GenomicFeatures

October 25, 2011

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

Description

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

Usage

```
DEFAULT_CIRC_SEQS
```

See Also

`makeTranscriptDbFromUCSC`, `makeTranscriptDbFromBiomart`

Examples

```
DEFAULT_CIRC_SEQS
```

`TranscriptDb-class` *TranscriptDb objects*

Description

The TranscriptDb class is a container for storing transcript annotations. The FeatureDb class is a container for storing more generic GenomicFeature annotations.

See `?makeTranscriptDbFromUCSC` and `?makeTranscriptDbFromBiomart` for making a TranscriptDb object from the UCSC or BioMart sources.

See `?makeFeatureDbFromUCSC` for making a FeatureDb object from the UCSC or BioMart sources.

See `?saveFeatures` and `?loadFeatures` for saving and loading the database contents of a TranscriptDb or FeatureDb object.

Methods

In the code snippets below, `x` is a TranscriptDb object. For the metadata and show methods, there is also support for FeatureDb objects.

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

`seqinfo(x)`: Gets the information about the underlying sequences as a Seqinfo object.

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

See `?transcripts`, `?transcriptsByOverlaps`, `?id2name` and `?transcriptsBy` for other useful operations on TranscriptDb objects.

Author(s)

H. Pages

See Also

Seqinfo-class, `makeTranscriptDbFromUCSC`, `makeTranscriptDbFromBiomart`, `loadFeatures`, `transcripts`, `transcriptsByOverlaps`, `id2name`, `transcriptsBy`

Examples

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

seqinfo(txdb)
seqlevels(txdb)  # shortcut for 'seqlevels(seqinfo(txdb))'
seqlengths(txdb)  # shortcut for 'seqlengths(seqinfo(txdb))'
isCircular(txdb)  # shortcut for 'isCircular(seqinfo(txdb))'
names(which(isCircular(txdb)))

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

`extractTranscriptsFromGenome` *Tools for extracting transcript sequences*

Description

`extractTranscriptsFromGenome` extracts the transcript sequences from a BSgenome data package using the transcript information (exon boundaries) stored in a TranscriptDb or GRangesList object.

`extractTranscripts` extracts a set of transcripts from a single DNA sequence.

Related utilities:

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

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

Usage

```
extractTranscriptsFromGenome(genome, txdb, use.names=TRUE)

extractTranscripts(x,
        exonStarts=list(), exonEnds=list(), strand=character(0),
        reorder.exons.on.minus.strand=FALSE)

## Related utilities:

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

transcriptLocs2refLocs(tlocs,
        exonStarts=list(), exonEnds=list(), strand=character(0),
        reorder.exons.on.minus.strand=FALSE)
```

Arguments

`genome`	A BSgenome object. See the `available.genomes` function in the BSgenome package for how to install a genome.
`txdb`	A TranscriptDb object or a GRangesList object.
`use.names`	`TRUE` or `FALSE`. Ignored if `txdb` is not a TranscriptDb object. If `TRUE` (the default), the returned sequences are named with the transcript names. If `FALSE`, they are named with the transcript internal ids. Note that, unlike the transcript internal ids, the transcript names are not guaranteed to be unique or even defined (they could be all `NA`s). A warning is issued when this happens.
`x`	A DNAString or MaskedDNAString object.
`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.
`reorder.exons.on.minus.strand`	`TRUE` or `FALSE`. Should the order of exons for transcripts coming from the minus strand be reversed?
`tlocs`	A list of integer vectors of the same length as `exonStarts` and `exonEnds`. Each element in `tlocs` must contain transcript-based locations.

Value

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

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

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

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

Author(s)

