Package 'GenomicFeatures'

April 10, 2015

Title Tools for making and manipulating transcript centric annotations

Version 1.18.7

Author M. Carlson, H. Pages, P. Aboyoun, S. Falcon, M. Morgan, D. Sarkar, M. Lawrence

License Artistic-2.0

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.

Maintainer Bioconductor Package Maintainer

<maintainer@bioconductor.org>

Depends BiocGenerics (>= 0.1.0), S4Vectors (>= 0.1.5), IRanges (>= 1.99.1), GenomeInfoDb (>= 1.1.3), GenomicRanges (>= 1.17.12), AnnotationDbi (>= 1.27.9)

Imports methods, DBI (>= 0.2-5), RSQLite (>= 0.8-1), Biostrings (>= 2.23.3), rtracklayer (>= 1.26.3), biomaRt (>= 2.17.1), RCurl, utils, Biobase (>= 2.15.1)

Suggests org.Mm.eg.db, BSgenome, BSgenome.Hsapiens.UCSC.hg19 (>= 1.3.17), BSgenome.Celegans.UCSC.ce2, BSgenome.Dmelanogaster.UCSC.dm3 (>= 1.3.17), mirbase.db, FDb.UCSC.tRNAs, TxDb.Hsapiens.UCSC.hg19.knownGene, TxDb.Dmelanogaster.UCSC.dm3.ensGene (>= 2.7.1), Rsamtools, pasillaBamSubset (>= 0.0.5), RUnit, BiocStyle, knitr

Collate utils.R Ensembl.utils.R findCompatibleMarts.R TxDb-class.R FeatureDb-class.R makeTranscriptDb.R makeTranscriptDbFromUCSC.R makeTranscriptDbFromBiomart.R makeTranscriptDbFromGFF.R makeFeatureDbFromUCSC.R id2name.R transcripts.R

transcriptsByOverlaps.R transcriptsBy.R features.R
extractUpstreamSeqs.R extractTranscriptSeqs.R
transcriptLocs2refLocs.R makeTxDbPackage.R seqnames-methods.R
select-methods.R nearest-methods.R getPromoterSeq-methods.R
extractTranscriptsFromGenome.R test_GenomicFeatures_package.R

VignetteBuilder knitr

biocViews Genetics, Infrastructure, Annotation, Sequencing, GenomeAnnotation

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)
```

DEFAULT_CIRC_SEQS *character vector: strings that are usually circular chromosomes*

Description

The DEFAULT_CIRC_SEQS character vector contains strings that are normally used by major repositories as the names of chromosomes that are typically circular, it is available as a convenience so that users can us it as a default value for circ_seqs arguments, and append to it as needed.

Usage

```
DEFAULT_CIRC_SEQS
```

See Also

makeTranscriptDbFromUCSC, makeTranscriptDbFromBiomart

Examples

```
DEFAULT_CIRC_SEQS
```

---

extractTranscriptSeqs *Extract transcript sequences from chromosomes*

Description

extractTranscriptSeqs is a generic function for extracting transcript sequences from an object representing a single chromosome (e.g. a DNAString object) or a collection of chromosomes (e.g. a BSgenome object).

Usage

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

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

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

Arguments

x	A DNAString or BSgenome object or any object with a getSeq method.
transcripts	An object representing the exon ranges of each transcript to extract. It must be an RangesList object when x is a DNAString object. It must be a GRangesList or TxDb object when x is a BSgenome object. If the latter, it's first turned into a GRangesList object with exonsBy(transcripts, by="tx", use.names=TRU
...	Additional arguments, for use in specific methods.
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 RangesList 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 RangesList 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 the returned object is the sequence of the i-th transcript in `transcripts`.

Author(s)

H. Pages

See Also

- 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 GRangesList class defined and documented in the **GenomicRanges** package.
- The RangesList class defined and documented in the **IRanges** package.
- The `exonsBy` function for extracting exon ranges grouped by transcript.
- 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.

Examples

```
## ---------------------------------------------------------------------
## A FIRST 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)))))

