Package 'ensembldb'

November 12, 2026

Type Package

Title Utilities to create and use Ensembl-based annotation databases

Version 2.36.1

Author Johannes Rainer <johannes.rainer@eurac.edu> with contributions from Tim Triche, Sebastian Gibb, Laurent Gatto Christian Weichenberger and Boyu Yu.

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

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

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

Imports methods, RSQLite (>= 1.1), DBI, Biobase, Seqinfo, GenomeInfoDb (>= 1.45.5), AnnotationDbi (>= 1.31.19), rtracklayer (>= 1.69.1), S4Vectors (>= 0.23.10), Rsamtools, IRanges (>= 2.13.24), ProtGenerics, Biostrings (>= 2.77.2), curl

Depends R (>= 3.5.0), BiocGenerics (>= 0.15.10), GenomicRanges (>= 1.61.1), GenomicFeatures (>= 1.61.4), AnnotationFilter (>= 1.5.2)

Suggests BiocStyle, knitr, EnsDb.Hsapiens.v86 (>= 0.99.8), testthat, BSgenome.Hsapiens.NCBI.GRCh38, ggbio (>= 1.24.0), Gviz (>= 1.20.0), rmarkdown, AnnotationHub

Enhances RMariaDB, shiny

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, ensembldb provides a filter framework allowing to retrieve annotations for specific entries like genes encoded on a chromosome region or transcript models of lincRNA genes. EnsDb databases built with ensembldb contain also protein annotations and mappings between proteins and

their encoding transcripts. Finally, ensembldb provides functions to map between genomic, transcript and protein coordinates.

Collate 'Classes.R' 'Deprecated.R' 'Generics.R' 'Methods-Filter.R' 'Methods.R' 'dbhelpers.R' 'functions-EnsDb.R' 'functions-Filter.R' 'functions-create-EnsDb.R' 'functions-utils.R' 'proteinToX.R' 'transcriptToX.R' 'genomeToX.R' 'select-methods.R' 'seqname-utils.R' 'zzz.R'

biocViews Genetics, AnnotationData, Sequencing, Coverage

License LGPL

RoxygenNote 7.3.2

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

git_branch RELEASE_3_23

git_last_commit 5c98c2b

git_last_commit_date 2026-05-25

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

addFilter,EnsDb-method . . . 3
cdsToTranscript . . . 4
convertFilter,AnnotationFilter,EnsDb-method . . . 6
Deprecated . . . 7
EnsDb . . . 8
EnsDb-class . . . 10
exonsBy . . . 13
Filter-classes . . . 22
genomeToProtein . . . 27
genomeToTranscript . . . 29
getGeneRegionTrackForGviz . . . 31
getGenomeFaFile . . . 33
hasProteinData,EnsDb-method . . . 34
lengthOf . . . 35
listEnsDbs . . . 37
makeEnsembldbPackage . . . 38
proteins,EnsDb-method . . . 42
proteinToGenome . . . 44
proteinToTranscript . . . 47
runEnsDbApp . . . 50
select . . . 51
seqlevelsStyle . . . 54
transcriptToCds . . . 56
transcriptToGenome . . . 58
transcriptToProtein . . . 60

useMySQL,EnsDb-method 62

Index **64**

`addFilter,EnsDb-method`

Globally add filters to an EnsDb database

Description

These methods allow to set, delete or show globally defined filters on an `EnsDb` object.

`addFilter`: adds an annotation filter to the `EnsDb` object.

`dropFilter` deletes all globally set filters from the `EnsDb` object.

`activeFilter` returns the globally set filter from an `EnsDb` object.

`filter` filters an `EnsDb` object. `filter` is an alias for the `addFilter` function.

Usage

```
## S4 method for signature 'EnsDb'
addFilter(x, filter = AnnotationFilterList())

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

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

filter(x, filter = AnnotationFilterList())
```

Arguments

`x`	The `EnsDb` object to which the filter should be added.
`filter`	The filter as an `AnnotationFilter`, `AnnotationFilterList` or filter expression. See

Details

Adding a filter to an `EnsDb` object causes this filter to be permanently active. The filter will be used for all queries to the database and is added to all additional filters passed to the methods such as `genes`.

Value

`addFilter` and `filter` return an `EnsDb` object with the specified filter added.

`activeFilter` returns an `AnnotationFilterList` object being the active global filter or `NA` if no filter was added.

`dropFilter` returns an `EnsDb` object with all eventually present global filters removed.

Author(s)

Johannes Rainer

See Also

`Filter-classes` for a list of all supported filters.

Examples

```
library(EnsDb.Hsapiens.v86)
edb <- EnsDb.Hsapiens.v86

## Add a global SeqNameFilter to the database such that all subsequent
## queries will be applied on the filtered database.
edb_y <- addFilter(edb, SeqNameFilter("Y"))

## Note: using the filter function is equivalent to a call to addFilter.

## Each call returns now only features encoded on chromosome Y
gns <- genes(edb_y)

seqlevels(gns)

## Get all lincRNA gene transcripts on chromosome Y
transcripts(edb_y, filter = ~ gene_biotype == "lincRNA")

## Get the currently active global filter:
activeFilter(edb_y)

## Delete this filter again.
edb_y <- dropFilter(edb_y)

activeFilter(edb_y)
```

---

cdsToTranscript *Map positions within the CDS to coordinates relative to the start of the transcript*

Description

Converts CDS-relative coordinates to positions within the transcript, i.e. relative to the start of the transcript and hence including its 5' UTR.

Usage

```
cdsToTranscript(x, db, id = "name", exons = NA, transcripts = NA)
```

Arguments

x	IRanges with the coordinates within the CDS. Coordinates are expected to be relative to the transcription start (the first nucleotide of the transcript). The Ensembl IDs of the corresponding transcripts have to be provided either as names of the IRanges, or in one of its metadata columns.
db	EnsDb object.
id	character(1) specifying where the transcript identifier can be found. Has to be either "name" or one of colnames(mcols(prng)).
exons	CompressedGRangesList object generated by exonsBy() where by = 'tx'.
transcripts	GRanges object generated by transcripts().

Value

IRanges with the same length (and order) than the input IRanges x. Each element in IRanges provides the coordinates within the transcripts CDS. The transcript-relative coordinates are provided as metadata columns. IRanges with a start coordinate of -1 is returned for transcripts that are not known in the database, non-coding transcripts or if the provided start and/or end coordinates are not within the coding region.

Author(s)

Johannes Rainer

See Also

Other coordinate mapping functions: genomeToProtein(), genomeToTranscript(), proteinToGenome(), proteinToTranscript(), transcriptToCds(), transcriptToGenome(), transcriptToProtein()

Examples

```
library(EnsDb.Hsapiens.v86)
## Defining transcript-relative coordinates for 4 transcripts of the gene
## BCL2
txcoords <- IRanges(start = c(4, 3, 143, 147), width = 1,
    names = c("ENST00000398117", "ENST00000333681",
    "ENST00000590515", "ENST00000589955"))

cdsToTranscript(txcoords, EnsDb.Hsapiens.v86)

## Next we map the coordinate for variants within the gene PKP2 to the
## genome. The variants is PKP2 c.1643DelG and the provided
## position is thus relative to the CDS. We have to convert the
## position first to transcript-relative coordinates.
pkp2 <- IRanges(start = 1643, width = 1, name = "ENST00000070846")

## Map the coordinates by first converting the CDS- to transcript-relative
## coordinates
transcriptToGenome(cdsToTranscript(pkp2, EnsDb.Hsapiens.v86),
    EnsDb.Hsapiens.v86)
```

```
## Meanwhile, this function can be called in parallel processes if you preload
## the exons and transcripts database.

exons <- exonsBy(EnsDb.Hsapiens.v86)
transcripts <- transcripts(EnsDb.Hsapiens.v86)

cdsToTranscript(txcoords, EnsDb.Hsapiens.v86, exons = exons,transcripts = transcripts)
```

`convertFilter,AnnotationFilter,EnsDb-method`
Convert an AnnotationFilter to a SQL WHERE condition for EnsDb

---

Description

`convertFilter` converts an `AnnotationFilter::AnnotationFilter` or `AnnotationFilter::AnnotationFilterList` to an SQL where condition for an `EnsDb` database.

Usage

```
## S4 method for signature 'AnnotationFilter,EnsDb'
convertFilter(object, db, with.tables = character())

## S4 method for signature 'AnnotationFilterList,EnsDb'
convertFilter(object, db, with.tables = character())
```

Arguments

`object`	`AnnotationFilter` or `AnnotationFilterList` objects (or objects extending these classes).
`db`	`EnsDb` object.
`with.tables`	optional `character` vector specifying the names of the database tables that are being queried.

Value

A `character(1)` with the SQL where condition.

Note

This function *might* be used in direct SQL queries on the SQLite database underlying an `EnsDb` but is more thought to illustrate the use of `AnnotationFilter` objects in combination with SQL databases. This method is used internally to create the SQL calls to the database.

Author(s)

Johannes Rainer

Examples

```
library(EnsDb.Hsapiens.v86)
edb <- EnsDb.Hsapiens.v86

## Define a filter
flt <- AnnotationFilter(~ gene_name == "BCL2")

## Use the method from the AnnotationFilter package:
convertFilter(flt)

## Create a combination of filters
flt_list <- AnnotationFilter(~ gene_name %in% c("BCL2", "BCL2L11") &
    tx_biotype == "protein_coding")
flt_list

convertFilter(flt_list)

## Use the filters in the context of an EnsDb database:
convertFilter(flt, edb)

convertFilter(flt_list, edb)
```

`Deprecated` *Deprecated functionality*

Description

All functions, methods and classes listed on this page are deprecated and might be removed in future releases.