H. Pages

See Also

`available.genomes`, TranscriptDb-class, GRangesList-class, DNAStringSet-class

Examples

```
library(BSgenome.Hsapiens.UCSC.hg18)  # load the genome

## ---------------------------------------------------------------------
## A. USING extractTranscriptsFromGenome() WITH A TranscriptDb OBJECT
## ---------------------------------------------------------------------
txdb_file <- system.file("extdata", "UCSC_knownGene_sample.sqlite",
                         package="GenomicFeatures")
txdb <- loadFeatures(txdb_file)
txseqs <- extractTranscriptsFromGenome(Hsapiens, txdb)
txseqs

## ---------------------------------------------------------------------
## B. USING extractTranscriptsFromGenome() WITH A GRangesList OBJECT
## ---------------------------------------------------------------------

## A GRangesList object containing exons grouped by transcripts gives
## the same result as above:
exbytx <- exonsBy(txdb, by="tx", use.names=TRUE)
txseqs2 <- extractTranscriptsFromGenome(Hsapiens, exbytx)
## A sanity check:
stopifnot(identical(unname(sapply(width(exbytx), sum)), width(txseqs2)))

## CDSs grouped by transcripts (this extracts only the translated parts
## of the transcripts):
cds <- extractTranscriptsFromGenome(Hsapiens, cdsBy(txdb))

## ---------------------------------------------------------------------
## C. GOING FROM TRANSCRIPT-BASED TO REFERENCE-BASED LOCATIONS
## ---------------------------------------------------------------------
## Get the reference-based locations of the first 4 (5' end)
## and last 4 (3' end) nucleotides in each transcript:
tlocs <- lapply(width(txseqs2), function(w) c(1:4, (w-3):w))
tx_strand <- sapply(strand(exbytx), 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(exbytx), end(exbytx),
             tx_strand, reorder.exons.on.minus.strand=TRUE)

## ---------------------------------------------------------------------
## D. 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))

library(BSgenome.Celegans.UCSC.ce2)
## Both transcripts are on chrII:
chrII <- Celegans$chrII
transcripts <- extractTranscripts(chrII,
                                  exonStarts=exon_starts,
                                  exonEnds=exon_ends,
                                  strand=c("+","-"))

## Same as 'width(transcripts)':
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]]] == transcripts[[1]])
stopifnot(complement(chrII)[ref_locs[[2]]] == transcripts[[2]])
```

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

`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 TranscriptDb object.
`feature.type`	The feature type for which the mapping must be retrieved.

Details

Transcripts, exons and CDS in a TranscriptDb 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 TranscriptDb 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

TranscriptDb, transcripts, transcriptsBy

Examples

```
txdb1_file <- system.file("extdata", "UCSC_knownGene_sample.sqlite",
                          package="GenomicFeatures")
txdb1 <- loadFeatures(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 <- loadFeatures(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

---

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")
```

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 `supportedUCSCFeature` 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.

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:
txdb <- makeFeatureDbFromUCSC(genome="mm9",
                              track="oreganno",
                              tablename="oreganno")
txdb
```

`makeTranscriptDb` *Making a TranscriptDb object from user supplied annotations*

Description

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

Usage

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

Arguments

`transcripts`	data frame containing the genomic locations of a set of transcripts
`splicings`	data frame containing the exon and cds locations of a set of transcripts
`genes`	data frame containing the genes associated to a set of transcripts
`chrominfo`	data frame containing information about the chromosomes hosting the set of transcripts
`metadata`	2-column data frame containing meta information about this set of transcripts like species, organism, genome, UCSC table, etc... The names of the columns must be `"name"` and `"value"` and their type must be character.
`...`	ignored for now

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).
- tx_chrom: Transcript chromosome. Character vector (or factor) with no NAs.
- tx_strand: Transcript strand. Character vector (or factor) 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 eventually the cds contained in this exon. Its columns must be:

- tx_id: Foreign key that links each row in the splicings data frame to a unique row in the transcripts data frame. Note that more than 1 row in splicings can be linked to the same row in transcripts (many-to-one relationship). Same type as transcripts$tx_id (integer vector). No NAs. All the values in this column must be present in transcripts$tx_id.
- exon_rank: The rank of the exon in the transcript. Integer vector with no NAs. (tx_id, exon_rank) pairs must be unique.
- exon_id: [optional] Exon ID. Integer vector with no NAs.
- exon_name: [optional] Exon name. Character vector (or factor).
- exon_chrom: [optional] Exon chromosome. Character vector (or factor) with no NAs. If missing then transcripts$tx_chrom is used. If present then exon_strand must be present too.
- exon_strand: [optional] Exon strand. Character vector (or factor) with no NAs. If missing then transcripts$tx_strand is used and exon_chrom must be missing too.
- 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 be too. 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 be too. NAs are allowed and must match NAs in cds_start and cds_end.
- cds_start, cds_end: [optional] cds start and end. Integer vectors. If one of the 2 columns is missing then all cds_* columns must be missing. NAs are allowed and must occur at the same positions in cds_start and cds_end.

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

genes must have N rows per transcript, where N is the nb of genes linked to the transcript (N will be 1 most of the time). Its columns must be:

- tx_id: [optional] genes must have either a tx_id or a tx_name column but not both. Like splicings$tx_id, this is a foreign key that links each row in the genes data frame to a unique row in the transcripts data frame.

- `tx_name`: [optional] Can be used as an alternative to the `genes$tx_id` foreign key.
- `gene_id`: Gene ID. Character vector (or factor). No NAs.

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

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

- `chrom`: Chromosome name. Character vector (or factor) with no NAs.
- `length`: Chromosome length. Either all NAs or an integer vector with no NAs.
- `is_circular`: [optional] Chromosome circularity flag. Either all NAs or a logical vector with no NAs.

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

Value

A TranscriptDb object.

Author(s)

H. Pages

See Also

`TranscriptDb`, `makeTranscriptDbFromUCSC`, `makeTranscriptDbFromBiomart`

Examples

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

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

`makeTranscriptDbFromBiomart`
Making a TranscriptDb object from annotations available on a BioMart

---

Description

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

Usage