## ---------------------------------------------------------------------
## 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:
five_utr_width <- sum(width(fiveUTRsByTranscript(txdb, use.names=TRUE)))
five_utr_width <- five_utr_width[names(cds_seqs)]
five_utr_width[is.na(five_utr_width)] <- 0L
three_utr_width <- sum(width(threeUTRsByTranscript(txdb, use.names=TRUE)))
three_utr_width <- three_utr_width[names(cds_seqs)]
three_utr_width[is.na(three_utr_width)] <- 0L
cds_seqs2 <- narrow(tx_seqs[names(cds_seqs)],
                    start=five_utr_width+1L,
                    end=-(three_utr_width+1L))
stopifnot(identical(as.character(cds_seqs), as.character(cds_seqs2)))

## ---------------------------------------------------------------------
## 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")
```

---

extractTranscriptsFromGenome

Various DEFUNCT tools for extracting transcript sequences

Description

WARNING: Starting with BioC 3.0, the tools described in this man page are DEFUNCT.

`extractTranscriptsFromGenome` extracts the transcript sequences from a BSgenome data package using the transcript information (exon boundaries) stored in a TxDb or GRangesList object. DEFUNCT. Please use `extractTranscriptSeqs` instead.

`extractTranscripts` extracts a set of transcripts from a single DNA sequence. DEFUNCT. Please use `extractTranscriptSeqs` instead.

Related utility:

`sortExonsByRank` orders (or reorders) by rank the exons stored in a GRangesList object containing exons grouped by transcript. DEPRECATED. No replacement.

Usage

```
# DEFUNCT. Please use extractTranscriptSeqs() instead.
extractTranscriptsFromGenome(genome, txdb,
                             decreasing.rank.on.minus.strand=FALSE,
                             use.names=TRUE)

# DEFUNCT. Please use extractTranscriptSeqs() instead.
extractTranscripts(x,
        exonStarts=list(), exonEnds=list(), strand=character(0),
        decreasing.rank.on.minus.strand=FALSE)

# DEPRECATED. No replacement.
sortExonsByRank(x, decreasing.rank.on.minus.strand=FALSE)
```

Arguments

`genome`	A BSgenome object. See the `available.genomes` function in the BSgenome package for how to install a genome.
`txdb`	A TxDb object or a GRangesList object.
`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? For all the functions described in this man page (except `sortExonsByRank`), this argument describes the input. For `sortExonsByRank`, it describes how exons should be ordered in the output.
`use.names`	`TRUE` or `FALSE`. Ignored if `txdb` is not a TxDb object. If `TRUE` (the default), the returned sequences are named with the transcript names. If `FALSE`, they are named with the transcript internal ids. Note that, unlike the transcript internal ids, the transcript names are not guaranteed to be unique or even defined (they could be all `NA`s). A warning is issued when this happens.
`x`	A DNAString or MaskedDNAString object for `extractTranscripts`. A GRangesList object for `sortExonsByRank`, typically coming from `exonsBy(... , by="tx")`.

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.

Value

For extractTranscriptsFromGenome: A named DNAStringSet object with one element per transcript. When txdb is a GRangesList object, elements in the output align with elements in the input (txdb), and they have the same names.

For extractTranscripts: A DNAStringSet object with one element per transcript.

For sortExonsByRank: A GRangesList object with one top-level element per transcript. More precisely, the returned object has the same "shape" (i.e. same length and same number of elements per top-level element) as the input GRangesList object x.

Author(s)

H. Pages

See Also

extractTranscriptSeqs as a replacement for extractTranscriptsFromGenome and extractTranscripts.

---

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. See the makeTranscriptDbFromUCSC, makeTranscriptDbFromBiomart, and makeTranscriptDbFromGFF functions for more information about this.

Author(s)

H. Pages

See Also

- The available.genomes function in the **BSgenome** package for checking avaibility of BSgenome data packages (and installing the desired one).
- The makeTranscriptDbFromUCSC, makeTranscriptDbFromBiomart, and makeTranscriptDbFromGFF functions for making your own custom TxDb object from various annotation resources.
- 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

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

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)
```

`GenomicFeatures-deprecated`
Deprecated Functions in package **GenomicFeatures**

Description

The functions or variables listed here have been deprecated and should no longer be used.

