Package 'ensembldb'

April 14, 2017

Type Package

Title Utilities to create and use an Ensembl based annotation database

Version 1.6.2

Author Johannes Rainer <johannes.rainer@eurac.edu>,
Tim Triche <tim.triche@usc.edu>

Maintainer Johannes Rainer <johannes.rainer@eurac.edu>

URL https://github.com/jotsetung/ensembldb

BugReports https://github.com/jotsetung/ensembldb/issues

Imports methods, RSQLite, DBI, Biobase, GenomeInfoDb, AnnotationDbi (>= 1.31.19), rtracklayer, S4Vectors, AnnotationHub, Rsamtools, IRanges

Depends BiocGenerics (>= 0.15.10), GenomicRanges (>= 1.23.21), GenomicFeatures (>= 1.23.18)

Suggests BiocStyle, knitr, rmarkdown, EnsDb.Hsapiens.v75 (>= 0.99.7), RUnit, shiny, Gviz, BSgenome.Hsapiens.UCSC.hg19

Enhances RMySQL

VignetteBuilder knitr

Description The package provides functions to create and use transcript centric annotation databases/packages. The annotation for the databases are directly fetched from Ensembl using their Perl API. The functionality and data is similar to that of the TxDb packages from the GenomicFeatures package, but, in addition to retrieve all gene/transcript models and annotations from the database, the ensembldb package provides also a filter framework allowing to retrieve annotations for specific entries like genes encoded on a chromosome region or transcript models of lincRNA genes.

Collate Classes.R Generics.R functions-utils.R dbhelpers.R Methods.R Methods-Filter.R loadEnsDb.R makeEnsemblDbPackage.R EnsDbFromGTF.R runEnsDbApp.R select-methods.R seqname-utils.R zzz.R

biocViews Genetics, AnnotationData, Sequencing, Coverage

License LGPL

RoxygenNote 5.0.1

NeedsCompilation no

R topics documented:

`EnsDb` *Connect to an EnsDb object*

Description

The `EnsDb` constructor function connects to the database specified with argument `x` and returns a corresponding `EnsDb` object.

Usage

```
EnsDb(x)
```

Arguments

`x`	Either a character specifying the *SQLite* database file, or a `DBIConnection` to e.g. a MySQL database.

Details

By providing the connection to a MySQL database, it is possible to use MySQL as the database backend and queries will be performed on that database. Note however that this requires the package `RMySQL` to be installed. In addition, the user needs to have access to a MySQL server providing already an EnsDb database, or must have write privileges on a MySQL server, in which case the `useMySQL` method can be used to insert the annotations from an EnsDB package into a MySQL database.

Value

A `EnsDb` object.

Author(s)

Johannes Rainer

Examples

```
## "Standard" way to create an EnsDb object:
library(EnsDb.Hsapiens.v75)
EnsDb.Hsapiens.v75

## Alternatively, provide the full file name of a SQLite database file
dbfile <- system.file("extdata/EnsDb.Hsapiens.v75.sqlite", package = "EnsDb.Hsapiens.v75")
edb <- EnsDb(dbfile)
edb

## Third way: connect to a MySQL database
## Not run:
library(RMySQL)
dbcon <- dbConnect(MySQL(), user = my_user, pass = my_pass, host = my_host, dbname = "ensdb_hsapiens_v75")
edb <- EnsDb(dbcon)

## End(Not run)
```

EnsDb-class *Basic usage of an Ensembl based annotation database*

Description

Get some basic information from an Ensembl based annotation package generated with makeEnsembldbPackage.

Usage

```
## S4 method for signature 'EnsDb'
buildQuery(x, columns=c("gene_id", "gene_biotype",
                                    "gene_name"), filter=list(), order.by,
                              order.type="asc", skip.order.check=FALSE)

## S4 method for signature 'EnsDb'
dbconn(x)

## S4 method for signature 'EnsDb'
ensemblVersion(x)

## S4 method for signature 'EnsDb'
listColumns(x, table, skip.keys=TRUE, ...)

## S4 method for signature 'EnsDb'
listGenebiotypes(x, ...)

## S4 method for signature 'EnsDb'
listTxbiotypes(x, ...)

## S4 method for signature 'EnsDb'
listTables(x, ...)

## S4 method for signature 'EnsDb'
```

```
metadata(x, ...)

## S4 method for signature 'EnsDb'
organism(object)

## S4 method for signature 'EnsDb'
returnFilterColumns(x)

## S4 method for signature 'EnsDb'
returnFilterColumns(x)

## S4 replacement method for signature 'EnsDb'
returnFilterColumns(x) <- value

## S4 method for signature 'EnsDb'
seqinfo(x)

## S4 method for signature 'EnsDb'
seqlevels(x)

## S4 method for signature 'EnsDb'
updateEnsDb(x, ...)
```

Arguments

	(in alphabetic order)
	Additional arguments. Not used.
columns	Columns (attributes) to be retrieved from the database tables. Use the listColumns or listTables method for a list of supported columns.
filter	list of BasicFilter instance(s) to select specific entries from the database (see examples below).
object	For organism: an EnsDb instance.
order.by	name of one of the columns above on which the results should be sorted.
order.type	if the results should be ordered ascending (asc, default) or descending (desc).
skip.keys	for listColumns: whether primary and foreign keys (not being e.g. "gene_id" or alike) should be returned or not. By default these will not be returned.
skip.order.check	if paramter order.by should be checked for allowed column names. If TRUE the function checks if the provided order criteria orders on columns present in the database tables.
table	For listColumns: optionally specify the table name for which the columns should be returned.
value	For returnFilterColumns: a logical of length one specifying whether columns that are used for eventual filters should also be returned.
x	An EnsDb instance.

Value

For buildQuery A character string with the SQL query.

For `connection` The SQL connection to the RSQLite database.

For `EnsDb` An `EnsDb` instance.

For `lengthOf` A named integer vector with the length of the genes or transcripts.

For `listColumns` A character vector with the column names.

For `listGenebiotypes` A character vector with the biotypes of the genes in the database.

For `listTxbiotypes` A character vector with the biotypes of the transcripts in the database.

For `listTables` A list with the names corresponding to the database table names and the elements being the attribute (column) names of the table.

For `metadata` A `data.frame`.

For `organism` A character string.

For `returnFilterColumns` A logical of length 1.

For `seqinfo` A `Seqinfo` class.

For `updateEnsDb` A `EnsDb` object.

Objects from the Class

A connection to the respective annotation database is created upon loading of an annotation package created with the `makeEnsembldbPackage` function. In addition, the `EnsDb` constructor specifying the SQLite database file can be called to generate an instance of the object (see `makeEnsemblSQLiteFromTables` for an example).

Slots

ensdb Object of class `"DBIConnection"`: the connection to the database.

tables Named list of database table columns with the names being the database table names. The tables are ordered by their degree, i.e. the number of other tables they can be joined with.

.properties Internal list storing user-defined properties. Should not be directly accessed.

Methods and Functions

buildQuery Helper function building the SQL query to be used to retrieve the wanted information. Usually there is no need to call this method.

dbconn Returns the connection to the internal SQL database.

ensemblVersion Returns the Ensembl version on which the package was built.

listColumns Lists all columns of all tables in the database, or, if `table` is specified, of the respective table.

listGenebiotypes Lists all gene biotypes defined in the database.

listTxbiotypes Lists all transcript biotypes defined in the database.

listTables Returns a named list of database table columns (names of the list being the database table names).

metadata Returns a `data.frame` with the metadata information from the database, i.e. informations about the Ensembl version or Genome build the database was build upon.

organism Returns the organism name (e.g. `"homo_sapiens"`).

returnFilterColumns, returnFilterColumns<- Get or set the option which results in columns that are used for eventually specified filters to be added as result columns. The default value is `TRUE` (i.e. filter columns are returned).

seqinfo Returns the sequence/chromosome information from the database.

seqlevels Returns the chromosome/sequence names that are available in the database.

show Displays some informations from the database.

updateEnsDb Updates the `EnsDb` object to the most recent implementation.

Note

While a column named `"tx_name"` is listed by the `listTables` and `listColumns` method, no such column is present in the database. Transcript names returned by the methods are actually the transcript IDs. This *virtual* column was only introduced to be compliant with `TxDb` objects (which provide transcript names).