`GeneidFilter` creates a `GeneIdFilter`. Use `GeneIdFilter` from the `AnnotationFilter` package instead.

`GenebiotypeFilter` creates a `GeneBiotypeFilter`. Use `GeneBiotypeFilter` from the `AnnotationFilter` package instead.

`EntrezidFilter` creates a `EntrezFilter`. Use `EntrezFilter` from the `AnnotationFilter` package instead.

`TxidFilter` creates a `TxIdFilter`. Use `TxIdFilter` from the `AnnotationFilter` package instead.

`TxbiotypeFilter` creates a `TxBiotypeFilter`. Use `TxBiotypeFilter` from the `AnnotationFilter` package instead.

`ExonidFilter` creates a `ExonIdFilter`. Use `ExonIdFilter` from the `AnnotationFilter` package instead.

`ExonrankFilter` creates a `ExonRankFilter`. Use `ExonRankFilter` from the `AnnotationFilter` package instead.

`SeqNameFilter` creates a `SeqNameFilter`. Use `SeqNameFilter` from the `AnnotationFilter` package instead.

SeqstrandFilter creates a SeqStrandFilter. Use SeqStrandFilter from the AnnotationFilter instead.

SeqstartFilter creates a GeneStartFilter, TxStartFilter or ExonStartFilter depending on the value of the parameter feature. Use GeneStartFilter, TxStartFilter and ExonStartFilter instead.

SeqendFilter creates a GeneEndFilter, TxEndFilter or ExonEndFilter depending on the value of the parameter feature. Use GeneEndFilter, TxEndFilter and ExonEndFilter instead.

Usage

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

GenebiotypeFilter(value, condition = "==")

EntrezidFilter(value, condition = "==")

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")
```

Arguments

value	The value for the filter.
condition	The condition for the filter.
feature	For SeqstartFilter and SeqendFilter: on what type of feature should the filter be applied? Supported are "gene", "tx" and "exon".

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 MariaDB/MySQL database.

Details

By providing the connection to a MariaDB/MySQL database, it is possible to use MariaDB/MySQL as the database backend and queries will be performed on that database. Note however that this requires the package RMariaDB 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.v86)
EnsDb.Hsapiens.v86

## Alternatively, provide the full file name of a SQLite database file
dbfile <- system.file("extdata/EnsDb.Hsapiens.v86.sqlite", package = "EnsDb.Hsapiens.v86")
edb <- EnsDb(dbfile)
edb

## Third way: connect to a MySQL database
## Not run:
library(RMariaDB)
dbcon <- dbConnect(MySQL(), user = my_user, pass = my_pass,
    host = my_host, dbname = "ensdb_hsapiens_v86")
edb <- EnsDb(dbcon)

## End(Not run)
```

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

Description

The EnsDb class provides access to an Ensembl-based annotation package. This help page describes functions to get some basic informations from such an object.

Usage

```
## 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, metadata = FALSE, ...)

## 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.
`metadata`	For `listColumns`: whether columns from the metadata database column should also be returned. Defaults to `metadata = FALSE`.
`object`	For `organism`: an `EnsDb` instance.
`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.
`table`	For `listColumns`: optionally specify the table name(s) 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 `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

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, exonsBy, genes, transcripts, makeEnsemblSQLiteFromTables

addFilter for globally adding filters to an EnsDb object.

Examples

```
library(EnsDb.Hsapiens.v86)

## Display some information:
EnsDb.Hsapiens.v86

## Show the tables along with its columns
listTables(EnsDb.Hsapiens.v86)

## For what species is this database?
organism(EnsDb.Hsapiens.v86)

## What Ensembl version if the database based on?
ensemblVersion(EnsDb.Hsapiens.v86)

## Get some more information from the database
metadata(EnsDb.Hsapiens.v86)

## Get all the sequence names.
seqlevels(EnsDb.Hsapiens.v86)

## List all available gene biotypes from the database:
listGenebiotypes(EnsDb.Hsapiens.v86)

## List all available transcript biotypes:
listTxbiotypes(EnsDb.Hsapiens.v86)

## Update the EnsDb; this is in most instances not necessary at all.
updateEnsDb(EnsDb.Hsapiens.v86)

######    returnFilterColumns
returnFilterColumns(EnsDb.Hsapiens.v86)

## Get protein coding genes on chromosome X, specifying to return
## only columns gene_name as additional column.
genes(EnsDb.Hsapiens.v86, 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.v86) <- FALSE
genes(EnsDb.Hsapiens.v86, 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. Parameter filter enables to define filters to retrieve only specific data. Alternatively, a global filter might be added to the EnsDb object using the addFilter method.

Usage

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

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

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

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

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

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

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

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

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

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

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

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

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

Arguments

(In alphabetic order)

`...`	For `promoters`: additional arguments to be passed to the `transcripts` method. For `intronsByTranscript`: additional arguments such as `filter`.
`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 describing which results to retrieve from the database. Can be a single object extending `AnnotationFilter`, an `AnnotationFilterList` object combining several such objects or a `formula` representing a filter expression (see examples below or `AnnotationFilter` for more details). Use the `supportedFilters` method to get an overview of supported filter classes and related fields.
`maxgap`	For `exonsByOverlaps` and `transcriptsByOverlaps`: see `exonsByOverlaps` in `GenomicFeatures` for more information.

minoverlap	For exonsByOverlaps and transcriptsByOverlaps: see exonsByOverlaps in GenomicFeatures for more information.
order.by	Character vector specifying the column(s) by which the result should be ordered. This can be either in the form of "gene_id, seq_name" or c("gene_id", "seq_name").
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 in GenomicFeatures for more information.
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 column contains a list of Entrezgene identifiers to accommodate the potential 1:n mapping between Ensembl genes and Entrezgene IDs.

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.

gc_content the G and C nucleotide content of the transcript's sequence expressed as a percentage (i.e. between 0 and 100).

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.

Many EnsDb databases provide also protein related annotations. See listProteinColumns for more information.

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 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

Note that many methods and functions from the GenomicFeatures package can also be used for EnsDb objects (such as exonicParts, intronicParts etc).

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.

intronsByTranscript Retrieve introns by transcripts. Filters can also be passed to the function. For more information see the intronsByTranscript method in the GenomicFeatures package.

exonsByOverlaps Retrieve exons overlapping specified genomic ranges. For more information see the 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. Note that column "entrezid" is a list of Entrezgene identifiers to accomodate the potential 1:n mapping between Ensembl genes and Entrezgene IDs.

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

`supportedFilters` to get an overview of supported filters. `makeEnsembldbPackage`, `listColumns`, `lengthOf`

`addFilter` for globally adding filters to an `EnsDb` object.

Examples

```
library(EnsDb.Hsapiens.v86)
edb <- EnsDb.Hsapiens.v86

######   genes
##
## Get all genes encoded on chromosome Y
AllY <- genes(edb, filter = SeqNameFilter("Y"))
AllY

## Return the result as a DataFrame; also, we use a filter expression here
## to define which features to extract from the database.
AllY.granges <- genes(edb,
                      filter = ~ seq_name == "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 protein coding transcript for the gene ENSG00000184895
Exon <- exons(edb,
              filter = ~ gene_id == "ENSG00000184895" &
                  tx_biotype == "protein_coding",
              columns = c("gene_id", "gene_seq_start", "gene_seq_end",
                          "tx_biotype", "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"
eg <- exonsBy(edb, by = "gene",
              filter = AnnotationFilterList(SeqNameFilter(c("X", "Y")),
                                            GeneIdFilter("ENS", "startsWith")),
              columns = c("gene_biotype", "gene_name"))
eg
length(grep(names(eg), pattern="LRG"))

#####    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, type = "any")
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")
```

`Filter-classes` *Filters supported by ensembldb*

Description

`ensembldb` supports most of the filters from the AnnotationFilter package to retrieve specific content from EnsDb databases. These filters can be passed to the methods such as `genes()` with the `filter` parameter or can be added as a *global* filter to an `EnsDb` object (see `addFilter()` for more details). Use `supportedFilters()` to get an overview of all filters supported by `EnsDb` object.

`seqnames`: accessor for the sequence names of the `GRanges` object within a `GRangesFilter`.

`seqnames`: accessor for the `seqlevels` of the `GRanges` object within a `GRangesFilter`.

`supportedFilters` returns a `data.frame` with the names of all filters and the corresponding field supported by the `EnsDb` object.

Usage

```
OnlyCodingTxFilter()

ProtDomIdFilter(value, condition = "==")

ProteinDomainIdFilter(value, condition = "==")

ProteinDomainSourceFilter(value, condition = "==")

UniprotDbFilter(value, condition = "==")

UniprotMappingTypeFilter(value, condition = "==")

TxSupportLevelFilter(value, condition = "==")

TxIsCanonicalFilter(value, condition = "==")

TxExternalNameFilter(value, condition = "==")

## S4 method for signature 'GRangesFilter'
seqnames(x)

## S4 method for signature 'GRangesFilter'
seqlevels(x)
```

```
## S4 method for signature 'EnsDb'
supportedFilters(object, ...)
```

Arguments

value	The value(s) for the filter. For GRangesFilter it has to be a GRanges object.
condition	character(1) specifying the *condition* of the filter. For character-based filters (such as GeneIdFilter) "==", "!=", "startsWith" and "endsWith" are supported. Allowed values for integer-based filters (such as GeneStartFilter) are "==", "!=", "<". "<=", ">" and ">=".
x	For seqnames, seqlevels: a GRangesFilter object.
object	For supportedFilters: an EnsDb object.
...	For supportedFilters: currently not used.