```
getChromInfoFromBiomart(biomart="ensembl",
                        dataset="hsapiens_gene_ensembl")

makeTranscriptDbFromBiomart(biomart="ensembl",
                            dataset="hsapiens_gene_ensembl",
                            transcript_ids=NULL,
                            circ_seqs=DEFAULT_CIRC_SEQS)
```

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 TranscriptDb 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.

Details

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

As of November 30, 2010, the BioMart databases with compatible transcript annotations are:

- ensembl: ENSEMBL GENES 60 (SANGER UK)
- bacterial_mart_7: ENSEMBL BACTERIA 7 (EBI UK)
- fungal_mart_7: ENSEMBL FUNGAL 7 (EBI UK)
- metazoa_mart_7: ENSEMBL METAZOA 7 (EBI UK)
- plant_mart_7: ENSEMBL PLANT 7 (EBI UK)
- protist_mart_7: ENSEMBL PROTISTS 7 (EBI UK)
- ensembl_expressionmart_48: EURATMART (EBI UK)
- Ensembl56: PANCREATIC EXPRESSION DATABASE (INSTITUTE OF CANCER UK)

Only ensembl and Ensembl56 have CDS information.

Value

A TranscriptDb object.

Author(s)

M. Carlson and H. Pages

See Also

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

Examples

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

## Retrieving an incomplete transcript dataset for Human from the
## "ensembl" BioMart database:
transcript_ids <- c(
    "ENST00000268655",
    "ENST00000313243",
    "ENST00000341724",
    "ENST00000400839",
    "ENST00000400840",
    "ENST00000435657",
    "ENST00000478783"
)
txdb <- makeTranscriptDbFromBiomart(transcript_ids=transcript_ids)
txdb  # note that these annotations match the GRCh37 genome assembly
```

`makeTranscriptDbFromUCSC` *Making a TranscriptDb object from annotations available at the UCSC*

Description

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

Usage

```
supportedUCSCtables()

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

makeTranscriptDbFromUCSC(
        genome="hg18",
        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")
```

Arguments

`genome`	genome abbreviation used by UCSC and obtained by `ucscGenomes()[ , "db"]`. For example: `"hg18"`.
`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 TranscriptDb 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.

Details

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

Value

A TranscriptDb object.

Author(s)

M. Carlson and H. Pages

See Also

`ucscGenomes`, `DEFAULT_CIRC_SEQS`, `makeTranscriptDbFromBiomart`, `makeTranscriptDb`

Examples

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

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

## Retrieving a full transcript dataset for Yeast from UCSC:
txdb1 <- makeTranscriptDbFromUCSC(genome="sacCer2", tablename="ensGene")
txdb1

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

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

`regions` *Functions that compute genomic regions of interest.*

Description

Functions that compute genomic regions of interest such as promotor, upstream regions etc, from the genomic locations provided in a UCSC-style data frame.

Usage

```
transcripts_deprecated(genes, proximal = 500, distal = 10000)
exons_deprecated(genes)
introns_deprecated(genes)
```

Arguments

`genes`	A UCSC-style data frame i.e. a data frame with 1 row per transcript and at least the following columns: `"name"`, `"chrom"`, `"strand"`, `"txStart"`, `"txEnd"`, `"exonCount"`, `"exonStarts"`, `"exonEnds"`, `"intronStarts"` and `"intronEnds"`. A value in any of the last 4 columns must be a comma-separated list of integers. Note that unlike what UCSC does the start values here must be 1-based, not 0-based.
`proximal`	The number of bases on either side of TSS and 3'-end for the promoter and end region, respectively.
`distal`	The number of bases on either side for upstream/downstream, i.e. enhancer/silencer regions.

