Package 'GenomicFeatures'

October 5, 2015

Title Tools for making and manipulating transcript centric annotations

Version 1.20.6

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 (>= 2.1.36), GenomeInfoDb (>= 1.4.3), GenomicRanges (>= 1.17.12), AnnotationDbi (>= 1.27.9)

Imports methods, utils, tools, DBI (>= 0.2-5), RSQLite (>= 0.8-1), RCurl, XVector, Biostrings (>= 2.23.3), rtracklayer (>= 1.25.2), biomaRt (>= 2.17.1), Biobase (>= 2.15.1)

Suggests org.Mm.eg.db, org.Hs.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), TxDb.Mmusculus.UCSC.mm10.knownGene, TxDb.Hsapiens.UCSC.hg19.lincRNAsTranscripts, TxDb.Hsapiens.UCSC.hg38.knownGene, SNPlocs.Hsapiens.dbSNP141.GRCh38, Rsamtools, pasillaBamSubset (>= 0.0.5), RUnit, BiocStyle, knitr

Collate utils.R Ensembl.utils.R findCompatibleMarts.R TxDb-class.R FeatureDb-class.R makeTxDb.R makeTxDbFromUCSC.R makeTxDbFromBiomart.R makeTxDbFromGRanges.R makeTxDbFromGFF.R makeFeatureDbFromUCSC.R id2name.R transcripts.R transcriptsBy.R transcriptsByOverlaps.R transcriptLengths.R features.R extractTranscriptSeqs.R extractUpstreamSeqs.R getPromoterSeq-methods.R makeTxDbPackage.R select-methods.R nearest-methods.R transcriptLocs2refLocs.R coordinate-mapping-methods.R sortExonsByRank.R test_GenomicFeatures_package.R

VignetteBuilder knitr

biocViews Genetics, Infrastructure, Annotation, Sequencing, GenomeAnnotation

NeedsCompilation no

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

makeTxDbFromUCSC, makeTxDbFromBiomart

Examples

```
DEFAULT_CIRC_SEQS
```

extractTranscriptSeqs *Extract transcript sequences from chromosomes*

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 RangesList 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 object). If the latter, then it's first turned into a GRangesList object with `exonsBy(transcripts, by="tx", use.names=TRUE)`. Note that, 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` (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.
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 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 RangesList 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)))))

## ---------------------------------------------------------------------
## 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:
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)))

## ---------------------------------------------------------------------
## 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. See the makeTxDbFromUCSC, makeTxDbFromBiomart, and makeTxDbFromGFF 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 makeTxDbFromUCSC, makeTxDbFromBiomart, and makeTxDbFromGFF 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)
```

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
```

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. See ?makeTxDbFromUCSC and ?makeTxDbFromBiomart for higher-level functions that feed data from the UCSC or BioMart sources to makeTxDb.

Usage

```
makeTxDb(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 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_type: [optional] Transcript type (e.g. mRNA, ncRNA, snoRNA, etc...). 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). 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 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 and/or duplicates are ok. Must be NA if corresponding cds_start and cds_end are NAs.
- 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. 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)

H. Pages

See Also

- makeTxDbFromUCSC, makeTxDbFromBiomart, makeTxDbFromGRanges, and makeTxDbFromGFF, for convenient ways to make a TxDb object from UCSC or BioMart online resources, or from a GRanges object, or from a GFF 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 <- 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.

Usage

```
makeTxDbFromBiomart(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=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.biomart.org.
port	The port to use in the HTTP communication with the host.
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.

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) makeTxDbFromBiomart. 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 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. Pages

See Also

- makeTxDbFromUCSC, makeTxDbFromGRanges, and makeTxDbFromGFF, for convenient ways to make a TxDb object from UCSC online resources, or from a GRanges object, or from a GFF or GTF file.
- The listMarts, useMart, listDatasets, and listFilters functions in the **biomaRt** package.
- DEFAULT_CIRC_SEQS.
- 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" BioMart database:
library(biomaRt)
head(listDatasets(useMart("ensembl")))

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

## ---------------------------------------------------------------------
## B. USING A HOST OTHER THAN www.biomart.org
## ---------------------------------------------------------------------

## A typical use case is to access the "ensembl" BioMart database on a
## mirror e.g. on uswest.ensembl.org. A gotcha when doing this is that
## the name of the database on the mirror can be different! We can check
## this with listMarts() from the biomaRt package:
listMarts(host="uswest.ensembl.org")