Details

ensembldb supports the following filters from the AnnotationFilter package:

- GeneIdFilter: filter based on the Ensembl gene ID.
- GeneNameFilter: filter based on the name of the gene as provided Ensembl. In most cases this will correspond to the official gene symbol.
- SymbolFilter filter based on the gene names. EnsDb objects don't have a dedicated *symbol* column, the filtering is hence based on the gene names.
- GeneBiotype: filter based on the biotype of genes (e.g. "protein_coding").
- GeneStartFilter: filter based on the genomic start coordinate of genes.
- GeneEndFilter: filter based on the genomic end coordinate of genes.
- EntrezidFilter: filter based on the genes' NCBI Entrezgene ID.
- TxIdFilter: filter based on the Ensembld transcript ID.
- TxNameFilter: to be compliant with TxDb object from the GenomicFeatures package tx_name in fact represents the Ensembl transcript ID. Thus, the the tx_id and tx_name columns contain the same information and the TxIdFilter and TxNameFilter are in fact identical. The names of transcripts (i.e. the *external name* field in Ensembl are stored in column "tx_external_name" (and which can be filtered using the TxExternalNameFilter.
- TxBiotypeFilter: filter based on the transcripts' biotype.
- TxStartFilter: filter based on the genomic start coordinate of the transcripts.
- TxEndFilter: filter based on the genonic end coordinates of the transcripts.
- ExonIdFilter: filter based on Ensembl exon IDs.
- ExonRankFilter: filter based on the index/rank of the exon within the transcrips.
- ExonStartFilter: filter based on the genomic start coordinates of the exons.
- ExonEndFilter: filter based on the genomic end coordinates of the exons.

- `GRangesFilter`: Allows to fetch features within or overlapping specified genomic region(s)/range(s). This filter takes a `GRanges` object as input and, if `type = "any"` (the default) will restrict results to features (genes, transcripts or exons) that are partially overlapping the region. Alternatively, by specifying `condition = "within"` it will return features located within the range. In addition, the `GRangesFilter` `condition = "start"`, `condition = "end"` and `condition = "equal"` filtering for features with the same start or end coordinate or that are equal to the `GRanges`.

  Note that the type of feature on which the filter is applied depends on the method that is called, i.e. `genes()` will filter on the genomic coordinates of genes, `transcripts()` on those of transcripts and `exons()` on exon coordinates.

  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.

  If the specified `GRanges` object defines multiple regions, 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.
- `SeqNameFilter`: filter based on chromosome names.
- `SeqStrandFilter`: filter based on the chromosome strand. The strand can be specified with `value = "+"`, `value = "-"`, `value = -1` or `value = 1`.
- `ProteinIdFilter`: filter based on Ensembl protein IDs. This filter is only supported if the `EnsDb` provides protein annotations; use the `hasProteinData()` method to check.
- `UniprotFilter`: filter based on Uniprot IDs. This filter is only supported if the `EnsDb` provides protein annotations; use the `hasProteinData()` method to check.

In addition, the following filters are defined by `ensembldb`:

- `TxExternalNameFilter`: filter based on the transcript's *external name* (if available).
- `TxSupportLevel`: allows to filter results using the provided transcript support level. Support levels for transcripts are defined by Ensembl based on the available evidences for a transcript with 1 being the highest evidence grade and 5 the lowest level. This filter is only supported on `EnsDb` databases with a db schema version higher 2.1.
- `UniprotDbFilter`: allows to filter results based on the specified Uniprot database name(s).
- `UniprotMappingTypeFilter`: allows to filter results based on the mapping method/type that was used to assign Uniprot IDs to Ensembl protein IDs.
- `ProtDomIdFilter`, `ProteinDomainIdFilter`: allows to retrieve entries from the database matching the provided filter criteria based on their protein domain ID (*protein_domain_id*).
- `ProteinDomainSourceFilter`: filter results based on the source (database/method) defining the protein domain (e.g. `"pfam"`).
- `OnlyCodingTxFilter`: allows to retrieve entries only for protein coding transcripts, i.e. transcripts with a CDS. This filter does not take any input arguments.

Value

For `ProtDomIdFilter`: A `ProtDomIdFilter` object.

For `ProteinDomainIdFilter`: A `ProteinDomainIdFilter` object.

For ProteinDomainSourceFilter: A ProteinDomainSourceFilter object.

For UniprotDbFilter: A UniprotDbFilter object.

For UniprotMappingTypeFilter: A UniprotMappingTypeFilter object.

For TxSupportLevel: A TxSupportLevel object.

For TxIsCanonicalFilter: A TxIsCanonicalFilter object.

For TxExternalNameFilter: A TxExternalNameFilter object.

For supportedFilters: a data.frame with the names and the corresponding field of the supported filter classes.

Note

For users of ensembldb version < 2.0: in the GRangesFilter from the AnnotationFilter package the condition parameter was renamed to type (to be consistent with the IRanges package). In addition, condition = "overlapping" is no longer recognized. To retrieve all features overlapping the range type = "any" has to be used.

Protein annotation based filters can only be used if the EnsDb database contains protein annotations, i.e. if hasProteinData is TRUE. Also, only protein coding transcripts will have protein annotations available, thus, non-coding transcripts/genes will not be returned by the queries using protein annotation filters.

Author(s)

Johannes Rainer

See Also

supportedFilters() to list all filters supported for EnsDb objects.

listUniprotDbs() and listUniprotMappingTypes() to list all Uniprot database names respectively mapping method types from the database.

GeneIdFilter() in the AnnotationFilter package for more details on the filter objects.

genes(), transcripts(), exons(), listGenebiotypes(), listTxbiotypes().

addFilter() and filter() for globally adding filters to an EnsDb.

Examples

```
## Create a filter that could be used to retrieve all informations for
## the respective gene.
gif <- GeneIdFilter("ENSG00000012817")
gif

## Create a filter for a chromosomal end position of a gene
sef <- GeneEndFilter(10000, condition = ">")
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(114129278, 114129328),
                             strand = "+"), type = "any")
library(EnsDb.Hsapiens.v86)
edb <- EnsDb.Hsapiens.v86
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"))


## Create a ProteinIdFilter:
pf <- ProteinIdFilter("ENSP00000362111")
pf
## Using this filter would retrieve all database entries that are associated
## with a protein with the ID "ENSP00000362111"
```

```
if (hasProteinData(edb)) {
    res <- genes(edb, filter = pf)
    res
}

## UniprotFilter:
uf <- UniprotFilter("O60762")
## Get the transcripts encoding that protein:
if (hasProteinData(edb)) {
    transcripts(edb, filter = uf)
    ## The mapping Ensembl protein ID to Uniprot ID can however be 1:n:
    transcripts(edb, filter = TxIdFilter("ENST00000371588"),
        columns = c("protein_id", "uniprot_id"))
}

## ProtDomIdFilter:
pdf <- ProtDomIdFilter("PF00335")
## Also here we could get all transcripts related to that protein domain
if (hasProteinData(edb)) {
    transcripts(edb, filter = pdf, columns = "protein_id")
}
```

`genomeToProtein` *Map genomic coordinates to protein coordinates*

Description

Map positions along the genome to positions within the protein sequence if a protein is encoded at the location. The provided coordinates have to be completely within the genomic position of an exon of a protein coding transcript (see `genomeToTranscript()` for details). Also, the provided positions have to be within the genomic region encoding the CDS of a transcript (excluding its stop codon; soo `transcriptToProtein()` for details).

For genomic positions for which the mapping failed an `IRanges` with negative coordinates (i.e. a start position of -1) is returned.

Usage

```
genomeToProtein(x, db, proteins = NA, exons = NA, transcripts = NA)
```

Arguments

`x`	`GRanges` with the genomic coordinates that should be mapped to within-protein coordinates.
`db`	`EnsDb` object.
`proteins`	`DFrame` object generated by `proteins()`.
`exons`	`CompressedGRangesList` object generated by `exonsBy()` where by = 'tx'.
`transcripts`	`GRanges` object generated by `transcripts()`.

Details

genomeToProtein combines calls to genomeToTranscript() and transcriptToProtein().

Value

An IRangesList with each element representing the mapping of one of the GRanges in x (i.e. the length of the IRangesList is length(x)). Each element in IRanges provides the coordinates within the protein sequence, names being the (Ensembl) IDs of the protein. The ID of the transcript encoding the protein, the ID of the exon within which the genomic coordinates are located and its rank in the transcript are provided in metadata columns "tx_id", "exon_id" and "exon_rank". Metadata columns "cds_ok" indicates whether the length of the CDS matches the length of the encoded protein. Coordinates for which cds_ok = FALSE should be taken with caution, as they might not be correct. Metadata columns "seq_start", "seq_end", "seq_name" and "seq_strand" provide the provided genomic coordinates.

For genomic coordinates that can not be mapped to within-protein sequences an IRanges with a start coordinate of -1 is returned.