Details

The following functions are deprecated and will be made defunct; use the replacement indicated below:

- determineDefaultSeqnameStyle: `seqlevelsStyle`

See Also

`Deprecated`

`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 methods for Ranges objects intra-range-methods ## promoters methods for GRanges 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)

H. Pages

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

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="http://hgdownload.cse.ucsc.edu/goldenPath",
        chromCol,
        chromStartCol,
        chromEndCol)
```

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'

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 and H. Pages

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("mm9")

## Display the list of tables supported by your track:
supportedUCSCFeatureDbTables(genome="mm9",
                             track="oreganno")

## Display fields that could be passed in to colnames:
UCSCFeatureDbTableSchema(genome="mm9",
                         track="oreganno",
```

```
                        tablename="oreganno")

## Retrieving a full transcript dataset for Yeast from UCSC:
fdb <- makeFeatureDbFromUCSC(genome="mm9",
                             track="oreganno",
                             tablename="oreganno")
fdb
```

---

makeTranscriptDb *Making a TxDb object from user supplied annotations*

Description

makeTranscriptDb is a low-level constructor for making a TxDb object from user supplied transcript annotations. See ?makeTranscriptDbFromUCSC and ?makeTranscriptDbFromBiomart for higher-level functions that feed data from the UCSC or BioMart sources to makeTranscriptDb.

Usage

```
makeTranscriptDb(transcripts, splicings,
                 genes=NULL, chrominfo=NULL, metadata=NULL,
                 reassign.ids=FALSE)
```

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 species, organism, genome, UCSC table, etc... The names of the columns must be "name" and "value" and their type must be character.
reassign.ids	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 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.

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_name: [optional] Transcript name. Character vector (or factor). NAs and/or duplicates are ok.
- 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.

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).
- 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 NAs 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 are allowed and must match NAs in cds_start and cds_end.
- 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.

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

genes 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. Either all NAs or an integer vector with no NAs.
- `is_circular`: [optional] Chromosome circularity flag. Either all NAs or a logical vector with no NAs.

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

Value

A TxDb object.

Author(s)

H. Pages

See Also

- `makeTranscriptDbFromUCSC` and `makeTranscriptDbFromBiomart` for convenient ways to make TxDb objects from UCSC or BioMart online resources.
- `makeTranscriptDbFromGFF` for making a TxDb object from annotations available as a GFF3 or GTF file.
- The TxDb class.

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))

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

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

Description

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

Usage

```
makeTranscriptDbFromBiomart(biomart="ensembl",
                            dataset="hsapiens_gene_ensembl",
                            transcript_ids=NULL,
                            circ_seqs=DEFAULT_CIRC_SEQS,
                            filters="",
                            id_prefix="ensembl_",
                            host="www.biomart.org",
                            port=80,
                            miRBaseBuild=NA)

getChromInfoFromBiomart(biomart="ensembl",
                        dataset="hsapiens_gene_ensembl",
                        id_prefix="ensembl_",
                        host="www.biomart.org",
                        port=80)
```

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.
filters	Additional filters to use in the BioMart query. Must be a named list. An example is filters=as.list(c(source="entrez"))
host	The host URL of the BioMart. Defaults to www.biomart.org.
port	The port to use in the HTTP communication with the host.

`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_"`.

`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

`makeTranscriptDbFromBiomart` is a convenience function that feeds data from a BioMart database to the lower level `makeTranscriptDb` function. See `?makeTranscriptDbFromUCSC` for a similar function that feeds data from the UCSC source.

The `listMarts` function from the **biomaRt** package can be used to list all public BioMart databases. Not all databases returned by this function contain datasets that are compatible with (i.e. understood by) `makeTranscriptDbFromBiomart`. Here is a list of datasets known to be compatible (updated on Sep 24, 2014):

- All the datasets in the main Ensembl database: use `biomart="ensembl"`.
- All the datasets in the Ensembl Fungi database: use `biomart="fungi_mart_XX"` where XX is the release version of the database e.g. `"fungi_mart_22"`.
- All the datasets in the Ensembl Metazoa database: use `biomart="metazoa_mart_XX"` where XX is the release version of the database e.g. `"metazoa_mart_22"`.
- All the datasets in the Ensembl Plants database: use `biomart="plants_mart_XX"` where XX is the release version of the database e.g. `"plants_mart_22"`.
- All the datasets in the Ensembl Protists database: use `biomart="protists_mart_XX"` where XX is the release version of the database e.g. `"protists_mart_22"`.
- All the datasets in the Gramene Mart: use `biomart="ENSEMBL_MART_PLANT"`.

Not all these datasets have CDS information.

Value

A TxDb object.