## Therefore, in addition to setting 'host' to "uswest.ensembl.org" we
## must also change the name passed to the 'biomart' argument:
txdb3 <- makeTxDbFromBiomart(biomart="ENSEMBL_MART_ENSEMBL",
                             dataset="hsapiens_gene_ensembl",
                             transcript_ids=transcript_ids,
                             host="uswest.ensembl.org")
txdb3

## ---------------------------------------------------------------------
## C. USING FILTERS
## ---------------------------------------------------------------------

## We can use listFilters() from the biomaRt package to get valid filter
## names:
mart <- useMart("ensembl", dataset="hsapiens_gene_ensembl")
```

```
head(listFilters(mart))

## Retrieve transcript dataset for Ensembl gene ENSG00000011198:
my_filter <- list(ensembl_gene_id="ENSG00000011198")
txdb4 <- makeTxDbFromBiomart(dataset="hsapiens_gene_ensembl",
                             filters=my_filter)
txdb4
transcripts(txdb4, columns=c("tx_id", "tx_name", "gene_id"))
transcriptLengths(txdb4)

## ---------------------------------------------------------------------
## D. RETRIEVING CHROMOSOME INFORMATION ONLY
## ---------------------------------------------------------------------

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

`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,
                circ_seqs=DEFAULT_CIRC_SEQS,
                chrominfo=NULL,
                miRBaseBuild=NA,
                exonRankAttributeName=NA,
                gffGeneIdAttributeName=NA,
                useGenesAsTranscripts=FALSE,
                gffTxName="mRNA",
                species=NA)
```

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.

`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.

`exonRankAttributeName`
ignored and deprecated

`gffGeneIdAttributeName`
ignored and deprecated

`useGenesAsTranscripts`
ignored and deprecated

`gffTxName` ignored and deprecated

`species` deprecated in favor of `organism`

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. Pages

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` and `makeTxDbFromBiomart` for convenient ways to make a TxDb object from UCSC or BioMart online resources.
- `DEFAULT_CIRC_SEQS`.
- 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))
txdb2 <- makeTxDbFromGFF(file=gtfFile,
             chrominfo=chrominfo,
             dataSource=paste("ftp://ftp.ensemblgenomes.org/pub/metazoa/",
                              "release-13/gtf/aedes_aegypti/",sep=""),
             organism="Aedes aegypti")
```

---

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)
```

Arguments

gr	A GRanges object structured as GFF3 or GTF, typically obtained with rtracklayer::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.

Value

A TxDb object.

Author(s)

H. Pages

See Also

- makeTxDbFromUCSC, makeTxDbFromBiomart, and makeTxDbFromGFF, for convenient ways to make a TxDb object from UCSC or BioMart online resources, or directly from a GFF or GTF file.
- The import function in the **rtracklayer** 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(rtracklayer)  # 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:
stopifnot(identical(as.list(txdb), 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.

Usage

```
supportedUCSCtables()

makeTxDbFromUCSC(
        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)

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

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

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

Value

A TxDb object for `makeTxDbFromUCSC`.

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

Author(s)

M. Carlson and H. Pages

See Also

- `makeTxDbFromBiomart`, `makeTxDbFromGRanges`, and `makeTxDbFromGFF`, for convenient ways to make a TxDb object from BioMart online resources, or from a GRanges object, or from a GFF or GTF file.
- `ucscGenomes` in the **rtracklayer** package.
- `DEFAULT_CIRC_SEQS`.
- 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 supported by makeTxDbFromUCSC():
supportedUCSCtables()

## Retrieve a full transcript dataset for Yeast from UCSC:
```

```
txdb1 <- makeTxDbFromUCSC(genome="sacCer3", tablename="ensGene",
                          circ_seqs="chrM")
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(genome="mm10", tablename="knownGene",
                          transcript_ids=transcript_ids,
                          circ_seqs="chrM")
txdb2

## ---------------------------------------------------------------------
## B. RETRIEVING CHROMOSOME INFORMATION ONLY
## ---------------------------------------------------------------------

chrominfo <- getChromInfoFromUCSC(genome="hg38")
chrominfo
```

`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,
    filters="",
    id_prefix="ensembl_",
    host="www.biomart.org",
    port=80,
    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.
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_".
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
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 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.

Value

A TxDb object.

Author(s)

M. Carlson

See Also

makeTxDbFromUCSC, makeTxDbFromBiomart, makeTxDb, 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)
```

`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

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

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

Arguments

x	GenomicRanges object of positions to be mapped. x must have names when mapping to the genome.
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 to the genome the names are used to determine mapping pairs and in the reverse direction they become the seqlevels of the output object.
ignore.strand	When TRUE, strand is ignored in overlap operations. When transcripts is a GRangesList and ignore.strand = FALSE all inner list elements of a common list element must have the same strand.
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 or ?transcriptsBy man page. Valid extractor functions: • transcripts ## default • exons • cds • genes • promoters • disjointExons • microRNAs • tRNAs • transcriptsBy • exonsBy • cdsBy • intronsByTranscript • fiveUTRsByTranscript • threeUTRsByTranscript
...	Additional arguments passed to extractor.fun functions.