Author(s)

Johannes Rainer

See Also

Other coordinate mapping functions: cdsToTranscript(), genomeToTranscript(), proteinToGenome(), proteinToTranscript(), transcriptToCds(), transcriptToGenome(), transcriptToProtein()

Examples

```
library(EnsDb.Hsapiens.v86)
## Restrict all further queries to chromosome x to speed up the examples
edbx <- filter(EnsDb.Hsapiens.v86, filter = ~ seq_name == "X")

## In the example below we define 4 genomic regions:
## 630898: corresponds to the first nt of the CDS of ENST00000381578
## 644636: last nt of the CDS of ENST00000381578
## 644633: last nt before the stop codon in ENST00000381578
## 634829: position within an intron.
gnm <- GRanges("X", IRanges(start = c(630898, 644636, 644633, 634829),
    width = c(5, 1, 1, 3)))
res <- genomeToProtein(gnm, edbx)

## The result is an IRangesList with the same length as gnm
length(res)
length(gnm)

## The first element represents the mapping for the first GRanges:
## the coordinate is mapped to the first amino acid of the protein(s).
## The genomic coordinates can be mapped to several transcripts (and hence
## proteins).
res[[1]]
```

```
## The stop codon is not translated, thus the mapping for the second
## GRanges fails
res[[2]]

## The 3rd GRanges is mapped to the last amino acid.
res[[3]]

## Mapping of intronic positions fail
res[[4]]

## Meanwhile, this function can be called in parallel processes if you preload
## the protein, exons and transcripts database.

proteins <- proteins(edbx)
exons <- exonsBy(edbx)
transcripts <- transcripts(edbx)

genomeToProtein(gnm, edbx, proteins = proteins, exons = exons, transcripts = transcripts)
```

---

genomeToTranscript *Map genomic coordinates to transcript coordinates*

Description

genomeToTranscript maps genomic coordinates to positions within the transcript (if at the provided genomic position a transcript is encoded). The function does only support mapping of genomic coordinates that are completely within the genomic region at which an exon is encoded. If the genomic region crosses the exon boundary an empty IRanges is returned. See examples for details.

Usage

```
genomeToTranscript(x, db)
```

Arguments

x	GRanges object with the genomic coordinates that should be mapped.
db	EnsDb object or pre-loaded exons 'CompressedGRangesList' object using exonsBy().

Details

The function first retrieves all exons overlapping the provided genomic coordinates and identifies then exons that are fully containing the coordinates in x. The transcript-relative coordinates are calculated based on the relative position of the provided genomic coordinates in this exon.

Value

An IRangesList with length equal to length(x). Each element providing the mapping(s) to position within any encoded transcripts at the respective genomic location as an IRanges object. An IRanges with negative start coordinates is returned, if the provided genomic coordinates are not completely within the genomic coordinates of an exon.

The ID of the exon and its rank (index of the exon in the transcript) are provided in the result's IRanges metadata columns as well as the genomic position of x.

Note

The function throws a warning and returns an empty IRanges object if the genomic coordinates can not be mapped to a transcript.

Author(s)

Johannes Rainer

See Also

Other coordinate mapping functions: cdsToTranscript(), genomeToProtein(), proteinToGenome(), proteinToTranscript(), transcriptToCds(), transcriptToGenome(), transcriptToProtein()

Examples

```
library(EnsDb.Hsapiens.v86)

## Subsetting the EnsDb object to chromosome X only to speed up execution
## time of examples
edbx <- filter(EnsDb.Hsapiens.v86, filter = ~ seq_name == "X")

## Define a genomic region and calculate within-transcript coordinates
gnm <- GRanges("X:107716399-107716401")

res <- genomeToTranscript(gnm, edbx)
## Result is an IRanges object with the start and end coordinates within
## each transcript that has an exon at the genomic range.
res

## An IRanges with negative coordinates is returned if at the provided
## position no exon is present. Below we use the same coordinates but
## specify that the coordinates are on the forward (+) strand
gnm <- GRanges("X:107716399-107716401:+")
genomeToTranscript(gnm, edbx)

## Next we provide multiple genomic positions.
gnm <- GRanges("X", IRanges(start = c(644635, 107716399, 107716399),
    end = c(644639, 107716401, 107716401)), strand = c("*", "*", "+"))

## The result of the mapping is an IRangesList each element providing the
## within-transcript coordinates for each input region
genomeToTranscript(gnm, edbx)
```

```
## If you are tring to calculate within-transcript coordinates of a huge
## list of genomic region, you shall use pre-loaded exons GRangesList to
## replace the SQLite db edbx

## Below is just a lazy demo of querying multiple genomic region
library(parallel)

gnm <- rep(GRanges("X:107715899-107715901"),10)

exons <- exonsBy(EnsDb.Hsapiens.v86)

## You can pre-define the exons region to further accelerate the code.

exons <- exonsBy(
    EnsDb.Hsapiens.v86, by = "tx",
    filter = AnnotationFilterList(
        SeqNameFilter(as.character(unique(seqnames(gnm)))),
        GeneStartFilter(max(end(gnm)), condition = "<="),
        GeneEndFilter(min(start(gnm)), condition = ">=")
    )
)

## only run in Linux ##
# res_temp <- mclapply(1:10, function(ind){
#     genomeToTranscript(gnm[ind], exons)
# }, mc.preschedule = TRUE, mc.cores = detectCores() - 1)

# res <- do.call(c,res_temp)

cl <- makeCluster(detectCores() - 1)
clusterExport(cl,c('genomeToTranscript','gnm','exons'))

res <- parLapply(cl,1:10,function(ind){
   genomeToTranscript(gnm[ind], exons)
})
stopCluster(cl)
```

`getGeneRegionTrackForGviz` *Utility functions*

Description

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

Usage

```
## S4 method for signature 'EnsDb'
getGeneRegionTrackForGviz(x,
```

```
filter = AnnotationFilterList(), chromosome = NULL,
start = NULL, end = NULL, featureIs = "gene_biotype")
```

Arguments

(In alphabetic order)

chromosome	For getGeneRegionTrackForGviz: optional chromosome name to restrict the returned entry to a specific chromosome.
end	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 describing which results to retrieve from the database. Can be a single object extending AnnotationFilter, an AnnotationFilterList object combining several such objects or a formula representing a filter expression (see examples below or AnnotationFilter for more details).
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

transcripts

Examples

```
library(EnsDb.Hsapiens.v86)
edb <- EnsDb.Hsapiens.v86
######   getGeneRegionTrackForGviz
##
## Get all genes encoded on chromosome Y in the specifyed region.
AllY <- getGeneRegionTrackForGviz(edb, chromosome = "Y", start = 5131959,
                                  end = 7131959)
## We could plot this now using plotTracks(GeneRegionTrack(AllY))
```

`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")

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

Arguments

(In alphabetic order)

`pattern`	For method `getGenomeFaFile`: the pattern to be used to identify the fasta file representing genomic DNA sequence.
`x`	An `EnsDb` instance.

Value

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

For `getGenomeTwoBitFile`: a `TwoBitFile-class` object with the genome 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 matching 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.

getGenomeTwoBitFile Returns a TwoBitFile-class (defined in the rtracklayer package) with the genomeic sequence of the genome build matching the Ensembl version of the EnsDb object. The file is retrieved from AnnotationHub and hence requires (at least for the first query) an active internet connection to download the respective resource. If no DNA sequence matching the Ensembl version of x is available, the function tries to find the genomic sequence of the best matching genome build (closest Ensembl release) and returns that.

See the ensembldb vignette for details.

Author(s)

Johannes Rainer

See Also

transcripts exonsBy

Examples

```
## Loading an EnsDb for Ensembl version 86 (genome GRCh38):
library(EnsDb.Hsapiens.v86)
edb <- EnsDb.Hsapiens.v86

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


## End(Not run)
```

---

hasProteinData,EnsDb-method

Determine whether protein data is available in the database

Description

Determines whether the EnsDb provides protein annotation data.

Usage

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

Arguments

x	The EnsDb object.

Value

A logical of length one, TRUE if protein annotations are available and FALSE otherwise.

Author(s)

Johannes Rainer

See Also

listTables

Examples

```
library(EnsDb.Hsapiens.v86)
## Does this database/package have protein annotations?
hasProteinData(EnsDb.Hsapiens.v86)
```

---

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 = AnnotationFilterList())
```

Arguments

(In alphabetic order)

filter	A filter describing which results to retrieve from the database. Can be a single object extending AnnotationFilter, an AnnotationFilterList object combining several such objects or a formula representing a filter expression (see examples below or AnnotationFilter for more details).
of	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.v86)
edb <- EnsDb.Hsapiens.v86

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

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

## Average length of all protein coding genes encoded on chromosomes X
mean(lengthOf(edb, of = "gene",
              filter = ~ gene_biotype == "protein_coding" &
                  seq_name == "X"))
```

```
## Average length of all snoRNAs
mean(lengthOf(edb, of = "gene",
              filter = ~ gene_biotype == "snoRNA" &
                  seq_name == "X"))

##### 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 MariaDB/MySQL server*

Description

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

Usage

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

Arguments

`dbcon`	A `DBIConnection` object providing access to a MariaDB/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 MariaDB/MySQL server (usually `3306`).
`user`	The username for the MariaDB/MySQL server.
`pass`	The password for the MariaDB/MySQL server.

Details

The use of this function requires the `RMariaDB` package to be installed. In addition user credentials to access a MySQL server (with already installed EnsDb databases), or with write access are required. For the latter EnsDb databases can be added with the `useMySQL` method. EnsDb databases in a MariaDB/MySQL server follow the same naming conventions than EnsDb packages, with the exception that the name is all lower case and that each `"."` 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(RMariaDB)
dbcon <- dbConnect(MariaDB(), 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)

ah	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.
...	Currently not used.

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)
```

`proteins,EnsDb-method` *Protein related functionality*

Description

This help page provides information about most of the functionality related to protein annotations in `ensembldb`.