Author(s)

M. Carlson and H. Pages

See Also

`listMarts`, `useMart`, `listDatasets`, `DEFAULT_CIRC_SEQS`, `makeTranscriptDbFromUCSC`, `makeTranscriptDbFromGFF`, `makeTranscriptDb`, `supportedMiRBaseBuildValues`

Examples

```
## Discover which datasets are available in the "ensembl" BioMart
## database:
library("biomaRt")
head(listDatasets(useMart("ensembl")))

## Retrieving an incomplete transcript dataset for Human from the
## "ensembl" BioMart database:
transcript_ids <- c(
    "ENST00000013894",
    "ENST00000268655",
    "ENST00000313243",
    "ENST00000435657",
    "ENST00000384428",
    "ENST00000478783"
)
txdb <- makeTranscriptDbFromBiomart(transcript_ids=transcript_ids)
txdb  # note that these annotations match the GRCh37 genome assembly

## Now what if we want to use another mirror?  We might make use of the
## new host argument.  But wait!  If we use biomaRt, we can see that
## this host has named the mart differently!
listMarts(host="uswest.ensembl.org")
## Therefore we must also change the name passed into the "mart"
## argument thusly:
try(
    txdb <- makeTranscriptDbFromBiomart(biomart="ENSEMBL_MART_ENSEMBL",
                                        transcript_ids=transcript_ids,
                                        host="uswest.ensembl.org")
)
txdb
```

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

Description

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

Usage

```
makeTranscriptDbFromGFF(file,
                        format=c("gff3","gtf"),
                        exonRankAttributeName=NA,
                        gffGeneIdAttributeName=NA,
                        chrominfo=NA,
```

```
                    dataSource=NA,
                    species=NA,
                    circ_seqs=DEFAULT_CIRC_SEQS,
                    miRBaseBuild=NA,
                    useGenesAsTranscripts=FALSE)
```

Arguments

file	path/file to be processed
format	"gff3" or "gtf" depending on which file format you have to process
exonRankAttributeName	character(1) name of the attribute that defines the exon rank information, or NA to indicate that exon ranks are inferred from order of occurrence in the GFF.
gffGeneIdAttributeName	an optional argument that can be used for gff style files ONLY. If the gff file lacks rows to specify gene IDs but the mRNA rows of the gff file specify the gene IDs via a named attribute,then passing the name of the attribute for this argument can allow the file to still extract gene IDs that map to these transcripts. If left blank, then the parser will try and extract rows that are named 'gene' for gene to transcript mappings when parsing a gff3 file. For gtf files this argument is ignored entirely.
chrominfo	data frame containing information about the chromosomes. Will be passed to the internal call to makeTranscriptDb. See ?makeTranscriptDb for the details.
dataSource	Where did this data file originate? Please be as specific as possible.
species	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.
circ_seqs	a character vector to list out which chromosomes should be marked as circular.
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.
useGenesAsTranscripts	This flag is normally off, but if enabled it will try to salvage a file that has no RNA features by assuming that you can use the ranges available for the Gene features in their place. Obviously, this is something you won't want to do unless you are dealing with something very simple like a prokaryote.

Details

makeTranscriptDbFromGFF is a convenience function that feeds data from the parsed file to the lower level makeTranscriptDb function.

There are some real deficiencies in the gtf and the gff3 file formats to bear in mind when making use of them. For gtf files the length of the transcripts is not normally encoded and so it has to be inferred from the exon ranges presented. That's not a horrible problem, but it bears mentioning for the sake of full disclosure. And for gff3 files the situation is typically even worse since they usually

don't encode any information about the exon rank within a transcript. This is a serious oversight and so if you have an alternative to using this kind of data, you should really do so.

Some files will have an attribute defined to indicate the exon rank information. For GTF files this is usually given as "exon_number", however you still must specify this argument if you don't want the code to try and infer the exon rank information. For gff3 files, we have not seen any examples of this information encoded anywhere, but if you have a file with an attribute, you can still specify this to avoid the inference.

Value

A TxDb object.