Author(s)

Johannes Rainer

See Also

`EnsDb`, `makeEnsembldbPackage`, `BasicFilter`, `exonsBy`, `genes`, `transcripts`, `makeEnsemblSQLiteFromTables`

Examples

```
library(EnsDb.Hsapiens.v75)

## Display some information:
EnsDb.Hsapiens.v75

## Show the tables along with its columns
listTables(EnsDb.Hsapiens.v75)

## For what species is this database?
organism(EnsDb.Hsapiens.v75)

## What Ensembl version if the database based on?
ensemblVersion(EnsDb.Hsapiens.v75)

## Get some more information from the database
metadata(EnsDb.Hsapiens.v75)

## Get all the sequence names.
seqlevels(EnsDb.Hsapiens.v75)

######    buildQuery
##
## Join tables gene and transcript and return gene_id and tx_id
buildQuery(EnsDb.Hsapiens.v75, columns=c("gene_id", "tx_id"))


## Get all exon_ids and transcript ids of genes encoded on chromosome Y.
buildQuery(EnsDb.Hsapiens.v75, columns=c("exon_id", "tx_id"),
           filter=list(SeqnameFilter( "Y")))

## List all available gene biotypes from the database:
listGenebiotypes(EnsDb.Hsapiens.v75)
```

```
## List all available transcript biotypes:
listTxbiotypes(EnsDb.Hsapiens.v75)

## Update the EnsDb; this is in most instances not necessary at all.
updateEnsDb(EnsDb.Hsapiens.v75)

######    returnFilterColumns
returnFilterColumns(EnsDb.Hsapiens.v75)

## Get protein coding genes on chromosome X, specifying to return
## only columns gene_name as additional column.
genes(EnsDb.Hsapiens.v75, filter=list(SeqnameFilter("X"),
                                      GenebiotypeFilter("protein_coding")),
      columns=c("gene_name"))
## By default we get also the gene_biotype column as the data was filtered
## on this column.

## This can be changed using the returnFilterColumns option
returnFilterColumns(EnsDb.Hsapiens.v75) <- FALSE
genes(EnsDb.Hsapiens.v75, filter=list(SeqnameFilter("X"),
                                      GenebiotypeFilter("protein_coding")),
      columns=c("gene_name"))
```

---

exonsBy *Retrieve annotation data from an Ensembl based package*

Description

Retrieve gene/transcript/exons annotations stored in an Ensembl based database package generated with the makeEnsembldbPackage function.

Usage

```
## S4 method for signature 'EnsDb'
exons(x, columns=listColumns(x,"exon"),
                        filter, order.by, order.type="asc",
                        return.type="GRanges")

## S4 method for signature 'EnsDb'
exonsBy(x, by=c("tx", "gene"),
                       columns=listColumns(x, "exon"), filter, use.names=FALSE)

## S4 method for signature 'EnsDb'
exonsByOverlaps(x, ranges, maxgap=0L, minoverlap=1L,
                                    type=c("any", "start", "end"),
                                    columns=listColumns(x, "exon"),
                                    filter)

## S4 method for signature 'EnsDb'
```

```
transcripts(x, columns=listColumns(x, "tx"),
                              filter, order.by, order.type="asc",
                              return.type="GRanges")

## S4 method for signature 'EnsDb'
transcriptsBy(x, by=c("gene", "exon"),
                                columns=listColumns(x, "tx"), filter)

## S4 method for signature 'EnsDb'
transcriptsByOverlaps(x, ranges, maxgap=0L, minoverlap=1L,
                                        type=c("any", "start", "end"),
                                        columns=listColumns(x, "tx"),
                                        filter)

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

## S4 method for signature 'EnsDb'
genes(x, columns=listColumns(x, "gene"), filter,
                        order.by, order.type="asc",
                        return.type="GRanges")

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

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

## S4 method for signature 'EnsDb'
fiveUTRsByTranscript(x, columns=NULL, filter)

## S4 method for signature 'EnsDb'
threeUTRsByTranscript(x, columns=NULL, filter)

## S4 method for signature 'GRangesList'
toSAF(x, ...)
```

Arguments

	(In alphabetic order)
	For `promoters`: additional arguments to be passed to the `transcripts` method.
`aggregateGenes`	For `disjointExons`: 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.
`by`	For `exonsBy`: wheter exons sould be fetched by genes or by transcripts; as in the corresponding function of the `GenomicFeatures` package. For `transcriptsBy`: whether transcripts should be fetched by genes or by exons; fetching transcripts by cds as supported by the `transcriptsBy` method in the `GenomicFeatures` package is currently not implemented. For `cdsBy`: whether cds should be fetched

by transcript of by gene.

`columns` Columns to be retrieved from the database tables.

Default values for `genes` are all columns from the `gene` database table, for `exons` and `exonsBy` the column names of the `exon` database table table and for `transcript` and `transcriptBy` the columns of the `tx` data base table (see details below for more information).

Note that any of the column names of the database tables can be submitted to any of the methods (use `listTables` or `listColumns` methods for a complete list of allowed column names).

For `cdsBy`: this argument is only supported for for `by="tx"`.

`downstream` For method `promoters`: the number of nucleotides downstream of the transcription start site that should be included in the promoter region.

`filter` A filter object extending `BasicFilter` or a list of such object(s) to select specific entries from the database (see examples below).

`includeTranscripts`

For `disjointExons`: When `TRUE` (default) a `tx_name` metadata column is included that lists all transcript IDs that overlap the exon fragment. Note: this is different to the `disjointExons` function in the `GenomicFeatures` package, that lists the transcript names, not IDs.

`maxgap` For `exonsByOverlaps` and `transcriptsByOverlaps`: see `exonsByOverlaps` help page in the `GenomicFeatures` package.

`minoverlap` For `exonsByOverlaps` and `transcriptsByOverlaps`: see `exonsByOverlaps` help page in the `GenomicFeatures` package.

`order.by` Name of one of the columns above on which the results should be sorted.

`order.type` If the results should be ordered ascending (`asc`, default) or descending (`desc`).

`ranges` For `exonsByOverlaps` and `transcriptsByOverlaps`: a `GRanges` object specifying the genomic regions.

`return.type` Type of the returned object. Can be either `"data.frame"`, `"DataFrame"` or `"GRanges"`. In the latter case the return object will be a `GRanges` object with the GRanges specifying the chromosomal start and end coordinates of the feature (gene, transcript or exon, depending whether `genes`, `transcripts` or `exons` was called). All additional columns are added as metadata columns to the GRanges object.

`type` For `exonsByOverlaps` and `transcriptsByOverlaps`: see `exonsByOverlaps` help page in the `GenomicFeatures` package.

`upstream` For method `promoters`: the number of nucleotides upstream of the transcription start site that should be included in the promoter region.

`use.names` For `cdsBy` and `exonsBy`: only for `by="gene"`: use the names of the genes instead of their IDs as names of the resulting `GRangesList`.

`x` For `toSAF` a `GRangesList` object. For all other methods an `EnsDb` instance.

Details

A detailed description of all database tables and the associated attributes/column names is also given in the vignette of this package. An overview of the columns is given below:

gene_id the Ensembl gene ID of the gene.

gene_name the name of the gene (in most cases its official symbol).

entrezid the NCBI Entrezgene ID of the gene; note that this can also be a ";" separated list of IDs for Ensembl genes mapped to more than one Entrezgene.

gene_biotype the biotype of the gene.

gene_seq_start the start coordinate of the gene on the sequence (usually a chromosome).

gene_seq_end the end coordinate of the gene.

seq_name the name of the sequence the gene is encoded (usually a chromosome).

seq_strand the strand on which the gene is encoded

seq_coord_system the coordinate system of the sequence.

tx_id the Ensembl transcript ID.

tx_biotype the biotype of the transcript.

tx_seq_start the chromosomal start coordinate of the transcript.

tx_seq_end the chromosomal end coordinate of the transcript.

tx_cds_seq_start the start coordinate of the coding region of the transcript (NULL for non-coding transcripts).

tx_cds_seq_end the end coordinate of the coding region.

exon_id the ID of the exon. In Ensembl, each exon specified by a unique chromosomal start and end position has its own ID. Thus, the same exon might be part of several transcripts.

exon_seq_start the chromosomal start coordinate of the exon.

exon_seq_end the chromosomal end coordinate of the exon.

exon_idx the index of the exon in the transcript model. As noted above, an exon can be part of several transcripts and thus its position inside these transcript might differ.

Also, the vignette provides examples on how to retrieve sequences for genes/transcripts/exons.

Value

For `exons`, `transcripts` and `genes`, a `data.frame`, `DataFrame` or a `GRanges`, depending on the value of the `return.type` parameter. The result is ordered as specified by the parameter `order.by` or, if not provided, by `seq_name` and chromosomal start coordinate, but NOT by any ordering of values in eventually submitted filter objects.