The `proteins` method retrieves protein related annotations from an `EnsDb` database.

The `listUniprotDbs` method lists all Uniprot database names in the `EnsDb`.

The `listUniprotMappingTypes` method lists all methods that were used for the mapping of Uniprot IDs to Ensembl protein IDs.

The `listProteinColumns` function allows to conveniently extract all database columns containing protein annotations from an `EnsDb` database.

Usage

```
## S4 method for signature 'EnsDb'
proteins(
  object,
  columns = listColumns(object, "protein"),
  filter = AnnotationFilterList(),
  order.by = "",
```

```
  order.type = "asc",
  return.type = "DataFrame"
)

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

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

listProteinColumns(object)
```

Arguments

object	The EnsDb object.
columns	For proteins: character vector defining the columns to be extracted from the database. Can be any column(s) listed by the listColumns method.
filter	For proteins: A filter object extending AnnotationFilter or a list of such objects to select specific entries from the database. See Filter-classes for a documentation of available filters and use supportedFilters to get the full list of supported filters.
order.by	For proteins: a character vector specifying the column(s) by which the result should be ordered.
order.type	For proteins: if the results should be ordered ascending (order.type = "asc") or descending (order.type = "desc")
return.type	For proteins: character of lenght one specifying the type of the returned object. Can be either "DataFrame", "data.frame" or "AAStringSet".

Details

The proteins method performs the query starting from the protein tables and can hence return all annotations from the database that are related to proteins and transcripts encoding these proteins from the database. Since proteins does thus only query annotations for protein coding transcripts, the genes or transcripts methods have to be used to retrieve annotations for non-coding transcripts.

Value

The proteins method returns protein related annotations from an EnsDb object with its return.type argument allowing to define the type of the returned object. Note that if return.type = "AAStringSet" additional annotation columns are stored in a DataFrame that can be accessed with the mcols method on the returned object.

The listProteinColumns function returns a character vector with the column names containing protein annotations or throws an error if no such annotations are available.

Author(s)

Johannes Rainer

Examples

```
library(ensembldb)
library(EnsDb.Hsapiens.v86)
edb <- EnsDb.Hsapiens.v86
## Get all proteins from tha database for the gene ZBTB16, if protein
## annotations are available
if (hasProteinData(edb))
    proteins(edb, filter = GeneNameFilter("ZBTB16"))

## List the names of all Uniprot databases from which Uniprot IDs are
## available in the EnsDb
if (hasProteinData(edb))
    listUniprotDbs(edb)


## List the type of all methods that were used to map Uniprot IDs to Ensembl
## protein IDs
if (hasProteinData(edb))
    listUniprotMappingTypes(edb)


## List all columns containing protein annotations
library(EnsDb.Hsapiens.v86)
edb <- EnsDb.Hsapiens.v86
if (hasProteinData(edb))
    listProteinColumns(edb)
```

`proteinToGenome` *Map within-protein coordinates to genomic coordinates*

Description

`proteinToGenome` maps protein-relative coordinates to genomic coordinates based on the genomic coordinates of the CDS of the encoding transcript. The encoding transcript is identified using protein-to-transcript annotations (and eventually Uniprot to Ensembl protein identifier mappings) from the submitted `EnsDb` object (and thus based on annotations from Ensembl).

The regions within the protein sequence need to be provided as a named `IRanges` object with the names being protein identifiers and the start and end coordinates (within these proteins) defined by the `IRanges` object. As an alternative to the `IRanges`' names, protein identifiers can also be provided through a metadata column (see details below).

Note that not all coding regions for protein coding transcripts are complete, and the function thus checks also if the length of the coding region matches the length of the protein sequence and throws a warning if that is not the case.

The genomic coordinates for the within-protein coordinates, the Ensembl protein ID, the ID of the encoding transcript and the within protein start and end coordinates are reported for each input range.

Usage

```
## S4 method for signature 'EnsDb'
proteinToGenome(x, db, id = "name", idType = "protein_id")

## S4 method for signature 'CompressedGRangesList'
proteinToGenome(x, db, id = "name", idType = "protein_id")
```

Arguments

x	IRanges with the coordinates within the protein(s). The object has also to provide some means to identify the protein (see details).
db	For the method for EnsDb objects: An EnsDb object to be used to retrieve genomic coordinates of encoding transcripts. For the method for CompressedGRangesList objects: A CompressedGRangesList object generated by cdsBy() where by = 'tx' and columns = c('tx_id' ,'protein_id','uniprot_id','protein_sequence').
id	character(1) specifying where the protein identifier can be found. Has to be either "name" or one of colnames(mcols(x)).
idType	character(1) defining what type of IDs are provided. Has to be one of "protein_id" (default), "uniprot_id" or "tx_id".

Details

Protein identifiers (supported are Ensembl protein IDs or Uniprot IDs) can be passed to the function as names of the x IRanges object, or alternatively in any one of the metadata columns (mcols) of x.

Value

list, each element being the mapping results for one of the input ranges in x and names being the IDs used for the mapping. Each element can be either a:

- GRanges object with the genomic coordinates calculated on the protein-relative coordinates for the respective Ensembl protein (stored in the "protein_id" metadata column.
- GRangesList object, if the provided protein identifier in x was mapped to several Ensembl protein IDs (e.g. if Uniprot identifiers were used). Each element in this GRangesList is a GRanges with the genomic coordinates calculated for the protein-relative coordinates from the respective Ensembl protein ID.

The following metadata columns are available in each GRanges in the result:

- "protein_id": the ID of the Ensembl protein for which the within-protein coordinates were mapped to the genome.
- "tx_id": the Ensembl transcript ID of the encoding transcript.
- "exon_id": ID of the exons that have overlapping genomic coordinates.
- "exon_rank": the rank/index of the exon within the encoding transcript.
- "cds_ok": contains TRUE if the length of the CDS matches the length of the amino acid sequence and FALSE otherwise.
- "protein_start": the within-protein sequence start coordinate of the mapping.

- "protein_end": the within-protein sequence end coordinate of the mapping.

Genomic coordinates are returned ordered by the exon index within the transcript.

Note

While the mapping for Ensembl protein IDs to encoding transcripts (and thus CDS) is 1:1, the mapping between Uniprot identifiers and encoding transcripts (which is based on Ensembl annotations) can be one to many. In such cases proteinToGenome calculates genomic coordinates for within-protein coordinates for all of the annotated Ensembl proteins and returns all of them. See below for examples.

Mapping using Uniprot identifiers needs also additional internal checks that have a significant impact on the performance of the function. It is thus strongly suggested to first identify the Ensembl protein identifiers for the list of input Uniprot identifiers (e.g. using the proteins() function and use these as input for the mapping function.

A warning is thrown for proteins which sequence does not match the coding sequence length of any encoding transcripts. For such proteins/transcripts a FALSE is reported in the respective "cds_ok" metadata column. The most common reason for such discrepancies are incomplete 3' or 5' ends of the CDS. The positions within the protein might not be correclty mapped to the genome in such cases and it might be required to check the mapping manually in the Ensembl genome browser.

Author(s)

Johannes Rainer based on initial code from Laurent Gatto and Sebastian Gibb

See Also

proteinToGenome in the **GenomicFeatures** package for methods that operate on a TxDb or GRangesList object.

Other coordinate mapping functions: cdsToTranscript(), genomeToProtein(), genomeToTranscript(), proteinToTranscript(), transcriptToCds(), transcriptToGenome(), transcriptToProtein()

Examples

```
library(EnsDb.Hsapiens.v86)
## Restrict all further queries to chromosome x to speed up the examples
edbx <- filter(EnsDb.Hsapiens.v86, filter = ~ seq_name == "X")

## Define an IRange with protein-relative coordinates within a protein for
## the gene SYP
syp <- IRanges(start = 4, end = 17)
names(syp) <- "ENSP00000418169"
res <- proteinToGenome(syp, edbx)
res
## Positions 4 to 17 within the protein span two exons of the encoding
## transcript.

## Perform the mapping for multiple proteins identified by their Uniprot
## IDs.
ids <- c("O15266", "Q9HBJ8", "unexistant")
```

```
prngs <- IRanges(start = c(13, 43, 100), end = c(21, 80, 100))
names(prngs) <- ids

res <- proteinToGenome(prngs, edbx, idType = "uniprot_id")

## The result is a list, same length as the input object
length(res)
names(res)

## No protein/encoding transcript could be found for the last one
res[[3]]

## The first protein could be mapped to multiple Ensembl proteins. The
## mapping result using all of their encoding transcripts are returned
res[[1]]

## The coordinates within the second protein span two exons
res[[2]]

## Meanwhile, this function can be called in parallel processes if you preload
## the CDS data with desired data columns
cds <- cdsBy(edbx,columns = c(listColumns(edbx,'tx'),'protein_id','uniprot_id','protein_sequence'))
# cds <- cdsBy(edbx,columns = c(listColumns(edbx,'tx'),'protein_id','protein_sequence'))
# cds <- cdsBy(edbx,columns = c('tx_id','protein_id','protein_sequence'))
## Define an IRange with protein-relative coordinates within a protein for
## the gene SYP
syp <- IRanges(start = 4, end = 17)
names(syp) <- "ENSP00000418169"
res <- proteinToGenome(syp, cds)
res
## Positions 4 to 17 within the protein span two exons of the encoding
## transcript.

## Perform the mapping for multiple proteins identified by their Uniprot
## IDs.
ids <- c("O15266", "Q9HBJ8", "unexistant")
prngs <- IRanges(start = c(13, 43, 100), end = c(21, 80, 100))
names(prngs) <- ids

res <- proteinToGenome(prngs, cds, idType = "uniprot_id")
```

---

proteinToTranscript *Map protein-relative coordinates to positions within the transcript*

Description

proteinToTranscript maps protein-relative coordinates to positions within the encoding transcript. Note that the returned positions are relative to the complete transcript length, which includes the 5' UTR.

