proteinToGenome(x3, cds_by_tx)

x4 <- c(x3, x2)
x4
proteinToGenome(x4, cds_by_tx)

## ---------------------------------------------------------------------
## USING A TxDb OBJECT
## ---------------------------------------------------------------------
library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
txdb <- TxDb.Dmelanogaster.UCSC.dm3.ensGene

## The first transcript (FBtr0309810) is non-coding:
x <- IRanges(c(FBtr0309810="11-55", FBtr0306539="90-300"))
res <- proteinToGenome(x, txdb)
res
```

`select-methods` *Using the "select" interface on TxDb objects*

Description

`select`, `columns` and `keys` can be used together to extract data from a TxDb object.

Details

In the code snippets below, `x` is a TxDb object.

`keytypes(x)`: allows the user to discover which keytypes can be passed in to `select` or `keys` and the `keytype` argument.

`keys(x, keytype, pattern, column, fuzzy)`: Return keys for the database contained in the TxDb object .

The `keytype` argument specifies the kind of keys that will be returned. By default `keys` will return the "GENEID" keys for the database.

If `keys` is used with `pattern`, it will pattern match on the `keytype`.

But if the `column` argument is also provided along with the `pattern` argument, then `pattern` will be matched against the values in `column` instead.

And if `keys` is called with `column` and no `pattern` argument, then it will return all keys that have corresponding values in the `column` argument.

Thus, the behavior of `keys` all depends on how many arguments are specified.

Use of the `fuzzy` argument will toggle fuzzy matching to TRUE or FALSE. If `pattern` is not used, fuzzy is ignored.

`columns(x)`: Show which kinds of data can be returned for the TxDb object.

`select(x, keys, columns, keytype)`: When all the appropriate arguments are specified `select` will retrieve the matching data as a data.frame based on parameters for selected `keys` and `columns` and `keytype` arguments.

Author(s)

Marc Carlson

See Also

- AnnotationDb-class for more descriptsion of methods `select`,`keytypes`,`keys` and `columns`.
- `transcripts`, `transcriptsBy`, and `transcriptsByOverlaps`, for other ways to extract genomic features from a TxDb object.
- The TxDb class.

Examples

```
txdb_file <- system.file("extdata", "Biomart_Ensembl_sample.sqlite",
                         package="GenomicFeatures")
txdb <- loadDb(txdb_file)
txdb

## find key types
keytypes(txdb)

## list IDs that can be used to filter
head(keys(txdb, "GENEID"))
head(keys(txdb, "TXID"))
head(keys(txdb, "TXNAME"))

## list columns that can be returned by select
columns(txdb)

## call select
res <- select(txdb, head(keys(txdb, "GENEID")),
              columns=c("GENEID","TXNAME"),
              keytype="GENEID")
head(res)
```

---

transcriptLengths *Extract the transcript lengths (and other metrics) from a TxDb object*

Description

The `transcriptLengths` function extracts the transcript lengths from a TxDb object. It also returns the CDS and UTR lengths for each transcript if the user requests them.

Usage

```
transcriptLengths(txdb, with.cds_len=FALSE,
                  with.utr5_len=FALSE, with.utr3_len=FALSE, ...)
```

Arguments

txdb	A TxDb object.
with.cds_len, with.utr5_len, with.utr3_len	TRUE or FALSE. Whether or not to also extract and return the CDS, 5' UTR, and 3' UTR lengths for each transcript.
...	Additional arguments used by transcripts and other accessor functions.

Details

All the lengths are counted in number of nucleotides.

The length of a processed transcript is just the sum of the lengths of its exons. This should not be confounded with the length of the stretch of DNA transcribed into RNA (a.k.a. transcription unit), which can be obtained with `width(transcripts(txdb))`.

Value

A data frame with 1 row per transcript. The rows are guaranteed to be in the same order as the elements of the GRanges object returned by `transcripts(txdb)`. The data frame has between 5 and 8 columns, depending on what the user requested via the `with.cds_len`, `with.utr5_len`, and `with.utr3_len` arguments.

The first 3 columns are the same as the metadata columns of the object returned by