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.

- `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 genomic chromosomes.

  When mapping to the genome, name matching is used to determine the mapping pairs (vs attempting to match all possible pairs). Ranges in `x` are only mapped to ranges in `transcripts` with the same name. 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`, which may contains gene or transcript ranges.

- element-wise versions `pmapToTranscripts` and `pmapFromTranscripts` are element-wise (aka 'parallel') versions of `mapToTranscripts` and `mapFromTranscripts`. The i-th range in `x` is mapped to the i-th range in `transcripts`; `x` and `transcripts` must have the same length.

  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 special seqname of "UNMAPPED". Note the non-parallel methods do not return unmapped ranges so the "UNMAPPED" seqname is unique to `pmapToTranscipts` and `pmapFromTranscripts`.

Value

An object the same class as `x`.

Parallel methods return an object the same shape as `x`. Ranges that cannot be mapped (out of bounds or strand mismatch) are returned as zero-width ranges starting at 0 with a seqname of "UNMAPPED".

Non-parallel methods return an object that varies in length similar to a Hits object. The result only contains mapped records, strand mismatch and out of bound ranges are not returned. `xHits` and `transcriptsHits` metadata columns indicate the elements of `x` and `transcripts` used in the mapping.

When present, names from `x` are propagated to the output. When mapping to transcript coordinates, seqlevels of the output are the names on the `transcripts` object; 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. Pages

See Also

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

Examples

```
## ---------------------------------------------------------------------
## A. Basic Use
## ---------------------------------------------------------------------

library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
cds <- cdsBy(TxDb.Dmelanogaster.UCSC.dm3.ensGene, "tx", use.names=TRUE)[1:3]
x <- GRanges("chr2L",
             IRanges(c(7500, 8400, 9000), width=200,
                     names=LETTERS[1:3]))

## Map to transcript coordinates:
mapToTranscripts(x, cds)

## Note the seqnames of the output are the transcript names, not
## chromosomes. The 'xHits' and 'transcriptsHits' metadata
## columns indicate which elements were involved in the mapping.

## The element-wise version returns both mapped and unmapped ranges.
x <- GRanges("chr2L", IRanges(c(8100, 8600), width=10, names=LETTERS[1:2]))
pmapToTranscripts(x, cds[1:2])

## Mapping to genome space requires both 'x' and 'transcripts' to have
## names. A map is only attempted for ranges with matching names.
x <- GRanges("chr1",
             IRanges(c(1, 20, 10, 3), width=5, names=c("A", "B", "A", "C")))
gr <- GRanges("chr1",
              IRanges(c(25, 30, 3), width=10, names=c("C", "C", "A")))
mapFromTranscripts(x, gr)

## ---------------------------------------------------------------------
## B. Map local sequence locations 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 3'UTR variants to genome coordinates
## ---------------------------------------------------------------------

## A study by Skeeles et. al (PLoS ONE 8(3): e58609. doi:
## 10.1371/journal.pone.0058609) investigated the impact of 3'UTR variants
## on the expression of cancer susceptibility genes.

## 8 candidate miRNA genes on chromosome 12 were used to test for
## differential luciferase expression in mice. In Table 2 of the manuscript
## variant locations are given as nucleotide position within the gene.
geneNames <- c("Bcap29", "Dgkb", "Etv1", "Hbp1", "Hbp1", "Ifrd1",
               "Ifrd1", "Pik3cg", "Pik3cg", "Tspan13", "Twistnb")
starts <- c(1409, 3170, 3132, 2437, 2626, 3239, 3261, 4947, 4979, 958, 1489)
snps <- GRanges("chr12", IRanges(starts, width=1, names=geneNames))

## To map these transcript-space coordinates to the genome we need gene ranges
## in genome space.
library(org.Mm.eg.db)
geneid <- select(org.Mm.eg.db, unique(geneNames), "ENTREZID", "SYMBOL")
geneid

## Extract the gene regions:
library(TxDb.Mmusculus.UCSC.mm10.knownGene)
txdb <- TxDb.Mmusculus.UCSC.mm10.knownGene
genes <- genes(txdb)[geneid$ENTREZID]

## mapToTranscripts(..., reverse=TRUE) determines pairs to map by comparing
## names in 'x' and 'transcripts'. Ranges in 'snps' will be mapped to all
## ranges in 'genes' with the same name. Currently the names of 'genes' are
## internal gene ids. Rename 'genes' with the appropriate gene symbol.
names(genes) <- geneid$SYMBOL
```

```
## The xHits and transcriptsHits metadta columns indicate which ranges in
## 'snps' and 'genes' were involved in the mapping.
mapFromTranscripts(snps, genes)


## ---------------------------------------------------------------------
## D. 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.dbSNP141.GRCh38)
snps <- snpid2grange(SNPlocs.Hsapiens.dbSNP141.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. Conversion between gene symbols and Entrez gene IDs is done by
## calling select() on an organism db package.
library(org.Hs.eg.db)
geneid <- select(org.Hs.eg.db, "GIPR", "ENTREZID", "SYMBOL")

## 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
txbygene <- transcriptsBy(txdb, "gene")
cDNA <- txbygene[geneid$ENTREZID]
cDNA

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

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

## The 'cDNA' object is a GRangesList of length 1. This single list element
## contains the cDNA range for 4 different transcripts. To map to each
## transcript individually 'cDNA' must be unlisted before mapping.

## Map all 5 SNPS to all 4 transcripts:
mapToTranscripts(snps, unlist(cDNA))
```

```
## Map the first SNP to transcript uc002pct.1 and the second to uc002pcu.1.
pmapToTranscripts(snps[1:2], unlist(cDNA)[1:2])

## The cdsBy() extractor returns coding regions by gene or by transcript.
## Extract the coding regions for transcript uc002pct.1.
cds <- cdsBy(txdb, "tx", use.names=TRUE)["uc002pct.1"]
cds

## The 'cds' object is a GRangesList of length 1 containing all cds ranges
## for the single transcript uc002pct.1. To map to the concatenated group
## of ranges we leave 'cds' as a GRangesList.

## Map all 5 SNPs to the total cds region:
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 had been on the negative strand the difference in mapping

## would be more obvious.
strand(snps) <- strand(cds) <- "-"
mapToTranscripts(snps[2], cds, ignore.strand=FALSE)
mapToTranscripts(snps[2], unlist(cds), ignore.strand=FALSE)
```