For `exonsBy`, `transcriptsBy`: a `GRangesList`, depending on the value of the `return.type` parameter. The results are ordered by the value of the by parameter.

For `exonsByOverlaps` and `transcriptsByOverlaps`: a `GRanges` with the exons or transcripts overlapping the specified regions.

For `toSAF`: a `data.frame` with column names `"GeneID"` (the group name from the `GRangesList`, i.e. the ID by which the `GRanges` are split), `"Chr"` (the seqnames from the `GRanges`), `"Start"` (the start coordinate), `"End"` (the end coordinate) and `"Strand"` (the strand).

For `disjointExons`: a `GRanges` of non-overlapping exon parts.

For `cdsBy`: a `GRangesList` with `GRanges` per either transcript or exon specifying the start and end coordinates of the coding region of the transcript or gene.

For `fiveUTRsByTranscript`: a `GRangesList` with `GRanges` for each protein coding transcript representing the start and end coordinates of full or partial exons that constitute the 5' untranslated region of the transcript.

For `threeUTRsByTranscript`: a `GRangesList` with `GRanges` for each protein coding transcript representing the start and end coordinates of full or partial exons that constitute the 3' untranslated region of the transcript.

Methods and Functions

exons Retrieve exon information from the database. Additional columns from transcripts or genes associated with the exons can be specified and are added to the respective exon annotation.

exonsBy Retrieve exons grouped by transcript or by gene. This function returns a `GRangesList` as does the analogous function in the `GenomicFeatures` package. Using the `columns` parameter it is possible to determine which additional values should be retrieved from the database. These will be included in the `GRanges` object for the exons as metadata columns. The exons in the inner `GRanges` are ordered by the exon index within the transcript (if `by="tx"`), or increasingly by the chromosomal start position of the exon or decreasingly by the chromosomal end position of the exon depending whether the gene is encoded on the + or - strand (for `by="gene"`). The `GRanges` in the `GRangesList` will be ordered by the name of the gene or transcript.

exonsByOverlaps Retrieve exons overlapping specified genomic ranges. For more information see `exonsByOverlaps` method in the `GenomicFeatures` package. The functionality is to some extent similar and redundant to the `exons` method in combination with `GRangesFilter` filter.

transcripts Retrieve transcript information from the database. Additional columns from genes or exons associated with the transcripts can be specified and are added to the respective transcript annotation.

transcriptsBy Retrieve transcripts grouped by gene or exon. This function returns a `GRangesList` as does the analogous function in the `GenomicFeatures` package. Using the `columns` parameter it is possible to determine which additional values should be retrieved from the database. These will be included in the `GRanges` object for the transcripts as metadata columns. The transcripts in the inner `GRanges` are ordered increasingly by the chromosomal start position of the transcript for genes encoded on the + strand and in a decreasing manner by the chromosomal end position of the transcript for genes encoded on the - strand. The `GRanges` in the `GRangesList` will be ordered by the name of the gene or exon.

transcriptsByOverlaps Retrieve transcripts overlapping specified genomic ranges. For more information see `transcriptsByOverlaps` method in the `GenomicFeatures` package. The functionality is to some extent similar and redundant to the `transcripts` method in combination with `GRangesFilter` filter.

promoters Retrieve promoter information from the database. Additional columns from genes or exons associated with the promoters can be specified and are added to the respective promoter annotation.

genes Retrieve gene information from the database. Additional columns from transcripts or exons associated with the genes can be specified and are added to the respective gene annotation.

disjointExons This method is identical to `disjointExons` defined in the `GenomicFeatures` package. It creates a `GRanges` of non-overlapping exon parts with metadata columns of `gene_id` and `exonic_part`. Exon parts that overlap more than one gene can be dropped with `aggregateGenes=FALSE`.

cdsBy Returns the coding region grouped either by transcript or by gene. Each element in the `GRangesList` represents the cds for one transcript or gene, with the individual ranges corresponding to the coding part of its exons. For `by="tx"` additional annotation columns can be added to the individual `GRanges` (in addition to the default columns `exon_id` and `exon_rank`). Note that the `GRangesList` is sorted by its names.

fiveUTRsByTranscript Returns the 5' untranslated region for protein coding transcripts.

threeUTRsByTranscript Returns the 3' untranslated region for protein coding transcripts.

toSAF Reformats a `GRangesList` object into a `data.frame` corresponding to a standard SAF (Simplified Annotation Format) file (i.e. with column names `"GeneID"`, `"Chr"`, `"Start"`, `"End"` and `"Strand"`). Note: this method makes only sense on a `GRangesList` that groups features (exons, transcripts) by gene.

Note

Ensembl defines genes not only on standard chromosomes, but also on patched chromosomes and chromosome variants. Thus it might be advisable to restrict the queries to just those chromosomes of interest (e.g. by specifying a `SeqnameFilter(c(1:22, "X", "Y"))`). In addition, also so called LRG genes (Locus Reference Genomic) are defined in Ensembl. Their gene id starts with LRG instead of ENS for Ensembl genes, thus, a filter can be applied to specifically select those genes or exclude those genes (see examples below).

Depending on the value of the global option `"ucscChromosomeNames"` (use `getOption(ucscChromosomeNames, FALSE` to get its value or `option(ucscChromosomeNames=TRUE)` to change its value) the sequence/chromosome names of the returned `GRanges` objects or provided in the returned `data.frame` or `DataFrame` correspond to Ensembl chromosome names (if value is `FALSE`) or UCSC chromosome names (if `TRUE`). This ensures a better integration with the `Gviz` package, in which this option is set by default to `TRUE`.

Note

While it is possible to request values from a column `"tx_name"` (with the `columns` argument), no such column is present in the database. The returned values correspond to the ID of the transcripts.

Author(s)

Johannes Rainer, Tim Triche

See Also

makeEnsembldbPackage, BasicFilter, listColumns, lengthOf

Examples

```
library(EnsDb.Hsapiens.v75)
edb <- EnsDb.Hsapiens.v75

######   genes
##
## get all genes endcoded on chromosome Y
AllY <- genes(edb, filter=SeqnameFilter("Y"))
AllY

## return result as DataFrame.
AllY.granges <- genes(edb,
                      filter=SeqnameFilter("Y"),
                      return.type="DataFrame")
AllY.granges

## include all transcripts of the gene and their chromosomal
## coordinates, sort by chrom start of transcripts and return as
## GRanges.
AllY.granges.tx <- genes(edb,
                         filter=SeqnameFilter("Y"),
                         columns=c("gene_id", "seq_name",
                             "seq_strand", "tx_id", "tx_biotype",
                             "tx_seq_start", "tx_seq_end"),
                         order.by="tx_seq_start")
AllY.granges.tx
```

```
######   transcripts
##
## get all transcripts of a gene
Tx <- transcripts(edb,
                  filter=GeneidFilter("ENSG00000184895"),
                  order.by="tx_seq_start")
Tx

## get all transcripts of two genes along with some information on the
## gene and transcript
Tx <- transcripts(edb,
                  filter=GeneidFilter(c("ENSG00000184895",
                      "ENSG00000092377")),
                      columns=c("gene_id", "gene_seq_start",
                          "gene_seq_end", "gene_biotype", "tx_biotype"))
Tx

######   promoters
##
## get the bona-fide promoters (2k up- to 200nt downstream of TSS)
promoters(edb, filter=GeneidFilter(c("ENSG00000184895",
                                     "ENSG00000092377")))

######   exons
##
## get all exons of the provided genes
Exon <- exons(edb,
              filter=GeneidFilter(c("ENSG00000184895",
                  "ENSG00000092377")),
              order.by="exon_seq_start",
              columns=c( "gene_id", "gene_seq_start",
                  "gene_seq_end", "gene_biotype"))
Exon



#####    exonsBy
##
## get all exons for transcripts encoded on chromosomes X and Y.
ETx <- exonsBy(edb, by="tx",
               filter=SeqnameFilter(c("X", "Y")))
ETx
## get all exons for genes encoded on chromosome 1 to 22, X and Y and
## include additional annotation columns in the result
EGenes <- exonsBy(edb, by="gene",
                  filter=SeqnameFilter(c("X", "Y")),
                  columns=c("gene_biotype", "gene_name"))
EGenes

## Note that this might also contain "LRG" genes.
length(grep(names(EGenes), pattern="LRG"))

## to fetch just Ensemblgenes, use an GeneidFilter with value
## "ENS%" and condition "like"
```

```
#####    transcriptsBy
##
TGenes <- transcriptsBy(edb, by="gene",
                        filter=SeqnameFilter(c("X", "Y")))
TGenes

## convert this to a SAF formatted data.frame that can be used by the
## featureCounts function from the Rsubreader package.
head(toSAF(TGenes))


#####   transcriptsByOverlaps
##
ir <- IRanges(start=c(2654890, 2709520, 28111770),
              end=c(2654900, 2709550, 28111790))
gr <- GRanges(rep("Y", length(ir)), ir)

## Retrieve all transcripts overlapping any of the regions.
txs <- transcriptsByOverlaps(edb, gr)
txs

## Alternatively, use a GRangesFilter
grf <- GRangesFilter(gr, condition="overlapping")
txs <- transcripts(edb, filter=grf)
txs


####    cdsBy
## Get the coding region for all transcripts on chromosome Y.
## Specifying also additional annotation columns (in addition to the default
## exon_id and exon_rank).
cds <- cdsBy(edb, by="tx", filter=SeqnameFilter("Y"),
             columns=c("tx_biotype", "gene_name"))

####    the 5' untranslated regions:
fUTRs <- fiveUTRsByTranscript(edb, filter=SeqnameFilter("Y"))

####    the 3' untranslated regions with additional column gene_name.
tUTRs <- threeUTRsByTranscript(edb, filter=SeqnameFilter("Y"),
                               columns="gene_name")
```