```
transcripts(txdb, columns=c("tx_id", "tx_name", "gene_id"))
```

that is:

- `tx_id`: The internal transcript ID. This ID is unique within the scope of the TxDb object. It is not an official or public ID (like an Ensembl or FlyBase ID) or an Accession number, so it cannot be used to lookup the transcript in public data bases or in other TxDb objects. Furthermore, this ID could change when re-running the code that was used to make the TxDb object.
- `tx_name`: An official/public transcript name or ID that can be used to lookup the transcript in public data bases or in other TxDb objects. This column is not guaranteed to contain unique values and it can contain NAs.
- `gene_id`: The official/public ID of the gene that the transcript belongs to. Can be NA if the gene is unknown or if the transcript is not considered to belong to a gene.

The other columns are quantitative:

- `nexon`: The number of exons in the transcript.
- `tx_len`: The length of the processed transcript.
- `cds_len`: [optional] The length of the CDS region of the processed transcript.
- `utr5_len`: [optional] The length of the 5' UTR region of the processed transcript.
- `utr3_len`: [optional] The length of the 3' UTR region of the processed transcript.

Author(s)

Hervé Pagès

See Also

- transcripts, transcriptsBy, and transcriptsByOverlaps, for extracting genomic feature locations from a TxDb-like object.
- exonicParts and intronicParts for extracting non-overlapping exonic or intronic parts from a TxDb-like object.
- extractTranscriptSeqs for extracting transcript (or CDS) sequences from chromosome sequences.
- coverageByTranscript for computing coverage by transcript (or CDS) of a set of ranges.
- makeTxDbFromUCSC, makeTxDbFromBiomart, and makeTxDbFromEnsembl, for making a TxDb object from online resources.
- makeTxDbFromGRanges and makeTxDbFromGFF for making a TxDb object from a GRanges object, or from a GFF or GTF file.
- The TxDb class.

Examples

```
library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
txdb <- TxDb.Dmelanogaster.UCSC.dm3.ensGene
dm3_txlens <- transcriptLengths(txdb)
head(dm3_txlens)

dm3_txlens <- transcriptLengths(txdb, with.cds_len=TRUE,
                                      with.utr5_len=TRUE,
                                      with.utr3_len=TRUE)
head(dm3_txlens)

## When cds_len is 0 (non-coding transcript), utr5_len and utr3_len
## must also be 0:
non_coding <- dm3_txlens[dm3_txlens$cds_len == 0, ]
stopifnot(all(non_coding[6:8] == 0))

## When cds_len is not 0 (coding transcript), cds_len + utr5_len +
## utr3_len must be equal to tx_len:
coding <- dm3_txlens[dm3_txlens$cds_len != 0, ]
stopifnot(all(rowSums(coding[6:8]) == coding[[5]]))

## A sanity check:
stopifnot(identical(dm3_txlens$tx_id, mcols(transcripts(txdb))$tx_id))
```

transcriptLocs2refLocs *Converting transcript-based locations into reference-based locations*

Description

transcriptLocs2refLocs converts transcript-based locations into reference-based (aka chromosome-based or genomic) locations.

transcriptWidths computes the lengths of the transcripts (called the "widths" in this context) based on the boundaries of their exons.

Usage