`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 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, 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)
```

`sortExonsByRank` *Sort exons by rank*

Description

WARNING: Starting with BioC 3.0, `sortExonsByRank` is defunct with no replacement.

`sortExonsByRank` orders (or reorders) by rank the exons stored in a GRangesList object containing exons grouped by transcript.

Usage

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

Arguments

`x`	A GRangesList object, typically coming from `exonsBy(... , by="tx")`.
`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.

Value

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

---

transcriptLengths *Extract the transcript lengths 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.

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)

H. Pages

See Also

- `transcripts`, `transcriptsBy`, and `transcriptsByOverlaps`, for how to extract the genomic locations of features from a TxDb object.
- `makeTxDbFromUCSC` and `makeTxDbFromBiomart` for convenient ways to make TxDb objects from UCSC or BioMart online resources.
- 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]]))
```

---

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)

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)

## 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.
...	For the transcripts, exons, cds, genes, and disjointExons 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.
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.
- transcriptLengths for extracting the transcript lengths 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)

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

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

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

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

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

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 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[elementLengths(all_genes) >= 2]
multiple_strand_genes
mcols(multiple_strand_genes)

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

```
## This:
promoters(txdb, upstream=100, downstream=50)
## is equivalent to:
promoters(transcripts(txdb), upstream=100, downstream=50)

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

## ---------------------------------------------------------------------
## microRNAs()
## ---------------------------------------------------------------------

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

## End(Not run)
```

`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 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. 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 ?makeTxDbFromUCSC and ?makeTxDbFromBiomart for convenient ways to make TxDb objects from UCSC or BioMart online resources.

See ?makeTxDbFromGFF 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)`: 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)

H. Pages, Marc Carlson

See Also

- `makeTxDbFromUCSC`, `makeTxDbFromBiomart`, `makeTxDbFromGRanges`, and `makeTxDbFromGFF`, for convenient ways to make a TxDb object from UCSC or BioMart online resources, or from a GRanges object, or from a GFF 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.
- `transcriptLengths` for extracting the transcript lengths from a TxDb object.
- select-methods for how to use the simple "select" interface to extract information from a TxDb object.
- The FeatureDb class for storing genomic locations of an arbitrary type of genomic features.
- 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:
txdb <- restoreSeqlevels(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