`GeneidFilter-class` *Filter results fetched from the Ensembl database*

Description

These classes allow to specify which entries (i.e. genes, transcripts or exons) should be retrieved from the database.

Details

`ExonidFilter` Allows to filter based on the (Ensembl) exon identifier.

`ExonrankFilter` Allows to filter based on the rank (index) of the exon within the transcript model. Exons are always numbered 5' to 3' end of the transcript, thus, also on the reverse strand, the exon 1 is the most 5' exon of the transcript.

`EntrezidFilter` Filter results based on the NCBI Entrezgene identifierts of the genes. Use the `listGenebiotypes` method to get a complete list of all available gene biotypes.

`GenebiotypeFilter` Filter results based on the gene biotype as defined in the Ensembl database.

`GeneidFilter` Filter results based on the Ensembl gene identifiers.

`GenenameFilter` Allows to filter on the gene names (symbols) of the genes.

`SymbolFilter` Filter on gene symbols. Note that since no such database column is available in an `EnsDb` database the gene names are used to filter. These do however correspond all to the official gene symbols.

`GRangesFilter` Allows to fetch features within or overlapping specified genomic region(s)/range(s). This filter takes a `GRanges` object as input and, if `condition="within"` (the default) will restrict results to features (genes, transcripts or exons) that are completely within the region. Alternatively, by specifying `condition="overlapping"` it will return all features (i.e. genes for a call to `genes`, transcripts for a call to `transcripts` and exons for a call to `exons`) that are partially overlapping with the region, i.e. which start coordinate is smaller than the end coordinate of the region and which end coordinate is larger than the start coordinate of the region. Thus, genes and transcripts that have an intron overlapping the region will also be returned.

Calls to the methods `exonsBy`, `cdsBy` and `transcriptsBy` use the start and end coordinates of the feature type specified with argument `by` (i.e. `"gene"`, `"transcript"` or `"exon"`) for the filtering.

Note: if the specified `GRanges` object defines multiple region, all features within (or overlapping) any of these regions are returned.

Chromosome names/seqnames can be provided in UCSC format (e.g. `"chrX"`) or Ensembl format (e.g. `"X"`); see `seqlevelsStyle` for more information.

`SeqendFilter` Filter based on the chromosomal end coordinate of the exons, transcripts or genes.

`SeqnameFilter` Filter on the sequence name on which the features are encoded (mostly the chromosome names). Supports UCSC chromosome names (e.g. `"chrX"`) and Ensembl chromosome names (e.g. `"X"`).

`SeqstartFilter` Filter based on the chromosomal start coordinates of the exons, transcripts or genes.

`SeqstrandFilter` Filter based on the strand on which the features are encoded.

`TxbiotypeFilter` Filter on the transcript biotype defined in Ensembl. Use the `listTxbiotypes` method to get a complete list of all available transcript biotypes.

`TxidFilter` Filter on the Ensembl transcript identifiers.

Objects from the Class

While objects can be created by calls e.g. of the form `new("GeneidFilter", ...)` users are strongly encouraged to use the specific functions: `GeneidFilter`, `EntrezidFilter`, `GenenameFilter`, `GenebiotypeFilter`, `GRangesFilter`, `SymbolFilter`, `TxidFilter`, `TxbiotypeFilter`, `ExonidFilter`, `ExonrankFilter`, `SeqnameFilter`, `SeqstrandFilter`, `SeqstartFilter` and `SeqendFilter`.

See examples below for usage.

Slots

`condition`**:** Object of class `"character"`: can be either `"="`, `"in"` or `"like"` to filter on character values (e.g. gene id, gene biotype, seqname etc), or `"="`, `">"` or `"<"` for numerical values (chromosome/seq coordinates). Note that for `"like"` `value` should be a SQL pattern (e.g. `"ENS%"`).

`value`**:** Object of class `"character"`: the value to be used for filtering.

Extends

Class `BasicFilter`, directly.

Methods for all `BasicFilter` objects

Note: these methods are applicable to all classes extending the `BasicFilter` class.

`signature(object = "GeneidFilter", db = "EnsDb",with.tables = "character")`: returns the column (attribute name) to be used for the filtering. Submitting the `db` parameter ensures that returned column is valid in the corresponding database schema. The optional argument `with.tables` allows to specify which in which database table the function should look for the attribute/column name. By default the method will check all database tables.

colu~~mn~~column `signature(object = "GeneidFilter", db = "EnsDb",with.tables = "missing")`: returns the column (attribute name) to be used for the filtering. Submitting the `db` parameter ensures that returned column is valid in the corresponding database schema.

column `signature(object = "GeneidFilter", db = "missing",with.tables = "missing")`: returns the column (table column name) to be used for the filtering.

condition `signature(x = "BasicFilter")`: returns the value for the `condition` slot.

condition<- setter method for condition.

value `signature(x = "BasicFilter", db = "EnsDb")`: returns the value of the `value` slot of the filter object.

value<- setter method for value.

where `signature(object = "GeneidFilter", db = "EnsDb",with.tables = "character")`: returns the where condition for the SQL call. Submitting also the `db` parameter ensures that the columns are valid in the corresponding database schema. The optional argument `with.tables` allows to specify which in which database table the function should look for the attribute/column name. By default the method will check all database tables.

where `signature(object = "GeneidFilter", db = "EnsDb",with.tables = "missing")`: returns the where condition for the SQL call. Submitting also the `db` parameter ensures that the columns are valid in the corresponding database schema.

where `signature(object = "GeneidFilter", db = "missing",with.tables = "missing")`: returns the where condition for the SQL call.

Methods for `GRangesFilter` objects

start, end, strand Get the start and end coordinate and the strand from the `GRanges` within the filter.

seqlevels, seqnames Get the names of the sequences from the `GRanges` of the filter.

Note

The column and where methods should be always called along with the EnsDb object, as this ensures that the returned column names are valid for the database schema. The optional argument with.tables should on the other hand only be used rarely as it is more intended for internal use.

Note that the database column "entrezid" queried for EntrezidFilter classes can contain multiple, ";" separated, Entrezgene IDs, thus, using this filter at present might not return all entries from the database. Also, the database does not provide a column with the official gene symbols and a SymbolFilter queries the gene names instead.