```
transcriptLocs2refLocs(tlocs,
        exonStarts=list(), exonEnds=list(), strand=character(0),
        decreasing.rank.on.minus.strand=FALSE, error.if.out.of.bounds=TRUE)

transcriptWidths(exonStarts=list(), exonEnds=list())
```

Arguments

tlocs	A list of integer vectors of the same length as exonStarts and exonEnds. Each element in tlocs must contain transcript-based locations.
exonStarts, exonEnds	The starts and ends of the exons, respectively. Each argument can be a list of integer vectors, an IntegerList object, or a character vector where each element is a comma-separated list of integers. In addition, the lists represented by exonStarts and exonEnds must have the same shape i.e. have the same lengths and have elements of the same lengths. The length of exonStarts and exonEnds is the number of transcripts.
strand	A character vector of the same length as exonStarts and exonEnds specifying the strand ("+" or "-") from which the transcript is coming.
decreasing.rank.on.minus.strand	TRUE or FALSE. Describes the order of exons in transcripts located on the minus strand: are they ordered by increasing (default) or decreasing rank?
error.if.out.of.bounds	TRUE or FALSE. Controls how out of bound tlocs are handled: an error is thrown (default) or NA is returned.

Value

For transcriptLocs2refLocs: A list of integer vectors of the same shape as tlocs.

For transcriptWidths: An integer vector with one element per transcript.

Author(s)

Hervé Pagès

See Also

- extractTranscriptSeqs for extracting transcript (or CDS) sequences from chromosomes.
- coverageByTranscript for computing coverage by transcript (or CDS) of a set of ranges.

Examples

```
## ---------------------------------------------------------------------
## WITH A SMALL SET OF HUMAN TRANSCRIPTS
## ---------------------------------------------------------------------
txdb_file <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                         package="GenomicFeatures")
txdb <- loadDb(txdb_file)
ex_by_tx <- exonsBy(txdb, by="tx", use.names=TRUE)
genome <- BSgenome::getBSgenome("hg19")  # load the hg19 genome
tx_seqs <- extractTranscriptSeqs(genome, ex_by_tx)

## Get the reference-based locations of the first 4 (5' end)
## and last 4 (3' end) nucleotides in each transcript:
tlocs <- lapply(width(tx_seqs), function(w) c(1:4, (w-3):w))
tx_strand <- sapply(strand(ex_by_tx), runValue)

## Note that, because of how we made them, 'tlocs', 'start(ex_by_tx)',
## 'end(ex_by_tx)' and 'tx_strand' are "parallel" objects i.e. they
## have the same length, and, for any valid positional index, elements
## at this position are corresponding to each other. This is how
## transcriptLocs2refLocs() expects them to be!
rlocs <- transcriptLocs2refLocs(tlocs,
             start(ex_by_tx), end(ex_by_tx),
             tx_strand, decreasing.rank.on.minus.strand=TRUE)

## ---------------------------------------------------------------------
## WITH TWO WORM TRANSCRIPTS: ZC101.3.1 AND F37B1.1.1
## ---------------------------------------------------------------------
library(TxDb.Celegans.UCSC.ce11.ensGene)
txdb <- TxDb.Celegans.UCSC.ce11.ensGene
my_tx_names <- c("ZC101.3.1", "F37B1.1.1")
## Both transcripts are on chromosome II, the first one on its positive
## strand and the second one on its negative strand:
my_tx <- transcripts(txdb, filter=list(tx_name=my_tx_names))
my_tx

## Using transcripts stored in a GRangesList object:
ex_by_tx <- exonsBy(txdb, use.names=TRUE)[my_tx_names]
genome <- getBSgenome("ce11")  # load the ce11 genome
tx_seqs <- extractTranscriptSeqs(genome, ex_by_tx)
tx_seqs

## Since the 2 transcripts are on the same chromosome, an alternative
```

```
## is to store them in an IRangesList object and use that object with
## extractTranscriptSeqs():
ex_by_tx2 <- ranges(ex_by_tx)
tx_seqs2 <- extractTranscriptSeqs(genome$chrII, ex_by_tx2,
                                  strand=strand(my_tx))
stopifnot(identical(as.character(tx_seqs), as.character(tx_seqs2)))

## Store exon starts and ends in two IntegerList objects for use with
## transcriptWidths() and transcriptLocs2refLocs():
exon_starts <- start(ex_by_tx)
exon_ends <- end(ex_by_tx)