Details

The assumption made for introns is that there must be more than one exon, and that the introns are between the end of one exon and before the start of the next exon.

Value

All of these functions return a `RangedData` object with a `gene` column with the UCSC ID of the gene. For `transcripts_deprecated`, each element corresponds to a transcript, and there are columns for each type of region (promoter, threeprime, upstream, and downstream). For `exons_deprecated`, each element corresponds to an exon. For `introns_deprecated`, each element corresponds to an intron.

Author(s)

M. Lawrence.

`saveFeatures` *Methods to save and load the database contents for a Transcript Object.*

Description

These methods provide a way to dump a TranscriptDb object to an SQLite file, and to recreate that object the saved file.

Usage

```
saveFeatures(x, file)
loadFeatures(file)
```

Arguments

`x`	a transcripts object, which contains a connection to a DB.
`file`	A SQLite Database filename.

Value

For `loadFeatures` only, a TranscriptDb object is returned.

Author(s)

M. Carlson

See Also

TranscriptDb

Examples

```
txdb <-
  loadFeatures(system.file("extdata", "UCSC_knownGene_sample.sqlite",
                           package = "GenomicFeatures"))
txdb
```

`transcripts` *Extract genomic features from an object*

Description

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

Usage

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

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

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

Arguments

`x`	A TranscriptDb object.
`...`	Arguments to be passed to or from methods.
`vals`	Either `NULL` or a named list of vectors to be used to restrict the output. Valid names for this list are: `"gene_id"`, `"tx_id"`, `"tx_name"`, `"tx_chrom"`, `"tx_strand"`, `"exon_id"`, `"exon_name"`, `"exon_chrom"`, `"exon_strand"`, `"cds_id"`, `"cds_name"`, `"cds_chrom"`, `"cds_strand"` and `"exon_rank"`.
`columns`	Columns to include in the output. Must be `NULL` or a character vector with values in the above list of valid names. With the following restrictions: • `"tx_chrom"` and `"tx_strand"` are not allowed for `transcripts`. • `"exon_chrom"` and `"exon_strand"` are not allowed for `exons`. • `"cds_chrom"` and `"cds_strand"` are not allowed for `cds`.

Details

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

Value

a GRanges object

Author(s)

M. Carlson, P. Aboyoun and H. Pages

See Also

TranscriptDb, `id2name`, `transcriptsBy`, `transcriptsByOverlaps`

Examples

```
txdb <- loadFeatures(system.file("extdata", "UCSC_knownGene_sample.sqlite",
                                 package="GenomicFeatures"))
vals <- list(tx_chrom = c("chr3", "chr5"), tx_strand = "+")
transcripts(txdb, vals)
```

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

`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 TranscriptDb objects only.

Usage

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

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

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

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

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

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

Arguments

`x`	A TranscriptDb 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 `NA`s). A warning is issued when this happens. See `?id2name` for more information about feature internal ids and feature external names and how to map the formers to the latters.

Finally, `use.names=TRUE` cannot be used when grouping by gene `by="gene"`. This is because, unlike for the other features, the gene ids are external ids (e.g. Entrez Gene or Ensembl ids) so the db doesn't have a `"gene_name"` column for storing alternate gene names.

Details

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

When using `exonsBy` and `cdsBy` with `by = "tx"`, the ranges are returned in the order they appear in the transcript, i.e. order by the splicing.exon_rank field in `x`'s internal database. In all other cases, the ranges will be ordered by chromosome, strand, start, and end values.

Value

A GRangesList object.

Author(s)

M. Carlson, P. Aboyoun and H. Pages

See Also

TranscriptDb, `transcripts`, `id2name`, `transcriptsByOverlaps`

Examples

```
txdb_file <- system.file("extdata", "UCSC_knownGene_sample.sqlite",
                         package="GenomicFeatures")
txdb <- loadFeatures(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*

Description

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

Usage

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

Arguments

`x`	A TranscriptDb 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

TranscriptDb, `transcripts`

Examples

```
txdb <- loadFeatures(system.file("extdata", "UCSC_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)
```

Index