Author(s)

Johannes Rainer

See Also

genes, transcripts, exons, listGenebiotypes, listTxbiotypes

Examples

```
## create a filter that could be used to retrieve all informations for
## the respective gene.
Gif <- GeneidFilter("ENSG00000012817")
Gif
## returns the where condition of the SQL querys
where(Gif)

## create a filter for a chromosomal end position of a gene
Sef <- SeqendFilter(10000, condition=">", "gene")
Sef

## for additional examples see the help page of "genes"


## Example for GRangesFilter:
## retrieve all genes overlapping the specified region
grf <- GRangesFilter(GRanges("11", ranges=IRanges(114000000, 114000050),
                             strand="+"), condition="overlapping")
library(EnsDb.Hsapiens.v75)
edb <- EnsDb.Hsapiens.v75
genes(edb, filter=grf)

## Get also all transcripts overlapping that region
transcripts(edb, filter=grf)

## Retrieve all transcripts for the above gene
gn <- genes(edb, filter=grf)
txs <- transcripts(edb, filter=GenenameFilter(gn$gene_name))
## Next we simply plot their start and end coordinates.
plot(3, 3, pch=NA, xlim=c(start(gn), end(gn)), ylim=c(0, length(txs)), yaxt="n", ylab="")
## Highlight the GRangesFilter region
rect(xleft=start(grf), xright=end(grf), ybottom=0, ytop=length(txs), col="red", border="red")
for(i in 1:length(txs)){
    current <- txs[i]
   rect(xleft=start(current), xright=end(current), ybottom=i-0.975, ytop=i-0.125, border="grey")
```

```
    text(start(current), y=i-0.5,pos=4, cex=0.75, labels=current$tx_id)
}
## Thus, we can see that only 4 transcripts of that gene are indeed overlapping the region.


## No exon is overlapping that region, thus we're not getting anything
exons(edb, filter=grf)


## Example for ExonrankFilter
## Extract all exons 1 and (if present) 2 for all genes encoded on the
## Y chromosome
exons(edb, columns=c("tx_id", "exon_idx"),
      filter=list(SeqnameFilter("Y"),
                  ExonrankFilter(3, condition="<")))


## Get all transcripts for the gene SKA2
transcripts(edb, filter=GenenameFilter("SKA2"))

## Which is the same as using a SymbolFilter
transcripts(edb, filter=SymbolFilter("SKA2"))
```

`getGeneRegionTrackForGviz` *Utility functions*

Description

Utility functions integrating `EnsDb` objects with other Bioconductor packages.

Usage

```
## S4 method for signature 'EnsDb'
getGeneRegionTrackForGviz(x, filter=list(),
                                                chromosome=NULL,
                                                start=NULL, end=NULL,
                                                featureIs="gene_biotype")
```

Arguments

	(In alphabetic order)
	For `getGeneRegionTrackForGviz`: optional chromosome name to restrict the returned entry to a specific chromosome.
ehdomosome	For `getGeneRegionTrackForGviz`: optional chromosomal end coordinate specifying, together with `start`, the chromosomal region from which features should be retrieved.
`featureIs`	For `getGeneRegionTrackForGviz`: whether the gene (`"gene_biotype"`) or the transcript biotype (`"tx_biotype"`) should be returned in column `"feature"`.

filter	A filter object extending BasicFilter or a list of such object(s) to select specific entries from the database (see examples below).
start	For getGeneRegionTrackForGviz: optional chromosomal start coordinate specifying, together with end, the chromosomal region from which features should be retrieved.
x	For toSAF a GRangesList object. For all other methods an EnsDb instance.

Value

For getGeneRegionTrackForGviz: see method description above.

Methods and Functions

getGeneRegionTrackForGviz Retrieve a GRanges object with transcript features from the EnsDb that can be used directly in the Gviz package to create a GeneRegionTrack. Using the filter, chromosome, start and end arguments it is possible to fetch specific features (e.g. lincRNAs) from the database.

If chromosome, start and end is provided the function internally first retrieves all transcripts that have an exon or an intron in the specified chromosomal region and subsequently fetch all of these transcripts. This ensures that all transcripts of the region are returned, even those that have *only* an intron in the region.

The function returns a GRanges object with additional annotation columns "feature", "gene", "exon", "exon_rank", "trancript", "symbol" specifying the feature type (either gene or transcript biotype), the (Ensembl) gene ID, the exon ID, the rank/index of the exon in the transcript, the transcript ID and the gene symbol/name.

Author(s)

Johannes Rainer

See Also

BasicFilter transcripts

Examples

```
library(EnsDb.Hsapiens.v75)
edb <- EnsDb.Hsapiens.v75
######   getGeneRegionTrackForGviz
##
## Get all genes encoded on chromosome Y in the specifyed region.
AllY <- getGeneRegionTrackForGviz(edb, chromosome="Y", start=5000000,
                                  end=7000000)
## We could plot this now using plotTracks(GeneRegionTrack(AllY))

## We can also use filters to further restrict the query to e.g.
## all lincRNA genes encoded in that region.
lincsY <- getGeneRegionTrackForGviz(edb, chromosome="Y", start=5000000,
                                    end=7000000,
                                    filter=GenebiotypeFilter("lincRNA"))
```

getGenomeFaFile *Functionality related to DNA/RNA sequences*

Description

Utility functions related to RNA/DNA sequences, such as extracting RNA/DNA sequences for features defined in Ensb.

Usage

```
## S4 method for signature 'EnsDb'
getGenomeFaFile(x, pattern="dna.toplevel.fa")
```

Arguments

	(In alphabetic order)
	For method getGenomeFaFile: the pattern to be used to identify the fasta file representing genomic DNA sequence.
pattern	For all other methods an EnsDb instance.

Value

For getGenomeFaFile: a FaFile-class object with the genomic DNA sequence.

Methods and Functions

getGenomeFaFile Returns a FaFile-class (defined in Rsamtools) with the genomic sequence of the genome build matching the Ensembl version of the EnsDb object. The file is retrieved using the AnnotationHub package, thus, at least for the first invocation, an internet connection is required to locate and download the file; subsequent calls will load the cached file instead. If no fasta file for the actual Ensembl version is available the function tries to identify a file matchint the species and genome build version of the closest Ensembl release and returns that instead. See the vignette for an example to work with such files.

Author(s)

Johannes Rainer

See Also

BasicFilter transcripts exonsBy

Examples

```
## Loading an EnsDb for Ensembl version 75 (genome GRCh37):
library(EnsDb.Hsapiens.v75)
edb <- EnsDb.Hsapiens.v75

## Not run:
    ## Retrieve a FaFile with the gneomic DNA sequence matching the organism,
    ## genome release version and, if possible, the Ensembl version of the
    ## EnsDb object.
    Dna <- getGenomeFaFile(edb)
    ## Extract the transcript sequence for all transcripts encoded on chromosome
    ## Y.
    ##extractTranscriptSeqs(Dna, edb, filter=SeqnameFilter("Y"))


## End(Not run)
```

`lengthOf` *Calculating lengths of features*

Description

These methods allow to calculate the lengths of features (transcripts, genes, CDS, 3' or 5' UTRs) defined in an `EnsDb` object or database.

Usage

```
## S4 method for signature 'EnsDb'
lengthOf(x, of="gene", filter=list())
```

Arguments

	(In alphabetic order)
	list of `BasicFilter` instance(s) to select specific entries from the database (see examples below).
`of`, `filter`	for `lengthOf`: whether the length of genes or transcripts should be retrieved from the database.
`x`	For `lengthOf`: either an `EnsDb` or a `GRangesList` object. For all other methods an `EnsDb` instance.

Value

For `lengthOf`: see method description above.

Methods and Functions

lengthOf Retrieve the length of genes or transcripts from the database. The length is the sum of the lengths of all exons of a transcript or a gene. In the latter case the exons are first reduced so that the length corresponds to the part of the genomic sequence covered by the exons.

Note: in addition to this method, also the `transcriptLengths` function in the `GenomicFeatures` package can be used.

Author(s)

Johannes Rainer

See Also

`exonsBy` `transcripts` `transcriptLengths`

Examples

```
library(EnsDb.Hsapiens.v75)
edb <- EnsDb.Hsapiens.v75

#####    lengthOf
##
## length of a specific gene.
lengthOf(edb,
         filter=list(GeneidFilter("ENSG00000000003")))

## length of a transcript
lengthOf(edb, of="tx",
         filter=list(TxidFilter("ENST00000494424")))

## average length of all protein coding genes encoded on chromosomes X
## and Y
mean(lengthOf(edb, of="gene",
              filter=list(GenebiotypeFilter("protein_coding"),
                  SeqnameFilter(c("X", "Y")))))

## average length of all snoRNAs
mean(lengthOf(edb, of="gene",
              filter=list(GenebiotypeFilter("snoRNA"),
                  SeqnameFilter(c("X", "Y")))))

##### transcriptLengths
##
## Calculate the length of transcripts encoded on chromosome Y, including
## length of the CDS, 5' and 3' UTR.
##len <- transcriptLengths(edb, with.cds_len=TRUE, with.utr5_len=TRUE,
##                         with.utr3_len=TRUE, filter=SeqnameFilter("Y"))
##head(len)
```

`listEnsDbs` *List EnsDb databases in a MySQL server*

Description

The `listEnsDbs` function lists EnsDb databases in a MySQL server.

Usage

```
listEnsDbs(dbcon, host, port, user, pass)
```

Arguments

`dbcon`	A `DBIConnection` object providing access to a MySQL database. Either `dbcon` or all of the other arguments have to be specified.
`host`	Character specifying the host on which the MySQL server is running.
`port`	The port of the MySQL server (usually `3306`).
`user`	The username for the MySQL server.
`pass`	The password for the MySQL server.