## Same as 'width(tx_seqs)':
transcriptWidths(exonStarts=exon_starts, exonEnds=exon_ends)

transcriptLocs2refLocs(list(c(1:2, 202:205, 1687:1688),
                            c(1:2, 193:196, 721:722)),
                       exonStarts=exon_starts,
                       exonEnds=exon_ends,
                       strand=c("+","-"))

## A sanity check:
ref_locs <- transcriptLocs2refLocs(list(1:1688, 1:722),
                                   exonStarts=exon_starts,
                                   exonEnds=exon_ends,
                                   strand=c("+","-"))
stopifnot(genome$chrII[ref_locs[[1]]] == tx_seqs[[1]])
stopifnot(complement(genome$chrII)[ref_locs[[2]]] == tx_seqs[[2]])
```

`transcripts` *Extract genomic features from a TxDb-like object*

Description

Generic functions to extract genomic features from a TxDb-like object. This page documents the methods for TxDb objects only.

Usage

```
transcripts(x, ...)
## S4 method for signature 'TxDb'
transcripts(x, columns=c("tx_id", "tx_name"), filter=NULL, use.names=FALSE)

exons(x, ...)
## S4 method for signature 'TxDb'
exons(x, columns="exon_id", filter=NULL, use.names=FALSE)

cds(x, ...)
## S4 method for signature 'TxDb'
```

```
cds(x, columns="cds_id", filter=NULL, use.names=FALSE)

genes(x, ...)
## S4 method for signature 'TxDb'
genes(x, columns="gene_id", filter=NULL, single.strand.genes.only=TRUE)

## S4 method for signature 'TxDb'
promoters(x, upstream=2000, downstream=200, use.names=TRUE, ...)
```

Arguments

x	A TxDb object.
...	For the transcripts, exons, cds, and genes generic functions: arguments to be passed to methods. For the promoters method for TxDb objects: arguments to be passed to the internal call to transcripts.
columns	Columns to include in the output. Must be NULL or a character vector as given by the columns method. With the following restrictions: • "TXCHROM" and "TXSTRAND" are not allowed for transcripts. • "EXONCHROM" and "EXONSTRAND" are not allowed for exons. • "CDSCHROM" and "CDSSTRAND" are not allowed for cds. If the vector is named, those names are used for the corresponding column in the element metadata of the returned object.
filter	Either NULL or a named list of vectors to be used to restrict the output. Valid names for this list are: "gene_id", "tx_id", "tx_name", "tx_chrom", "tx_strand", "exon_id", "exon_name", "exon_chrom", "exon_strand", "cds_id", "cds_name", "cds_chrom", "cds_strand" and "exon_rank".
use.names	TRUE or FALSE. If TRUE, the feature names are set as the names of the returned object, with NAs being replaced with empty strings.
single.strand.genes.only	TRUE or FALSE. If TRUE (the default), then genes are returned in a GRanges object and those genes that cannot be represented by a single genomic range (because they have exons located on both strands of the same reference sequence or on more than one reference sequence) are dropped with a message. If FALSE, then all the genes are returned in a GRangesList object with the columns specified thru the columns argument set as *top level* metadata columns. (Please keep in mind that the *top level* metadata columns of a GRangesList object are not displayed by the show() method.)
upstream	For promoters : An integer(1) value indicating the number of bases upstream from the transcription start site. For additional details see ?`promoters,GRanges-method`.
downstream	For promoters : An integer(1) value indicating the number of bases downstream from the transcription start site. For additional details see ?`promoters,GRanges-method`.

Details

These are the main functions for extracting transcript information from a TxDb-like object. These methods can restrict the output based on categorical information. To restrict the output based on interval information, use the `transcriptsByOverlaps`, `exonsByOverlaps`, and `cdsByOverlaps` functions.

The `promoters` function computes user-defined promoter regions for the transcripts in a TxDb-like object. The return object is a `GRanges` of promoter regions around the transcription start site the span of which is defined by `upstream` and `downstream`. For additional details on how the promoter range is computed and the handling of `+` and `-` strands see `` ?`promoters,GRanges-method` ``.

Value

A GRanges object. The only exception being when `genes` is used with `single.strand.genes.only=FALSE`, in which case a GRangesList object is returned.

Author(s)

M. Carlson, P. Aboyoun and H. Pagès

See Also

- `transcriptsBy` and `transcriptsByOverlaps` for more ways to extract genomic features from a TxDb-like object.
- `transcriptLengths` for extracting the transcript lengths (and other metrics) from a TxDb object.
- `exonicParts` and `intronicParts` for extracting non-overlapping exonic or intronic parts from a TxDb-like object.
- `extendExonsIntoIntrons` for extending exons into their adjacent introns.
- `extractTranscriptSeqs` for extracting transcript (or CDS) sequences from reference sequences.
- `coverageByTranscript` for computing coverage by transcript (or CDS) of a set of ranges.
- select-methods for how to use the simple "select" interface to extract information from a TxDb object.
- `microRNAs` and `tRNAs` for extracting microRNA or tRNA genomic ranges from a TxDb object.
- `id2name` for mapping TxDb internal ids to external names for a given feature type.
- The TxDb class.

Examples