Author(s)

M. Carlson

See Also

DEFAULT_CIRC_SEQS, makeTranscriptDbFromUCSC, makeTranscriptDbFromBiomart, makeTranscriptDb, supportedMiRBaseBuildValues

Examples

```
## TESTING GFF3
gffFile <- system.file("extdata","a.gff3",package="GenomicFeatures")
txdb <- makeTranscriptDbFromGFF(file=gffFile,
            format="gff3",
            exonRankAttributeName=NA,
            dataSource="partial gtf file for Tomatoes for testing",
            species="Solanum lycopersicum")
if(interactive()) {
saveDb(txdb,file="TESTGFF.sqlite")
}

## TESTING GTF, this time specifying the chrominfo
gtfFile <- system.file("extdata","Aedes_aegypti.partial.gtf",
                       package="GenomicFeatures")
chrominfo <- data.frame(chrom = c(supercont1.1,supercont1.2),
                        length=c(5220442, 5300000),
                        is_circular=c(FALSE, FALSE))
txdb2 <- makeTranscriptDbFromGFF(file=gtfFile,
             format="gtf",
             exonRankAttributeName="exon_number",
             chrominfo=chrominfo,
             dataSource=paste("ftp://ftp.ensemblgenomes.org/pub/metazoa/",
                              "release-13/gtf/aedes_aegypti/",sep=""),
             species="Aedes aegypti")
if(interactive()) {
    saveDb(txdb2,file="TESTGTF.sqlite")
}
```

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

Description

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

Usage

```
supportedUCSCtables()

getChromInfoFromUCSC(
        genome,
        goldenPath_url="http://hgdownload.cse.ucsc.edu/goldenPath")

makeTranscriptDbFromUCSC(
        genome="hg19",
        tablename="knownGene",
        transcript_ids=NULL,
        circ_seqs=DEFAULT_CIRC_SEQS,
        url="http://genome.ucsc.edu/cgi-bin/",
        goldenPath_url="http://hgdownload.cse.ucsc.edu/goldenPath",
        miRBaseBuild=NA)
```

Arguments

genome	genome abbreviation used by UCSC and obtained by ucscGenomes()[, "db"]. For example: "hg19".
tablename	name of the UCSC table containing the transcript annotations to retrieve. Use the supportedUCSCtables utility function to get the list of supported tables. Note that not all tables are available for all genomes.
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.
url,goldenPath_url	use to specify the location of an alternate UCSC Genome Browser.
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

makeTranscriptDbFromUCSC is a convenience function that feeds data from the UCSC source to the lower level makeTranscriptDb function. See ?makeTranscriptDbFromBiomart for a similar function that feeds data from a BioMart database.

Value

A TxDb object.

Author(s)

M. Carlson and H. Pages

See Also

ucscGenomes, DEFAULT_CIRC_SEQS, makeTranscriptDbFromBiomart, makeTranscriptDbFromGFF, makeTranscriptDb, supportedMiRBaseBuildValues

Examples

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

## Display the list of tables supported by makeTranscriptDbFromUCSC():
supportedUCSCtables()

## Not run:
## Retrieving a full transcript dataset for Yeast from UCSC:
txdb1 <- makeTranscriptDbFromUCSC(genome="sacCer2", tablename="ensGene")

## End(Not run)

## Retrieving 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 <- makeTranscriptDbFromUCSC(genome="mm9", tablename="knownGene",
                                  transcript_ids=transcript_ids)
txdb2
```

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=DEFAULT_CIRC_SEQS,
    url="http://genome.ucsc.edu/cgi-bin/",
    goldenPath_url="http://hgdownload.cse.ucsc.edu/goldenPath",
    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="http://hgdownload.cse.ucsc.edu/goldenPath",
    chromCol=NULL,
    chromStartCol=NULL,
    chromEndCol=NULL)
```

```
makeTxDbPackageFromBiomart(
  version,
  maintainer,
  author,
  destDir=".",
  license="Artistic-2.0",
  biomart="ensembl",
  dataset="hsapiens_gene_ensembl",
  transcript_ids=NULL,
  circ_seqs=DEFAULT_CIRC_SEQS,
  miRBaseBuild=NA)

makeTxDbPackage(txdb,
                version,
   maintainer,
                author,
           destDir=".",
                license="Artistic-2.0")

supportedMiRBaseBuildValues()
```

Arguments

version	What is the version number for this package?
maintainer	Who is the package maintainer? (must include email to be valid)
author	Who is the creator of this package?
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 supported tables. Note that not all tables are available for all genomes.
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.
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 makeTranscriptDbFromUCSC, makeTranscriptDbFromUCSC or makeTranscriptDb
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

makeTxDbPackageFromUCSC is a convenience function that calls both the makeTranscriptDbFromUCSC and the makeTxDbPackage functions. The makeTxDbPackageFromBiomart follows a similar pattern and calls the makeTranscriptDbFromBiomart and makeTxDbPackage functions. supportedMiRBaseBuildValues is a convenience function that will list all the possible values for the miRBaseBuild argument.