Details

The use of this function requires that the `RMySQL` package is installed and that the user has either access to a MySQL server with already installed EnsDb databases, or write access to a MySQL server in which case EnsDb databases could be added with the `useMySQL` method. EnsDb databases follow the same naming conventions than the EnsDb packages, with the exception that the name is all lower case and that `"."` is replaced by `"_"`.

Value

A `data.frame` listing the database names, organism name and Ensembl version of the EnsDb databases found on the server.

Author(s)

Johannes Rainer

See Also

`useMySQL`

Examples

```
## Not run:
library(RMySQL)
dbcon <- dbConnect(MySQL(), host = "localhost", user = my_user, pass = my_pass)
listEnsDbs(dbcon)

## End(Not run)
```

makeEnsembldbPackage *Generating a Ensembl annotation package from Ensembl*

Description

The functions described on this page allow to build EnsDb annotation objects/databases from Ensembl annotations. The most complete set of annotations, which include also the NCBI Entrezgene identifiers for each gene, can be retrieved by the functions using the Ensembl Perl API (i.e. functions fetchTablesFromEnsembl, makeEnsemblSQLiteFromTables). Alternatively the functions ensDbFromAH, ensDbFromGRanges, ensDbFromGff and ensDbFromGtf can be used to build EnsDb objects using GFF or GTF files from Ensembl, which can be either manually downloaded from the Ensembl ftp server, or directly form within R using AnnotationHub. The generated SQLite database can be packaged into an R package using the makeEnsembldbPackage.

Usage

```
ensDbFromAH(ah, outfile, path, organism, genomeVersion, version)

ensDbFromGRanges(x, outfile, path, organism, genomeVersion,
                 version)

ensDbFromGff(gff, outfile, path, organism, genomeVersion,
             version)

ensDbFromGtf(gtf, outfile, path, organism, genomeVersion,
             version)

fetchTablesFromEnsembl(version, ensemblapi, user="anonymous",
                       host="ensembldb.ensembl.org", pass="",
                       port=5306, species="human")

makeEnsemblSQLiteFromTables(path=".", dbname)

makeEnsembldbPackage(ensdb, version, maintainer, author,
                     destDir=".", license="Artistic-2.0")
```

Arguments

	(in alphabetical order)
	For ensDbFromAH: an AnnotationHub object representing a single resource (i.e. GTF file from Ensembl) from AnnotationHub.
author	The author of the package.
dbname	The name for the database (optional). By default a name based on the species and Ensembl version will be automatically generated (and returned by the function).
destDir	Where the package should be saved to.
ensdb	The file name of the SQLite database generated by makeEnsemblSQLiteFromTables.

ensemblapi	The path to the Ensembl perl API installed locally on the system. The Ensembl perl API version has to fit the version.
genomeVersion	For ensDbFromAH, ensDbFromGtf and ensDbFromGff: the version of the genome (e.g. "GRCh37"). If not provided the function will try to guess it from the file name (assuming file name convention of Ensembl GTF files).
gff	The GFF file to import.
gtf	The GTF file name.
host	The hostname to access the Ensembl database.
license	The license of the package.
maintainer	The maintainer of the package.
organism	For ensDbFromAH, ensDbFromGff and ensDbFromGtf: the organism name (e.g. "Homo_sapiens"). If not provided the function will try to guess it from the file name (assuming file name convention of Ensembl GTF files).
outfile	The desired file name of the SQLite file. If not provided the name of the GTF file will be used.
pass	The password for the Ensembl database.
path	The directory in which the tables retrieved by fetchTablesFromEnsembl or the SQLite database file generated by ensDbFromGtf are stored.
port	The port to be used to connect to the Ensembl database.
species	The species for which the annotations should be retrieved.
user	The username for the Ensembl database.
version	For fetchTablesFromEnsembl, ensDbFromGRanges and ensDbFromGtf: the Ensembl version for which the annotation should be retrieved (e.g. 75). The ensDbFromGtf function will try to guess the Ensembl version from the GTF file name if not provided. For makeEnsemblDbPackage: the version for the package.
x	For ensDbFromGRanges: the GRanges object.

Details

The fetchTablesFromEnsembl function internally calls the perl script get_gene_transcript_exon_tables.pl to retrieve all required information from the Ensembl database using the Ensembl perl API.

As an alternative way, a EnsDb database file can be generated by the ensDbFromGtf or ensDbFromGff from a GTF or GFF file downloaded from the Ensembl ftp server or using the ensDbFromAH to build a database directly from corresponding resources from the AnnotationHub. The returned database file name can then be used as an input to the makeEnsembldbPackage or it can be directly loaded and used by the EnsDb constructor.

Value

makeEnsemblSQLiteFromTables, ensDbFromAH, ensDbFromGRanges and ensDbFromGtf: the name of the SQLite file.

Functions

ensDbFromAH Create an EnsDb (SQLite) database from a GTF file provided by AnnotationHub. The function returns the file name of the generated database file. For usage see the examples below.

ensDbFromGff Create an EnsDb (SQLite) database from a GFF file from Ensembl. The function returns the file name of the generated database file. For usage see the examples below.

ensDbFromGtf Create an EnsDb (SQLite) database from a GTF file from Ensembl. The function returns the file name of the generated database file. For usage see the examplesbelow.

ensDbFromGRanges Create an EnsDb (SQLite) database from a GRanges object (e.g. from AnnotationHub). The function returns the file name of the generated database file. For usage see the examples below.

fetchTablesFromEnsembl Uses the Ensembl Perl API to fetch all required data from an Ensembl database server and stores them locally to text files (that can be used as input for the makeEnsembldbSQLiteFromTables function).

makeEnsemblSQLiteFromTables Creates the SQLite EnsDb database from the tables generated by the fetchTablesFromEnsembl.

makeEnsembldbPackage Creates an R package containing the EnsDb database from a EnsDb SQLite database created by any of the above functions ensDbFromAH, ensDbFromGff, ensDbFromGtf or makeEnsemblSQLiteFromTables.

Note

A local installation of the Ensembl perl API is required for the fetchTablesFromEnsembl. See http://www.ensembl.org/info/docs/api/api_installation.html for installation inscructions.

A database generated from a GTF/GFF files lacks some features as they are not available in the GTF files from Ensembl. These are: NCBI Entrezgene IDs.

Author(s)

Johannes Rainer

See Also

EnsDb, genes

Examples

```
## Not run:

    ## get all human gene/transcript/exon annotations from Ensembl (75)
    ## the resulting tables will be stored by default to the current working
    ## directory; if the correct Ensembl api (version 75) is defined in the
    ## PERL5LIB environment variable, the ensemblapi parameter can also be omitted.
    fetchTablesFromEnsembl(75,
                           ensemblapi="/home/bioinfo/ensembl/75/API/ensembl/modules",
                           species="human")

    ## These tables can then be processed to generate a SQLite database
    ## containing the annotations
    DBFile <- makeEnsemblSQLiteFromTables()

    ## and finally we can generate the package
    makeEnsembldbPackage(ensdb=DBFile, version="0.0.1",
                         maintainer="Johannes Rainer <johannes.rainer@eurac.edu>",
                         author="J Rainer")
```

```
    ## Build an annotation database form a GFF file from Ensembl.
    ## ftp://ftp.ensembl.org/pub/release-83/gff3/rattus_norvegicus
    gff <- "Rattus_norvegicus.Rnor_6.0.83.gff3.gz"
    DB <- ensDbFromGff(gff=gff)
    edb <- EnsDb(DB)
    edb

    ## Build an annotation file from a GTF file.
    ## the GTF file can be downloaded from
    ## ftp://ftp.ensembl.org/pub/release-75/gtf/homo_sapiens/
    gtffile <- "Homo_sapiens.GRCh37.75.gtf.gz"
    ## generate the SQLite database file
    DB <- ensDbFromGtf(gtf=paste0(ensemblhost, gtffile))

    ## load the DB file directly
    EDB <- EnsDb(DB)

    ## Alternatively, we could fetch a GTF file directly from AnnotationHub
    ## and build the database from that:
    library(AnnotationHub)
    ah <- AnnotationHub()
    ## Query for all GTF files from Ensembl for Ensembl version 81
    query(ah, c("Ensembl", "release-81", "GTF"))
    ## We could get the one from e.g. Bos taurus:
    DB <- ensDbFromAH(ah["AH47941"])
    edb <- EnsDb(DB)
    edb

## End(Not run)

## Generate a sqlite database for genes encoded on chromosome Y
chrY <- system.file("chrY", package="ensembldb")
DBFile <- makeEnsemblSQLiteFromTables(path=chrY ,dbname=tempfile())
## load this database:
edb <- EnsDb(DBFile)

edb

## Generate a sqlite database from a GRanges object specifying
## genes encoded on chromosome Y
load(system.file("YGRanges.RData", package="ensembldb"))

Y

DB <- ensDbFromGRanges(Y, path=tempdir(), version=75,
                       organism="Homo_sapiens")
edb <- EnsDb(DB)
```

---

runEnsDbApp *Search annotations interactively*

Description

This function starts the interactive EnsDb shiny web application that allows to look up gene/transcript/exon annotations from an EnsDb annotation package installed locally.