```
txdb_file <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                         package="GenomicFeatures")
txdb <- loadDb(txdb_file)

## ---------------------------------------------------------------------
## transcripts()
## ---------------------------------------------------------------------
```

```
tx1 <- transcripts(txdb)
tx1

transcripts(txdb, use.names=TRUE)
transcripts(txdb, columns=NULL, use.names=TRUE)

filter <- list(tx_chrom = c("chr3", "chr5"), tx_strand = "+")
tx2 <- transcripts(txdb, filter=filter)
tx2

## Sanity checks:
stopifnot(
  identical(mcols(tx1)$tx_id, seq_along(tx1)),
  identical(tx2, tx1[seqnames(tx1) == "chr3" & strand(tx1) == "+"])
)

## ---------------------------------------------------------------------
## exons()
## ---------------------------------------------------------------------

exons(txdb, columns=c("EXONID", "TXNAME"),
            filter=list(exon_id=1))
exons(txdb, columns=c("EXONID", "TXNAME"),
            filter=list(tx_name="uc009vip.1"))

## ---------------------------------------------------------------------
## genes()
## ---------------------------------------------------------------------

genes(txdb)  # a GRanges object
cols <- c("tx_id", "tx_chrom", "tx_strand",
          "exon_id", "exon_chrom", "exon_strand")
## By default, genes are returned in a GRanges object and those that
## cannot be represented by a single genomic range (because they have
## exons located on both strands of the same reference sequence or on
## more than one reference sequence) are dropped with a message:
single_strand_genes <- genes(txdb, columns=cols)

## Because we've returned single strand genes only, the "tx_chrom"
## and "exon_chrom" metadata columns are guaranteed to match
## 'seqnames(single_strand_genes)':
stopifnot(identical(as.character(seqnames(single_strand_genes)),
                    as.character(mcols(single_strand_genes)$tx_chrom)))
stopifnot(identical(as.character(seqnames(single_strand_genes)),
                    as.character(mcols(single_strand_genes)$exon_chrom)))

## and also the "tx_strand" and "exon_strand" metadata columns are
## guaranteed to match 'strand(single_strand_genes)':
stopifnot(identical(as.character(strand(single_strand_genes)),
                    as.character(mcols(single_strand_genes)$tx_strand)))
stopifnot(identical(as.character(strand(single_strand_genes)),
                    as.character(mcols(single_strand_genes)$exon_strand)))
```

```
all_genes <- genes(txdb, columns=cols, single.strand.genes.only=FALSE)
all_genes  # a GRangesList object
multiple_strand_genes <- all_genes[elementNROWS(all_genes) >= 2]
multiple_strand_genes
mcols(multiple_strand_genes)