Value

A TxDb object.

Author(s)

M. Carlson

See Also

makeTranscriptDbFromUCSC, makeTranscriptDbFromBiomart, makeTranscriptDb, ucscGenomes, DEFAULT_CIRC_SEQS

Examples

```
## First consider relevant helper/discovery functions:
## Display the list of tables supported by makeTxDbPackageFromUCSC():
supportedUCSCtables()
```

```
## 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)
```

`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 behavior of distance 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 ids 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, 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)
```

`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)
```

---

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)

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?

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)

H. Pages

See Also

extractTranscriptSeqs for extracting transcript sequences from chromosomes.

Examples

```
## ---------------------------------------------------------------------
## GOING FROM TRANSCRIPT-BASED TO REFERENCE-BASED LOCATIONS
## ---------------------------------------------------------------------
library(BSgenome.Hsapiens.UCSC.hg19)  # load the genome
genome <- BSgenome.Hsapiens.UCSC.hg19
txdb_file <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                         package="GenomicFeatures")
txdb <- loadDb(txdb_file)
transcripts <- exonsBy(txdb, by="tx", use.names=TRUE)
tx_seqs <- extractTranscriptSeqs(genome, transcripts)

## 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(transcripts), runValue)
## Note that, because of how we made them, tlocs, start(exbytx),
## end(exbytx) and tx_strand 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(transcripts), end(transcripts),
             tx_strand, decreasing.rank.on.minus.strand=TRUE)

## ---------------------------------------------------------------------
## EXTRACTING WORM TRANSCRIPTS ZC101.3 AND F37B1.1
## ---------------------------------------------------------------------
## Transcript ZC101.3 (is on + strand):
##   Exons starts/ends relative to transcript:
rstarts1 <- c(1, 488, 654, 996, 1365, 1712, 2163, 2453)
rends1 <- c(137, 578, 889, 1277, 1662, 1870, 2410, 2561)
##   Exons starts/ends relative to chromosome:
starts1 <- 14678410 + rstarts1
ends1 <- 14678410 + rends1

## Transcript F37B1.1 (is on - strand):
##   Exons starts/ends relative to transcript:
rstarts2 <- c(1, 325)
rends2 <- c(139, 815)
##   Exons starts/ends relative to chromosome:
starts2 <- 13611188 - rends2
ends2 <- 13611188 - rstarts2

exon_starts <- list(as.integer(starts1), as.integer(starts2))
exon_ends <- list(as.integer(ends1), as.integer(ends2))
transcripts <- IRangesList(start=exon_starts, end=exon_ends)

library(BSgenome.Celegans.UCSC.ce2)
## Both transcripts are on chrII:
chrII <- Celegans$chrII
tx_seqs <- extractTranscriptSeqs(chrII, transcripts, strand=c("+","-"))

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

transcriptLocs2refLocs(list(c(1:6, 135:140, 1555:1560),
                            c(1:6, 137:142, 625:630)),
                       exonStarts=exon_starts,
                       exonEnds=exon_ends,
                       strand=c("+","-"))

## A sanity check:
ref_locs <- transcriptLocs2refLocs(list(1:1560, 1:630),
                                   exonStarts=exon_starts,
                                   exonEnds=exon_ends,
                                   strand=c("+","-"))
stopifnot(chrII[ref_locs[[1]]] == tx_seqs[[1]])
stopifnot(complement(chrII)[ref_locs[[2]]] == tx_seqs[[2]])
```

transcripts *Extract genomic features from an object*

Description

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

Usage

```
transcripts(x, ...)
## S4 method for signature TxDb
transcripts(x, vals=NULL, columns=c("tx_id", "tx_name"))

exons(x, ...)
## S4 method for signature TxDb
exons(x, vals=NULL, columns="exon_id")

cds(x, ...)
## S4 method for signature TxDb
cds(x, vals=NULL, columns="cds_id")

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

#promoters(x, upstream=2000, downstream=200, ...)
## S4 method for signature TxDb
promoters(x, upstream=2000, downstream=200, ...)

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

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

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

Arguments

x	A TxDb object.
...	Arguments to be passed to or from methods.

vals
: 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".

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.

single.strand.genes.only
: TRUE or FALSE. If TRUE (the default), then genes that have exons located on both strands of the same chromosome or on two different chromosomes are dropped. In that case, the genes are returned in a GRanges object. Otherwise, all 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.

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

These are the main functions for extracting transcript information from a TxDb object. With the exception of microRNAs, 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 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.

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. 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. Pages. disjointExons was contributed by Mike Love and Alejandro Reyes.

See Also

- transcriptsBy and transcriptsByOverlaps for more ways to extract genomic features from a TxDb object.
- 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