The regions within the protein sequence need to be provided as a named IRanges object with the names being protein identifiers and the start and end coordinates (within these proteins) defined by the IRanges object. As an alternative to the IRanges' names, protein identifiers can also be provided through a metadata column (see details below).

Similar to the proteinToGenome() function, proteinToTranscript compares for each protein whether the length of its sequence matches the length of the encoding CDS and throws a warning if that is not the case. Incomplete 3' or 5' CDS of the encoding transcript are the most common reasons for a mismatch between protein and transcript sequences.

Usage

```
proteinToTranscript(x, db, ...)

## S4 method for signature 'CompressedGRangesList'
proteinToTranscript(x, db, id = "name", idType = "protein_id", fiveUTR)
```

Arguments

x	IRanges with the coordinates within the protein(s). The object has also to provide some means to identify the protein (see details).
db	For the method for EnsDb objects: An EnsDb object to be used to retrieve genomic coordinates of encoding transcripts. For the method for CompressedGRangesList objects: A CompressedGRangesList object generated by cdsBy() where by = 'tx' and columns = c('tx_id' ,'protein_id','uniprot_id','protein_sequence').
...	Further arguments to be passed on.
id	character(1) specifying where the protein identifier can be found. Has to be either "name" or one of colnames(mcols(x)).
idType	character(1) defining what type of IDs are provided. Has to be one of "protein_id" (default), "uniprot_id" or "tx_id".
fiveUTR	A CompressedGRangesList object generated by fiveUTRsByTranscript().

Details

Protein identifiers (supported are Ensembl protein IDs or Uniprot IDs) can be passed to the function as names of the x IRanges object, or alternatively in any one of the metadata columns (mcols) of x.

Value

IRangesList, each element being the mapping results for one of the input ranges in x. Each element is a IRanges object with the positions within the encoding transcript (relative to the start of the transcript, which includes the 5' UTR). The transcript ID is reported as the name of each IRanges. The IRanges can be of length > 1 if the provided protein identifier is annotated to more than one Ensembl protein ID (which can be the case if Uniprot IDs are provided). If the coordinates can not be mapped (because the protein identifier is unknown to the database) an IRanges with negative coordinates is returned.

The following metadata columns are available in each IRanges in the result:

- "protein_id": the ID of the Ensembl protein for which the within-protein coordinates were mapped to the genome.
- "tx_id": the Ensembl transcript ID of the encoding transcript.
- "cds_ok": contains TRUE if the length of the CDS matches the length of the amino acid sequence and FALSE otherwise.
- "protein_start": the within-protein sequence start coordinate of the mapping.
- "protein_end": the within-protein sequence end coordinate of the mapping.

Note

While mapping of Ensembl protein IDs to Ensembl transcript IDs is 1:1, a single Uniprot identifier can be annotated to several Ensembl protein IDs. proteinToTranscript calculates in such cases transcript-relative coordinates for each annotated Ensembl protein.

Mapping using Uniprot identifiers needs also additional internal checks that can have a significant impact on the performance of the function. It is thus strongly suggested to first identify the Ensembl protein identifiers for the list of input Uniprot identifiers (e.g. using the proteins() function and use these as input for the mapping function.

Author(s)

Johannes Rainer

See Also

Other coordinate mapping functions: cdsToTranscript(), genomeToProtein(), genomeToTranscript(), proteinToGenome(), transcriptToCds(), transcriptToGenome(), transcriptToProtein()

Other coordinate mapping functions: cdsToTranscript(), genomeToProtein(), genomeToTranscript(), proteinToGenome(), transcriptToCds(), transcriptToGenome(), transcriptToProtein()

Examples

```
library(EnsDb.Hsapiens.v86)
## Restrict all further queries to chromosome x to speed up the examples
edbx <- filter(EnsDb.Hsapiens.v86, filter = ~ seq_name == "X")

## Define an IRange with protein-relative coordinates within a protein for
## the gene SYP
syp <- IRanges(start = 4, end = 17)
names(syp) <- "ENSP00000418169"
res <- proteinToTranscript(syp, edbx)
res
## Positions 4 to 17 within the protein span are encoded by the region
## from nt 23 to 64.

## Perform the mapping for multiple proteins identified by their Uniprot
## IDs.
ids <- c("O15266", "Q9HBJ8", "unexistant")
prngs <- IRanges(start = c(13, 43, 100), end = c(21, 80, 100))
names(prngs) <- ids
```

```
res <- proteinToTranscript(prngs, edbx, idType = "uniprot_id")

## The result is a list, same length as the input object
length(res)
names(res)

## No protein/encoding transcript could be found for the last one
res[[3]]

## The first protein could be mapped to multiple Ensembl proteins. The
## region within all transcripts encoding the region in the protein are
## returned
res[[1]]

## The result for the region within the second protein
res[[2]]

## Meanwhile, this function can be called in parallel processes if you preload
## the CDS data with desired data columns and fiveUTR data

cds <- cdsBy(edbx,columns = c(listColumns(edbx,'tx'),'protein_id','uniprot_id','protein_sequence'))
# cds <- cdsBy(edbx,columns = c(listColumns(edbx,'tx'),'protein_id','protein_sequence'))
# cds <- cdsBy(edbx,columns = c('tx_id','protein_id','protein_sequence'))

fiveUTR <- fiveUTRsByTranscript(edbx)

## Define an IRange with protein-relative coordinates within a protein for
## the gene SYP
syp <- IRanges(start = 4, end = 17)
names(syp) <- "ENSP00000418169"
res <- proteinToTranscript(syp, cds, fiveUTR = fiveUTR)
res
## Positions 4 to 17 within the protein span are encoded by the region
## from nt 23 to 64.

## Perform the mapping for multiple proteins identified by their Uniprot
## IDs.
ids <- c("O15266", "Q9HBJ8", "unexistant")
prngs <- IRanges(start = c(13, 43, 100), end = c(21, 80, 100))
names(prngs) <- ids

res <- proteinToTranscript(prngs, cds, idType = "uniprot_id", fiveUTR = fiveUTR)
```

---

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)

column	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 AnnotationFilter, an combination of filters AnnotationFilterList or a formula representing a filter expression (see AnnotationFilter for more details).
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 AnnotationFilter or a list of such objects. For keys: which keys should be returned from the database.
filter	For keys: either a single object extending AnnotationFilter 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" (default), "list", "filter", "asNA". See mapIds in the AnnotationDbi package 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 `AnnotationFilter` 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 `AnnotationFilter` 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`.

The use of `select` without filters or keys and without restricting to specicic columns is strongly discouraged, as the SQL query to join all of the tables, especially if protein annotation data is available is very expensive.

Author(s)

Johannes Rainer

See Also

`listColumns` `transcripts`

Examples

```
library(EnsDb.Hsapiens.v86)
edb <- EnsDb.Hsapiens.v86

## 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. Here we use
## a filter expression to define what data to retrieve.
Y <- select(edb, keys = ~ seq_name == "Y" & gene_biotype == "lincRNA",
            columns = c("GENEID", "GENEBIOTYPE", "TXID", "GENENAME"))
head(Y)
nrow(Y)
```

`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)

value	For `seqlevelsStyle<-`: a character string specifying the seqlevels style that should be set. Use the `supportedSeqlevelsStyle` to list all available and supported seqlevel styles. As an alternative, it is also possible to submit a `data.frame` with custom mapping. This needs to have two columns, one of them being called `"Ensembl"` with the original chromosome names and another column with the new names.
x	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. The function also supports using/defining custom mappings by submitting a mapping `data.frame` (see examples below).

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.v86)
edb <- EnsDb.Hsapiens.v86

## 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)

## Defining custom mapping for chromosome names. The `data.frame` should have
## one column named `"Ensembl"` with the original name and an additional column
## with the new names
mymap <- data.frame(Ensembl = c(4, 7, 9, 10),
                    myway = c("a", "b", "c", "d"))
seqlevelsStyle(edb) <- mymap
seqlevels(edb)

## This allows us also to rename individual chromosomes but keeping all
## original names for the others.
options(ensembldb.seqnameNotFound = "ORIGINAL")
seqlevels(edb)
```

---

transcriptToCds *Map transcript-relative coordinates to positions within the CDS*

Description

Converts transcript-relative coordinates to positions within the CDS (if the transcript encodes a protein).

Usage

```
transcriptToCds(x, db, id = "name", exons = NA, transcripts = NA)
```

Arguments

x	IRanges with the coordinates within the transcript. Coordinates are expected to be relative to the transcription start (the first nucleotide of the transcript). The Ensembl IDs of the corresponding transcripts have to be provided either as names of the IRanges, or in one of its metadata columns.
db	EnsDb object.
id	character(1) specifying where the transcript identifier can be found. Has to be either "name" or one of colnames(mcols(prng)).
exons	CompressedGRangesList object generated by exonsBy() where by = 'tx'.
transcripts	GRanges object generated by transcripts().

Value

IRanges with the same length (and order) than the input IRanges x. Each element in IRanges provides the coordinates within the transcripts CDS. The transcript-relative coordinates are provided as metadata columns. IRanges with a start coordinate of -1 is returned for transcripts that are not known in the database, non-coding transcripts or if the provided start and/or end coordinates are not within the coding region.

Author(s)

Johannes Rainer

See Also

Other coordinate mapping functions: cdsToTranscript(), genomeToProtein(), genomeToTranscript(), proteinToGenome(), proteinToTranscript(), transcriptToGenome(), transcriptToProtein()

Examples

```
library(EnsDb.Hsapiens.v86)
## Defining transcript-relative coordinates for 4 transcripts of the gene
## BCL2
txcoords <- IRanges(start = c(1463, 3, 143, 147), width = 1,
    names = c("ENST00000398117", "ENST00000333681",
    "ENST00000590515", "ENST00000589955"))

## Map the coordinates.
transcriptToCds(txcoords, EnsDb.Hsapiens.v86)
```

```
## ENST00000590515 does not encode a protein and thus -1 is returned
## The coordinates within ENST00000333681 are outside the CDS and thus also
## -1 is reported.