## ---------------------------------------------------------------------
## promoters()
## ---------------------------------------------------------------------

## This:
promoters(txdb, upstream=100, downstream=50)
## is equivalent to:
promoters(transcripts(txdb, use.names=TRUE), upstream=100, downstream=50)

## Extra arguments are passed to transcripts(). So this:
columns <- c("tx_name", "gene_id")
promoters(txdb, upstream=100, downstream=50, columns=columns)
## is equivalent to:
promoters(transcripts(txdb, columns=columns, use.names=TRUE),
          upstream=100, downstream=50)
```

`transcriptsBy` *Extract and group genomic features of a given type from a TxDb-like object*

Description

Generic functions to extract genomic features of a given type grouped based on another type of genomic feature. This page documents the methods for TxDb objects only.

Usage

```
transcriptsBy(x, by=c("gene", "exon", "cds"), ...)
## S4 method for signature 'TxDb'
transcriptsBy(x, by=c("gene", "exon", "cds"), use.names=FALSE)

exonsBy(x, by=c("tx", "gene"), ...)
## S4 method for signature 'TxDb'
exonsBy(x, by=c("tx", "gene"), use.names=FALSE)

cdsBy(x, by=c("tx", "gene"), ...)
## S4 method for signature 'TxDb'
cdsBy(x, by=c("tx", "gene"), use.names=FALSE)

intronsByTranscript(x, ...)
## S4 method for signature 'TxDb'
intronsByTranscript(x, use.names=FALSE)
```

```
fiveUTRsByTranscript(x, ...)
## S4 method for signature 'TxDb'
fiveUTRsByTranscript(x, use.names=FALSE)

threeUTRsByTranscript(x, ...)
## S4 method for signature 'TxDb'
threeUTRsByTranscript(x, use.names=FALSE)
```

Arguments

x	A TxDb object.
...	Arguments to be passed to or from methods.
by	One of "gene", "exon", "cds" or "tx". Determines the grouping.
use.names	Controls how to set the names of the returned GRangesList object. These functions return all the features of a given type (e.g. all the exons) grouped by another feature type (e.g. grouped by transcript) in a GRangesList object. By default (i.e. if use.names is FALSE), the names of this GRangesList object (aka the group names) are the internal ids of the features used for grouping (aka the grouping features), which are guaranteed to be unique. If use.names is TRUE, then the names of the grouping features are used instead of their internal ids. For example, when grouping by transcript (by="tx"), the default group names are the transcript internal ids ("tx_id"). But, if use.names=TRUE, the group names are the transcript names ("tx_name"). Note that, unlike the feature ids, the feature names are not guaranteed to be unique or even defined (they could be all NAs). A warning is issued when this happens. See ?id2name for more information about feature internal ids and feature external names and how to map the formers to the latters. Finally, use.names=TRUE cannot be used when grouping by gene by="gene". This is because, unlike for the other features, the gene ids are external ids (e.g. Entrez Gene or Ensembl ids) so the db doesn't have a "gene_name" column for storing alternate gene names.

Details

These functions return a GRangesList object where the ranges within each of the elements are ordered according to the following rule:

When using exonsBy or cdsBy with by = "tx", the returned exons or CDS are ordered by ascending rank for each transcript, that is, by their position in the transcript. In all other cases, the ranges will be ordered by chromosome, strand, start, and end values.

Value

A GRangesList object.

Author(s)

M. Carlson, P. Aboyoun and H. Pagès

See Also

- transcripts and transcriptsByOverlaps for more ways to extract genomic features from a TxDb-like object.
- transcriptLengths for extracting the transcript lengths (and other metrics) from a TxDb object.
- exonicParts and intronicParts for extracting non-overlapping exonic or intronic parts from a TxDb-like object.
- extendExonsIntoIntrons for extending exons into their adjacent introns.
- extractTranscriptSeqs for extracting transcript (or CDS) sequences from chromosome sequences.
- coverageByTranscript for computing coverage by transcript (or CDS) of a set of ranges.
- select-methods for how to use the simple "select" interface to extract information from a TxDb object.
- id2name for mapping TxDb internal ids to external names for a given feature type.
- The TxDb class.

Examples