```
## transcripts(), exons(), genes():
txdb <- loadDb(system.file("extdata", "hg19_knownGene_sample.sqlite",
                           package="GenomicFeatures"))
vals <- list(tx_chrom = c("chr3", "chr5"), tx_strand = "+")

transcripts(txdb, vals)

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

genes(txdb)  # a GRanges object
cols <- c("tx_id", "tx_chrom", "tx_strand",
          "exon_id", "exon_chrom", "exon_strand")
single_strand_genes <- genes(txdb, columns=cols)

## Because weve 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[elementLengths(all_genes) >= 2]
multiple_strand_genes
mcols(multiple_strand_genes)

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

## End(Not run)

## promoters() :
head(promoters(txdb, 100, 50))
```

`transcriptsBy` *Extract and group genomic features of a given type*

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 and cdsBy with by = "tx", the ranges are returned in the order they appear in the transcript, i.e. order by the splicing.exon_rank field in x's internal database. 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. Pages

See Also

- transcripts and transcriptsByOverlaps for more ways to extract genomic features from a TxDb object.
- 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 an object based on their by 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 = 0L, minoverlap = 1L,
                      type = c("any", "start", "end"), ...)
## S4 method for signature TxDb
transcriptsByOverlaps(x, ranges,
                      maxgap = 0L, minoverlap = 1L,
                      type = c("any", "start", "end"),
                      columns = c("tx_id", "tx_name"))

exonsByOverlaps(x, ranges,
                maxgap = 0L, minoverlap = 1L,
                type = c("any", "start", "end"), ...)
```

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

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

Arguments

x	A TxDb object.
...	Arguments to be passed to or from methods.
ranges	A GRanges object to restrict the output.
type	How to perform the interval overlap operations of the ranges. See the findOverlaps manual page in the GRanges package for more information.
maxgap	A non-negative integer representing the maximum distance between a query interval and a subject interval.
minoverlap	Ignored.
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 object.
- 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(seqnames = rep("chr1",2),
              ranges = IRanges(start=c(500,10500), end=c(10000,30000)),
              strand = strand(rep("-",2)))
transcriptsByOverlaps(txdb, gr)
```

TxDb-class *TxDb objects*

Description

The TxDb class is a container for storing transcript annotations.

See ?FeatureDb for a more generic container for storing genomic locations of an arbitrary type of genomic features.

See ?makeTranscriptDbFromUCSC and ?makeTranscriptDbFromBiomart for convenient ways to make TxDb objects from UCSC or BioMart online resources.

See ?makeTranscriptDbFromGFF for making a TxDb object from annotations available as a GFF3 or GTF file.

Methods

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

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

`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)`: Dumps the entire db into a list of data frames `txdump` that can be used in `do.call(makeTranscriptDb, txdum` to make the db again with no loss of information. Note that the transcripts are dumped in the same order in all the data frames.

Author(s)

H. Pages, Marc Carlson

See Also

- The FeatureDb class for storing genomic locations of an arbitrary type of genomic features.
- `makeTranscriptDbFromUCSC` and `makeTranscriptDbFromBiomart` for convenient ways to make TxDb objects from UCSC or BioMart online resources.
- `makeTranscriptDbFromGFF` for making a TxDb object from annotations available as a GFF3 or GTF file.
- `saveDb` and `loadDb` for saving and loading the database content of a TxDb object.
- `transcripts`, `transcriptsBy`, and `transcriptsByOverlaps`, for how to extract genomic features 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)))

## Can set txdb so that only chr1 and chr2 are used by using the seqlevels
seqlevels(txdb, force=TRUE) <- c("1", "2")
## And then you can restore the default seqlevels
txdb <- restoreSeqlevels(txdb)


## Use of as.list
txdump <- as.list(txdb)
txdump
txdb1 <- do.call(makeTranscriptDb, txdump)
stopifnot(identical(as.list(txdb1), txdump))
```

Index