## Meanwhile, this function can be called in parallel processes if you preload
## the exons and transcripts database.

exons <- exonsBy(EnsDb.Hsapiens.v86)
transcripts <- transcripts(EnsDb.Hsapiens.v86)

transcriptToCds(txcoords, EnsDb.Hsapiens.v86, exons = exons,transcripts = transcripts)
```

`transcriptToGenome` *Map transcript-relative coordinates to genomic coordinates*

Description

`transcriptToGenome` maps transcript-relative coordinates to genomic coordinates. Provided coordinates are expected to be relative to the first nucleotide of the **transcript**, not the **CDS**. CDS-relative coordinates have to be converted to transcript-relative positions first with the `cdsToTranscript()` function.

Usage

```
transcriptToGenome(x, db, id = "name")
```

Arguments

`x`	`IRanges` with the coordinates within the transcript. Coordinates are counted from the start of the transcript (including the 5' UTR). The Ensembl IDs of the corresponding transcripts have to be provided either as `names` of the `IRanges`, or in one of its metadata columns.
`db`	`EnsDb` object or pre-loaded exons 'CompressedGRangesList' object using exonsBy().
`id`	`character(1)` specifying where the transcript identifier can be found. Has to be either `"name"` or one of `colnames(mcols(prng))`.

Value

`GRangesList` with the same length (and order) than the input `IRanges` `x`. Each `GRanges` in the `GRangesList` provides the genomic coordinates corresponding to the provided within-transcript coordinates. The original transcript ID and the transcript-relative coordinates are provided as metadata columns as well as the ID of the individual exon(s). An empty `GRanges` is returned for transcripts that can not be found in the database.

Author(s)

Johannes Rainer

See Also

cdsToTranscript() and transcriptToCds() for the mapping between CDS- and transcript-relative coordinates.

Other coordinate mapping functions: cdsToTranscript(), genomeToProtein(), genomeToTranscript(), proteinToGenome(), proteinToTranscript(), transcriptToCds(), transcriptToProtein()

Examples

```
library(EnsDb.Hsapiens.v86)
## Restrict all further queries to chromosome x to speed up the examples
edbx <- filter(EnsDb.Hsapiens.v86, filter = ~ seq_name == "X")

## Below we map positions 1 to 5 within the transcript ENST00000381578 to
## the genome. The ID of the transcript has to be provided either as names
## or in one of the IRanges' metadata columns
txpos <- IRanges(start = 1, end = 5, names = "ENST00000381578")

transcriptToGenome(txpos, edbx)
## The object returns a GRangesList with the genomic coordinates, in this
## example the coordinates are within the same exon and map to a single
## genomic region.

## Next we map nucleotides 501 to 505 of ENST00000486554 to the genome
txpos <- IRanges(start = 501, end = 505, names = "ENST00000486554")

transcriptToGenome(txpos, edbx)
## The positions within the transcript are located within two of the
## transcripts exons and thus a `GRanges` of length 2 is returned.

## Next we map multiple regions, two within the same transcript and one
## in a transcript that does not exist.
txpos <- IRanges(start = c(501, 1, 5), end = c(505, 10, 6),
    names = c("ENST00000486554", "ENST00000486554", "some"))

res <- transcriptToGenome(txpos, edbx)

## The length of the result GRangesList has the same length than the
## input IRanges
length(res)

## The result for the last region is an empty GRanges, because the
## transcript could not be found in the database
res[[3]]

res
## If you are tring to map a huge list of transcript-relative coordinates
## to genomic level, you shall use pre-loaded exons GRangesList to replace
## the SQLite db edbx

exons <- exonsBy(EnsDb.Hsapiens.v86)
```

```
## Below is just a lazy demo of querying 10^4 transcript-relative
## coordinates without any pre-splitting
library(parallel)

txpos <-  IRanges(
    start = rep(1,10),
    end = rep(30,10),
    names = c(rep('ENST00000486554',9),'some'),
    note = rep('something',10))

## only run in Linux ##
# res_temp <- mclapply(1:10, function(ind){
#     transcriptToGenome(txpos[ind], exons)
# }, mc.preschedule = TRUE, mc.cores = detectCores() - 1)

# res <- do.call(c,res_temp)
cl <- makeCluster(detectCores() - 1)
clusterExport(cl,c('transcriptToGenome','txpos','exons'))
res <- parLapply(cl,1:10,function(ind){
    transcriptToGenome(txpos[ind], exons)
})
stopCluster(cl)
```

`transcriptToProtein` *Map transcript-relative coordinates to amino acid residues of the encoded protein*

Description

`transcriptToProtein` maps within-transcript coordinates to the corresponding coordinates within the encoded protein sequence. The provided coordinates have to be within the coding region of the transcript (excluding the stop codon) but are supposed to be relative to the first nucleotide of the transcript (which includes the 5' UTR). Positions relative to the CDS of a transcript (e.g. /PKP2 c.1643delg/) have to be first converted to transcript-relative coordinates. This can be done with the `cdsToTranscript()` function.

Usage

```
transcriptToProtein(
  x,
  db,
  id = "name",
  proteins = NA,
  exons = NA,
  transcripts = NA
)
```

Arguments

x	IRanges with the coordinates within the transcript. Coordinates are counted from the start of the transcript (including the 5' UTR). The Ensembl IDs of the corresponding transcripts have to be provided either as names of the IRanges, or in one of its metadata columns.
db	EnsDb object.
id	character(1) specifying where the transcript identifier can be found. Has to be either "name" or one of colnames(mcols(prng)).
proteins	DFrame object generated by proteins().
exons	CompressedGRangesList object generated by exonsBy() where by = 'tx'.
transcripts	GRanges object generated by transcripts().

Details

Transcript-relative coordinates are mapped to the amino acid residues they encode. As an example, positions within the transcript that correspond to nucleotides 1 to 3 in the CDS are mapped to the first position in the protein sequence (see examples for more details).

Value

IRanges with the same length (and order) than the input IRanges x. Each element in IRanges provides the coordinates within the protein sequence, names being the (Ensembl) IDs of the protein. The original transcript ID and the transcript-relative coordinates are provided as metadata columns. Metadata columns "cds_ok" indicates whether the length of the transcript's CDS matches the length of the encoded protein. IRanges with a start coordinate of -1 is returned for transcript coordinates that can not be mapped to protein-relative coordinates (either no transcript was found for the provided ID, the transcript does not encode a protein or the provided coordinates are not within the coding region of the transcript).

Author(s)

Johannes Rainer

See Also

cdsToTranscript() and transcriptToCds() for conversion between CDS- and transcript-relative coordinates.

Other coordinate mapping functions: cdsToTranscript(), genomeToProtein(), genomeToTranscript(), proteinToGenome(), proteinToTranscript(), transcriptToCds(), transcriptToGenome()

Examples

```
library(EnsDb.Hsapiens.v86)
## Restrict all further queries to chromosome x to speed up the examples
edbx <- filter(EnsDb.Hsapiens.v86, filter = ~ seq_name == "X")

## Define an IRanges with the positions of the first 2 nucleotides of the
## coding region for the transcript ENST00000381578
```

```
txpos <- IRanges(start = 692, width = 2, names = "ENST00000381578")

## Map these to the corresponding residues in the protein sequence
## The protein-relative coordinates are returned as an `IRanges` object,
## with the original, transcript-relative coordinates provided in metadata
## columns tx_start and tx_end
transcriptToProtein(txpos, edbx)

## We can also map multiple ranges. Note that for any of the 3 nucleotides
## encoding the same amino acid the position of this residue in the
## protein sequence is returned. To illustrate this we map below each of the
## first 4 nucleotides of the CDS to the corresponding position within the
## protein.
txpos <- IRanges(start = c(692, 693, 694, 695),
    width = rep(1, 4), names = rep("ENST00000381578", 4))
transcriptToProtein(txpos, edbx)

## If the mapping fails, an IRanges with negative start position is returned.
## Mapping can fail (as below) because the ID is not known.
transcriptToProtein(IRanges(1, 1, names = "unknown"), edbx)

## Or because the provided coordinates are not within the CDS
transcriptToProtein(IRanges(1, 1, names = "ENST00000381578"), edbx)

## Meanwhile, this function can be called in parallel processes if you preload
## the protein, exons and transcripts database.