```
txdb_file <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                         package="GenomicFeatures")
txdb <- loadDb(txdb_file)

## Get the transcripts grouped by gene:
transcriptsBy(txdb, "gene")

## Get the exons grouped by gene:
exonsBy(txdb, "gene")

## Get the CDS grouped by transcript:
cds_by_tx0 <- cdsBy(txdb, "tx")
## With more informative group names:
cds_by_tx1 <- cdsBy(txdb, "tx", use.names=TRUE)
## Note that 'cds_by_tx1' can also be obtained with:
names(cds_by_tx0) <- id2name(txdb, feature.type="tx")[names(cds_by_tx0)]
stopifnot(identical(cds_by_tx0, cds_by_tx1))

## Get the introns grouped by transcript:
intronsByTranscript(txdb)

## Get the 5' UTRs grouped by transcript:
fiveUTRsByTranscript(txdb)
fiveUTRsByTranscript(txdb, use.names=TRUE)  # more informative group names
```

transcriptsByOverlaps *Extract genomic features from a TxDb-like object based on their genomic location*

Description

Generic functions to extract genomic features for specified genomic locations. This page documents the methods for TxDb objects only.

Usage

```
transcriptsByOverlaps(x, ranges,
                      maxgap = -1L, minoverlap = 0L,
                      type = c("any", "start", "end"), ...)
## S4 method for signature 'TxDb'
transcriptsByOverlaps(x, ranges,
                      maxgap = -1L, minoverlap = 0L,
                      type = c("any", "start", "end"),
                      columns = c("tx_id", "tx_name"))

exonsByOverlaps(x, ranges,
                maxgap = -1L, minoverlap = 0L,
                type = c("any", "start", "end"), ...)
## S4 method for signature 'TxDb'
exonsByOverlaps(x, ranges,
                maxgap = -1L, minoverlap = 0L,
                type = c("any", "start", "end"),
                columns = "exon_id")

cdsByOverlaps(x, ranges,
              maxgap = -1L, minoverlap = 0L,
              type = c("any", "start", "end"), ...)
## S4 method for signature 'TxDb'
cdsByOverlaps(x, ranges,
              maxgap = -1L, minoverlap = 0L,
              type = c("any", "start", "end"),
              columns = "cds_id")
```

Arguments

x	A TxDb object.
ranges	A GRanges object to restrict the output.
maxgap,minoverlap,type	Used in the internal call to findOverlaps() to detect overlaps. See ?findOverlaps in the **IRanges** package for a description of these arguments.
...	Arguments to be passed to or from methods.
columns	Columns to include in the output. See ?transcripts for the possible values.

Details

These functions subset the results of `transcripts`, `exons`, and `cds` function calls with using the results of `findOverlaps` calls based on the specified `ranges`.

Value

a GRanges object

Author(s)

P. Aboyoun

See Also

- `transcripts` and `transcriptsBy` for more ways to extract genomic features from a TxDb-like object.
- `transcriptLengths` for extracting the transcript lengths (and other metrics) from a TxDb object.
- `exonicParts` and `intronicParts` for extracting non-overlapping exonic or intronic parts from a TxDb-like object.
- `extractTranscriptSeqs` for extracting transcript (or CDS) sequences from chromosome sequences.
- `coverageByTranscript` for computing coverage by transcript (or CDS) of a set of ranges.
- select-methods for how to use the simple "select" interface to extract information from a TxDb object.
- `id2name` for mapping TxDb internal ids to external names for a given feature type.
- The TxDb class.

Examples