Usage

```
runEnsDbApp(...)
```

Arguments

...	Additional arguments passed to the runApp function from the shiny package.

Details

The shiny based web application allows to look up any annotation available in any of the locally installed EnsDb annotation packages.

Value

If the button *Return & close* is clicked, the function returns the results of the present query either as data.frame or as GRanges object.

Author(s)

Johannes Rainer

See Also

EnsDb, genes

---

select *Integration into the AnnotationDbi framework*

Description

Several of the methods available for AnnotationDbi objects are also implemented for EnsDb objects. This enables to extract data from EnsDb objects in a similar fashion than from objects inheriting from the base annotation package class AnnotationDbi. In addition to the *standard* usage, the select and mapIds for EnsDb objects support also the filter framework of the ensembdb package and thus allow to perform more fine-grained queries to retrieve data.

Usage

```
## S4 method for signature 'EnsDb'
columns(x)
## S4 method for signature 'EnsDb'
keys(x, keytype, filter,...)
## S4 method for signature 'EnsDb'
keytypes(x)
```

```
## S4 method for signature 'EnsDb'
mapIds(x, keys, column, keytype, ..., multiVals)
## S4 method for signature 'EnsDb'
select(x, keys, columns, keytype, ...)
```

Arguments

	(In alphabetic order)
	For `mapIds`: the column to search on, i.e. from which values should be retrieved.
`columns`	For `select`: the columns from which values should be retrieved. Use the `columns` method to list all possible columns.
`keys`	The keys/ids for which data should be retrieved from the database. This can be either a character vector of keys/IDs, a single filter object extending `BasicFilter` or a list of such objects.
`keytype`	For `mapIds` and `select`: the type (column) that matches the provided keys. This argument does not have to be specified if argument `keys` is a filter object extending `BasicFilter` or a `list` of such objects.
	For `keys`: which keys should be returned from the database.
`filter`	For `keys`: either a single object extending `BasicFilter` or a list of such object to retrieve only specific keys from the database.
`multiVals`	What should `mapIds` do when there are multiple values that could be returned? Options are: `"first"`, `"list"`, `"filter"`, `"asNA"`. See `mapIds` for a detailed description.
`x`	The `EnsDb` object.
`...`	Not used.

Value

See method description above.

Methods and Functions

columns List all the columns that can be retrieved by the `mapIds` and `select` methods. Note that these column names are different from the ones supported by the `genes`, `transcripts` etc. methods that can be listed by the `listColumns` method.

Returns a character vector of supported column names.

keys Retrieves all keys from the column name specified with `keytype`. By default (if `keytype` is not provided) it returns all gene IDs. Note that `keytype="TXNAME"` will return transcript ids, since no transcript names are available in the database.

Returns a character vector of IDs.

keytypes List all supported key types (column names).

Returns a character vector of key types.

mapIds Retrieve the mapped ids for a set of keys that are of a particular keytype. Argument `keys` can be either a character vector of keys/IDs, a single filter object extending `BasicFilter` or a list of such objects. For the latter, the argument `keytype` does not have to be specified. Importantly however, if the filtering system is used, the ordering of the results might not represent the ordering of the keys.

The method usually returns a named character vector or, depending on the argument `multiVals` a named list, with names corresponding to the keys (same ordering is only guaranteed if `keys` is a character vector).

select Retrieve the data as a `data.frame` based on parameters for selected `keys`, `columns` and `keytype` arguments. Multiple matches of the keys are returned in one row for each possible match. Argument `keys` can be either a character vector of keys/IDs, a single filter object extending `BasicFilter` or a list of such objects. For the latter, the argument `keytype` does not have to be specified.

Note that values from a column `"TXNAME"` will be the same than for a column `"TXID"`, since internally no database column `"tx_name"` is present and the column is thus mapped to `"tx_id"`.

Returns a `data.frame` with the column names corresponding to the argument `columns` and rows with all data matching the criteria specified with `keys`.

Author(s)

Johannes Rainer

See Also

`BasicFilter` `listColumns` `transcripts`

Examples

```
library(EnsDb.Hsapiens.v75)
edb <- EnsDb.Hsapiens.v75

## List all supported keytypes.
keytypes(edb)

## List all supported columns for the select and mapIds methods.
columns(edb)

## List /real/ database column names.
listColumns(edb)

## Retrieve all keys corresponding to transcript ids.
txids <- keys(edb, keytype="TXID")
length(txids)
head(txids)

## Retrieve all keys corresponding to gene names of genes encoded on chromosome X
gids <- keys(edb, keytype="GENENAME", filter=SeqnameFilter("X"))
length(gids)
head(gids)

## Get a mapping of the genes BCL2 and BCL2L11 to all of their
## transcript ids and return the result as list
maps <- mapIds(edb, keys=c("BCL2", "BCL2L11"), column="TXID",
               keytype="GENENAME", multiVals="list")
maps

## Perform the same query using a combination of a GenenameFilter and a TxbiotypeFilter
## to just retrieve protein coding transcripts for these two genes.
mapIds(edb, keys=list(GenenameFilter(c("BCL2", "BCL2L11")),
                      TxbiotypeFilter("protein_coding")), column="TXID",
       multiVals="list")

## select:
```

```
## Retrieve all transcript and gene related information for the above example.
select(edb, keys=list(GenenameFilter(c("BCL2", "BCL2L11")),
                      TxbiotypeFilter("protein_coding")),
       columns=c("GENEID", "GENENAME", "TXID", "TXBIOTYPE", "TXSEQSTART", "TXSEQEND",
                 "SEQNAME", "SEQSTRAND"))

## Get all data for genes encoded on chromosome Y
Y <- select(edb, keys="Y", keytype="SEQNAME")
head(Y)
nrow(Y)

## Get selected columns for all lincRNAs encoded on chromosome Y
Y <- select(edb, keys=list(SeqnameFilter("Y"), GenebiotypeFilter("lincRNA")),
            columns=c("GENEID", "GENEBIOTYPE", "TXID", "GENENAME"))
head(Y)
nrow(Y)
```

SeqendFilter *Constructor functions for filter objects*

Description

These functions allow to create filter objects that can be used to retrieve specific elements from the annotation database.

Usage

```
EntrezidFilter(value, condition = "=")

GeneidFilter(value, condition = "=")

GenenameFilter(value, condition = "=")

GenebiotypeFilter(value, condition = "=")

GRangesFilter(value, condition="within", feature="gene")

TxidFilter(value, condition = "=")

TxbiotypeFilter(value, condition = "=")

ExonidFilter(value, condition = "=")

ExonrankFilter(value, condition = "=")

SeqnameFilter(value, condition = "=")

SeqstrandFilter(value, condition = "=")

SeqstartFilter(value, condition = "=", feature = "gene")
```

```
SeqendFilter(value, condition = "=", feature = "gene")

SymbolFilter(value, condition = "=")
```

Arguments

value	The filter value, e.g., for GeneidFilter the id of the gene for which the data should be retrieved. For character values (all filters except SeqstartFilter and SeqendFilter) also a character vector of values is allowed. Allowed values for SeqstrandFilter are: "+", "-", "1" or "-1". For GRangeFilter this has to be a GRanges object.
condition	The condition to be used in the comparison. For character values "=", "in" and "like" are allowed, for numeric values (SeqstartFilter and SeqendFilter) "=", ">", ">=", "<" and "<=". Note that for "like" value should be a SQL pattern (e.g. "ENS%"). For GRangesFilter, "within" and "overlapping" are allowed. See below for details.
feature	For SeqstartFilter and SeqendFilter: the chromosomal position of which features should be used in the filter (either "gene", "transcript" or "exon"). For GRangesFilter: the submitted value is overwritten internally depending on the called method, i.e. calling genes will set feature to "gene", transcripts to "tx" and exons to "exon".

Details

EntrezidFilter Filter results based on the NCBI Entrezgene ID of the genes.