```
txdb <- loadDb(system.file("extdata", "hg19_knownGene_sample.sqlite",
                           package="GenomicFeatures"))
gr <- GRanges(Rle("chr1", 2),
              IRanges(c(500,10500), c(10000,30000)),
              strand = Rle("-", 2))
transcriptsByOverlaps(txdb, gr)
```

`TxDb-class` *TxDb objects*

Description

The TxDb class is a container for storing transcript annotations.

Methods

In the code snippets below, x is a TxDb object.

`metadata(x)`: Return x's metadata in a data frame.

`seqlevels0(x)`: Get the *sequence levels* originally in x. This ignores any change the user might have made to the *sequence levels* with the `seqlevels` setter.

`seqlevels(x)`, `seqlevels(x) <- value`: Get or set the *sequence levels* in x.

`seqinfo(x)`, `seqinfo(x) <- value`: Get or set the information about the underlying sequences. Note that, for now, the setter only supports replacement of the sequence names, i.e., except for their sequence names (accessed with `seqnames(value)` and `seqnames(seqinfo(x))`, respectively), Seqinfo objects `value` (supplied) and `seqinfo(x)` (current) must be identical.

`isActiveSeq(x)`: Return the currently active sequences for this txdb object as a named logical vector. Only active sequences will be tapped when using the supplied accessor methods. Inactive sequences will be ignored. By default, all available sequences will be active.

`isActiveSeq(x) <- value`: Allows the user to change which sequences will be actively accessed by the accessor methods by altering the contents of this named logical vector.

`seqlevelsStyle(x)`, `seqlevelsStyle(x) <- value`: Get or set the seqname style for x. See the seqlevelsStyle generic getter and setter in the **GenomeInfoDb** package for more information.

`as.list(x)`: Dump the entire db into a list of data frames, say `txdb_dump`, that can then be used to recreate the original db with `do.call(makeTxDb, txdb_dump)` with no loss of information (except possibly for some of the metadata). Note that the transcripts are dumped in the same order in all the data frames.

Author(s)

Hervé Pagès, Marc Carlson

See Also

- `makeTxDbFromUCSC`, `makeTxDbFromBiomart`, and `makeTxDbFromEnsembl`, for making a TxDb object from online resources.
- `makeTxDbFromGRanges` and `makeTxDbFromGFF` for making a TxDb object from a GRanges object, or from a GFF or GTF file.
- `saveDb` and `loadDb` in the **AnnotationDbi** package for saving and loading a TxDb object as an SQLite file.
- `transcripts`, `transcriptsBy`, and `transcriptsByOverlaps`, for extracting genomic feature locations from a TxDb-like object.
- `transcriptLengths` for extracting the transcript lengths (and other metrics) from a TxDb object.
- select-methods for how to use the simple "select" interface to extract information from a TxDb object.
- The Seqinfo class in the **GenomeInfoDb** package.

Examples

```
txdb_file <- system.file("extdata", "Biomart_Ensembl_sample.sqlite",
                         package="GenomicFeatures")
txdb <- loadDb(txdb_file)
txdb

## Use of seqinfo():
seqlevelsStyle(txdb)
seqinfo(txdb)
seqlevels(txdb)
seqlengths(txdb)  # shortcut for 'seqlengths(seqinfo(txdb))'
isCircular(txdb)  # shortcut for 'isCircular(seqinfo(txdb))'
names(which(isCircular(txdb)))

## You can set user-supplied seqlevels on 'txdb' to restrict any further
## operations to a subset of chromosomes:
seqlevels(txdb) <- c("Y", "6")
## Then you can restore the seqlevels stored in the db:
seqlevels(txdb) <- seqlevels0(txdb)

## Use of as.list():
txdb_dump <- as.list(txdb)
txdb_dump
txdb1 <- do.call(makeTxDb, txdb_dump)
stopifnot(identical(as.list(txdb1), txdb_dump))
```

Index