GeneidFilter Filter results based on Ensembl gene IDs.

GenenameFilter Filter results based on gene names (gene symbols).

GenebiotypeFilter Filter results based on the biotype of the genes. For a complete list of available gene biotypes use the listGenebiotypes method.

GRangesFilter Allows to fetch features within or overlapping the specified genomic region(s)/range(s). This filter takes a GRanges object as input and, if condition="within" (the default) will restrict results to features (genes, transcripts or exons) that are completely within the region. Alternatively, by specifying condition="overlapping" it will return all features that are partially overlapping with the region, i.e. which start coordinate is smaller than the end coordinate of the region and which end coordinate is larger than the start coordinate of the region. Thus, genes and transcripts that have an intron overlapping the region will also be returned.

Note: if the specified GRanges object defines multiple region, all features within (or overlapping) any of these regions are returned.

See GRangesFilter for more details.

TxidFilter Filter results based on the Ensembl transcript IDs.

TxbiotypeFilter Filter results based on the biotype of the transcripts. For a complete list of available transcript biotypes use the listTxbiotypes method.

ExonidFilter Filter based on the Ensembl exon ID.

ExonrankFilter Filter results based on exon ranks (indices) of exons within transcripts.

SeqnameFilter Filter results based on the name of the sequence the features are encoded.

SeqstrandFilter Filter results based on the strand on which the features are encoded.

SeqstartFilter Filter results based on the (chromosomal) start coordinate of the features (exons, genes or transcripts).

SeqendFilter Filter results based on the (chromosomal) end coordinates.

SymbolFilter Filter results based on the gene names. The database does not provide an explicit *symbol* column, thus this filter uses the gene name instead (which in many cases corresponds to the official gene name).

Value

Depending on the function called an instance of: EntrezidFilter, GeneidFilter, GenenameFilter, GenebiotypeFilter, GRangesFilter, TxidFilter, TxbiotypeFilter, ExonidFilter, ExonrankFilter, SeqnameFilter, SeqstrandFilter, SeqstartFilter, SeqendFilter, SymbolFilter

Author(s)

Johannes Rainer

See Also

EntrezidFilter, GeneidFilter, GenenameFilter, GenebiotypeFilter, GRangesFilter, TxidFilter, TxbiotypeFilter, ExonidFilter, ExonrankFilter, SeqnameFilter, SeqstrandFilter, SeqstartFilter, SeqendFilter, SymbolFilter

Examples

```
## create a filter that could be used to retrieve all informations for
## the respective gene.
Gif <- GeneidFilter("ENSG00000012817")
Gif
## returns the where condition of the SQL querys
where(Gif)

## create a filter for a chromosomal end position of a gene
Sef <- SeqendFilter(100000, condition="<", "gene")
Sef

## To find genes within a certain chromosomal position filters should be
## combined:
Ssf <- SeqstartFilter(10000, condition=">", "gene")
Snf <- SeqnameFilter("2")
## combine the filters
Filter <- list(Ssf, Sef, Snf)

Filter

## generate the where SQL call for these filters:
where(Filter)


## Create a GRangesFilter
GRangesFilter(GRanges("X", IRanges(123, 5454)))

## Create a GRangesFilter with multiple ranges
grf <- GRangesFilter(GRanges(c("X", "Y"),
```

```
                              IRanges(start=c(123, 900),
                                      end=c(5454, 910))))
## Evaluate the 'where' SQL condition that would be applied.
where(grf)
## Change the "condition" of the filter and evaluate the
## 'where' condition again.
condition(grf) <- "overlapping"
where(grf)
```

---

seqlevelsStyle *Support for other than Ensembl seqlevel style*

Description

The methods and functions on this help page allow to integrate EnsDb objects and the annotations they provide with other Bioconductor annotation packages that base on chromosome names (seqlevels) that are different from those defined by Ensembl.

Usage

```
## S4 method for signature 'EnsDb'
seqlevelsStyle(x)

## S4 replacement method for signature 'EnsDb'
seqlevelsStyle(x) <- value

## S4 method for signature 'EnsDb'
supportedSeqlevelsStyles(x)
```

Arguments

	(In alphabetic order)
	For seqlevelsStyle<-: a character string specifying the seqlevels style that should be set. Use the supportedSeqlevelsStyle to list all available and supported seqlevel styles.
xalue	An EnsDb instance.

Value

For seqlevelsStyle: see method description above.

For supportedSeqlevelsStyles: see method description above.

Methods and Functions

seqlevelsStyle Get the style of the seqlevels in which results returned from the EnsDb object are encoded. By default, and internally, seqnames as provided by Ensembl are used.

The method returns a character string specifying the currently used seqlevelstyle.

seqlevelsStyle<- Change the style of the seqlevels in which results returned from the EnsDb object are encoded. Changing the seqlevels helps integrating annotations from EnsDb objects e.g. with annotations from packages that base on UCSC annotations.

supportedSeqlevelsStyles Lists all seqlevel styles for which mappings between seqlevel styles are available in the GenomeInfoDb package.

The method returns a character vector with supported seqlevel styles for the organism of the EnsDb object.

Note

The mapping between different seqname styles is performed based on data provided by the GenomeInfoDb package. Note that in most instances no mapping is provided for seqnames other than for primary chromosomes. By default functions from the ensembldb package return the *original* seqname is in such cases. This behaviour can be changed with the ensembldb.seqnameNotFound global option. For the special keyword "ORIGINAL" (the default), the original seqnames are returned, for "MISSING" an error is thrown if a seqname can not be mapped. In all other cases, the value of the option is returned as seqname if no mapping is available (e.g. setting options(ensembldb.seqnameNotFound=NA) returns an NA if the seqname is not mappable).

Author(s)

Johannes Rainer

See Also

EnsDb transcripts

Examples

```
library(EnsDb.Hsapiens.v75)
edb <- EnsDb.Hsapiens.v75

## Get the internal, default seqlevel style.
seqlevelsStyle(edb)

## Get the seqlevels from the database.
seqlevels(edb)

## Get all supported mappings for the organism of the EnsDb.
supportedSeqlevelsStyles(edb)

## Change the seqlevels to UCSC style.
seqlevelsStyle(edb) <- "UCSC"
seqlevels(edb)

## Change the option ensembldb.seqnameNotFound to return NA in case
## the seqname can not be mapped form Ensembl to UCSC.
options(ensembldb.seqnameNotFound=NA)

seqlevels(edb)

## Restoring the original setting.
options(ensembldb.seqnameNotFound="ORIGINAL")
```

```
## Integrate Ensembl based annotations with a BSgenome package that is based on
## UCSC style seqnames.
library(BSgenome.Hsapiens.UCSC.hg19)
bsg <- BSgenome.Hsapiens.UCSC.hg19

## Get the genome version
unique(genome(bsg))
unique(genome(edb))
## Although differently named, both represent genome build GRCh37.

## Extract the full transcript sequences of all lincRNAs encoded on chromsome Y.
yTxSeqs <- extractTranscriptSeqs(bsg, exonsBy(edb, "tx",
                                              filter=list(SeqnameFilter("chrY"),
                                                          GenebiotypeFilter("lincRNA"))))
yTxSeqs
```

`useMySQL,EnsDb-method` *Use a MySQL backend*

Description

Change the SQL backend from *SQLite* to *MySQL*. When first called on an `EnsDb` object, the function tries to create and save all of the data into a MySQL database. All subsequent calls will connect to the already existing MySQL database.

Usage

```
## S4 method for signature 'EnsDb'
useMySQL(x, host = "localhost", port = 3306, user, pass)
```

Arguments

`x`	The `EnsDb` object.
`host`	Character vector specifying the host on which the MySQL server runs.
`port`	The port on which the MySQL server can be accessed.
`user`	The user name for the MySQL server.
`pass`	The password for the MySQL server.

Details

This functionality requires that the `RMySQL` package is installed and that the user has (write) access to a running MySQL server. If the corresponding database does already exist users without write access can use this functionality.

Value

A `EnsDb` object providing access to the data stored in the MySQL backend.

Note

At present the function does not evaluate whether the versions between the SQLite and MySQL database differ.

Author(s)

Johannes Rainer

Examples

```
## Load the EnsDb database (SQLite backend).
library(EnsDb.Hsapiens.v75)
edb <- EnsDb.Hsapiens.v75
## Now change the backend to MySQL; my_user and my_pass should
## be the user name and password to access the MySQL server.
## Not run:
edb_mysql <- useMySQL(edb, host = "localhost", user = my_user, pass = my_pass)

## End(Not run)
```

Index