proteins <- proteins(edbx)
exons <- exonsBy(edbx)
transcripts <- transcripts(edbx)

txpos <- IRanges(start = c(692, 693, 694, 695),
    width = rep(1, 4),
    names = c(rep("ENST00000381578", 2), rep("ENST00000486554", 2)),
    info='test')

transcriptToProtein(txpos,edbx,proteins = proteins,exons = exons,transcripts = transcripts)
```

`useMySQL,EnsDb-method` *Use a MariaDB/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 MariaDB/MySQL server runs.
port	The port on which the MariaDB/MySQL server can be accessed.
user	The user name for the MariaDB/MySQL server.
pass	The password for the MariaDB/MySQL server.

Details

This functionality requires that the RMariaDB 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 MariaDB/MySQL database differ.

Author(s)

Johannes Rainer

Examples

```
## Load the EnsDb database (SQLite backend).
library(EnsDb.Hsapiens.v86)
edb <- EnsDb.Hsapiens.v86
## 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

∗ **classes**
EnsDb-class, 10
exonsBy, 13
getGeneRegionTrackForGviz, 31
getGenomeFaFile, 33
lengthOf, 35
select, 51
seqlevelsStyle, 54
∗ **coordinate mapping functions**
cdsToTranscript, 4
genomeToProtein, 27
genomeToTranscript, 29
proteinToGenome, 44
proteinToTranscript, 47
transcriptToCds, 56
transcriptToGenome, 58
transcriptToProtein, 60
∗ **data**
makeEnsembldbPackage, 38
runEnsDbApp, 50
∗ **shiny**
runEnsDbApp, 50

activeFilter *(*addFilter,EnsDb-method*)*, 3
activeFilter,EnsDb-method *(*addFilter,EnsDb-method*)*, 3
addFilter, *12*, *14*, *19*
addFilter *(*addFilter,EnsDb-method*)*, 3
addFilter(), *22*, *25*
addFilter,EnsDb-method, 3
AnnotationFilter, *3*, *15*, *22*, *32*, *36*, *52*
AnnotationFilterList, *3*, *15*, *32*, *36*, *52*

cdsBy *(*exonsBy*)*, 13
cdsBy(), *24*, *45*, *48*
cdsBy,EnsDb-method *(*exonsBy*)*, 13
cdsToTranscript, 4, *28*, *30*, *46*, *49*, *57*, *59*, *61*
cdsToTranscript(), *58–61*
columns,EnsDb-method *(*select*)*, 51
convertFilter,AnnotationFilter,EnsDb-method, 6
convertFilter,AnnotationFilterList,EnsDb-method *(*convertFilter,AnnotationFilter,EnsDb-method*)*, 6

dbconn *(*EnsDb-class*)*, 10
dbconn,EnsDb-method *(*EnsDb-class*)*, 10
Deprecated, 7
dropFilter *(*addFilter,EnsDb-method*)*, 3
dropFilter,EnsDb-method *(*addFilter,EnsDb-method*)*, 3

EnsDb, *3*, *8*, 8, *9*, *11*, *12*, *22*, *35*, *41–43*, *51*, *56*, *62*, *63*
EnsDb-class, 10
ensDbFromAH *(*makeEnsembldbPackage*)*, 38
ensDbFromGff *(*makeEnsembldbPackage*)*, 38
ensDbFromGRanges *(*makeEnsembldbPackage*)*, 38
ensDbFromGtf *(*makeEnsembldbPackage*)*, 38
ensembldb-deprecated *(*Deprecated*)*, 7
ensemblVersion *(*EnsDb-class*)*, 10
ensemblVersion,EnsDb-method *(*EnsDb-class*)*, 10
EntrezidFilter *(*Deprecated*)*, 7
exonicParts, *17*
ExonidFilter *(*Deprecated*)*, 7
ExonrankFilter *(*Deprecated*)*, 7
exons *(*exonsBy*)*, 13
exons(), *24*, *25*
exons,EnsDb-method *(*exonsBy*)*, 13
exonsBy, *12*, 13, *34*, *36*
exonsBy(), *5*, *24*, *27*, *57*, *61*
exonsBy,EnsDb-method *(*exonsBy*)*, 13
exonsByOverlaps,EnsDb-method *(*exonsBy*)*, 13

fetchTablesFromEnsembl *(*makeEnsembldbPackage*)*, 38

filter (addFilter,EnsDb-method), 3
filter(), *25*
Filter-classes, 22
fiveUTRsByTranscript(), *48*
fiveUTRsByTranscript,EnsDb-method (exonsBy), 13

GenebiotypeFilter (Deprecated), 7
GeneidFilter (Deprecated), 7
GeneIdFilter(), *25*
genes, *3*, *12*, *41*, *43*, *51*, *52*
genes (exonsBy), 13
genes(), *22*, *24*, *25*
genes,EnsDb-method (exonsBy), 13
genomeToProtein, *5*, 27, *30*, *46*, *49*, *57*, *59*, *61*
genomeToTranscript, *5*, *28*, 29, *46*, *49*, *57*, *59*, *61*
genomeToTranscript(), *27*, *28*
getGeneRegionTrackForGviz, 31
getGeneRegionTrackForGviz,EnsDb-method (getGeneRegionTrackForGviz), 31
getGenomeFaFile, 33
getGenomeFaFile,EnsDb-method (getGenomeFaFile), 33
getGenomeTwoBitFile (getGenomeFaFile), 33
getGenomeTwoBitFile,EnsDb-method (getGenomeFaFile), 33
GRangesFilter, *18*

hasProteinData (hasProteinData,EnsDb-method), 34
hasProteinData(), *24*
hasProteinData,EnsDb-method, 34

intronicParts, *17*
intronsByTranscript,EnsDb-method (exonsBy), 13

keys,EnsDb-method (select), 51
keytypes,EnsDb-method (select), 51

lengthOf, *19*, 35
lengthOf,EnsDb-method (lengthOf), 35
lengthOf,GRangesList-method (lengthOf), 35
listColumns, *15*, *19*, *43*, *52*, *53*
listColumns (EnsDb-class), 10
listColumns,EnsDb-method (EnsDb-class), 10
listEnsDbs, 37
listGenebiotypes (EnsDb-class), 10
listGenebiotypes(), *25*
listGenebiotypes,EnsDb-method (EnsDb-class), 10
listProteinColumns, *17*
listProteinColumns (proteins,EnsDb-method), 42
listTables, *15*, *35*
listTables (EnsDb-class), 10
listTables,EnsDb-method (EnsDb-class), 10
listTxbiotypes (EnsDb-class), 10
listTxbiotypes(), *25*
listTxbiotypes,EnsDb-method (EnsDb-class), 10
listUniprotDbs (proteins,EnsDb-method), 42
listUniprotDbs(), *25*
listUniprotDbs,EnsDb-method (proteins,EnsDb-method), 42
listUniprotMappingTypes (proteins,EnsDb-method), 42
listUniprotMappingTypes(), *25*
listUniprotMappingTypes,EnsDb-method (proteins,EnsDb-method), 42

makeEnsembldbPackage, *11*, *12*, *14*, *19*, 38
makeEnsemblSQLiteFromTables, *11*, *12*
makeEnsemblSQLiteFromTables (makeEnsembldbPackage), 38
mapIds,EnsDb-method (select), 51
metadata (EnsDb-class), 10
metadata,EnsDb-method (EnsDb-class), 10

OnlyCodingTxFilter (Filter-classes), 22
OnlyCodingTxFilter-class (Filter-classes), 22
organism (EnsDb-class), 10
organism,EnsDb-method (EnsDb-class), 10

promoters (exonsBy), 13
promoters,EnsDb-method (exonsBy), 13
ProtDomIdFilter (Filter-classes), 22
ProtDomIdFilter-class (Filter-classes), 22

ProteinDomainIdFilter *(Filter-classes)*, 22
ProteinDomainIdFilter-class *(Filter-classes)*, 22
ProteinDomainSourceFilter *(Filter-classes)*, 22
ProteinDomainSourceFilter-class *(Filter-classes)*, 22
proteins *(proteins,EnsDb-method)*, 42
proteins(), *27*, *46*, *49*, *61*
proteins,EnsDb-method, 42
proteinToGenome, *5*, *28*, *30*, 44, *46*, *49*, *57*, *59*, *61*
proteinToGenome(), *48*
proteinToGenome,CompressedGRangesList-method *(proteinToGenome)*, 44
proteinToGenome,EnsDb-method *(proteinToGenome)*, 44
proteinToGenome,Preloaded-method *(proteinToGenome)*, 44
proteinToTranscript, *5*, *28*, *30*, *46*, 47, *57*, *59*, *61*
proteinToTranscript,CompressedGRangesList-method *(proteinToTranscript)*, 47
proteinToTranscript,EnsDb-method *(proteinToTranscript)*, 47
proteinToTranscript,Preloaded-method *(proteinToTranscript)*, 47

returnFilterColumns *(EnsDb-class)*, 10
returnFilterColumns,EnsDb-method *(EnsDb-class)*, 10
returnFilterColumns<- *(EnsDb-class)*, 10
returnFilterColumns<-,EnsDb-method *(EnsDb-class)*, 10
runApp, *51*
runEnsDbApp, 50

select, 51
select,EnsDb-method *(select)*, 51
SeqendFilter *(Deprecated)*, 7
seqinfo *(EnsDb-class)*, 10
seqinfo,EnsDb-method *(EnsDb-class)*, 10
seqlevels *(EnsDb-class)*, 10
seqlevels,EnsDb-method *(EnsDb-class)*, 10
seqlevels,GRangesFilter-method *(Filter-classes)*, 22
seqlevelsStyle, 54
seqlevelsStyle(), *24*
seqlevelsStyle,EnsDb-method *(seqlevelsStyle)*, 54
seqlevelsStyle<- *(seqlevelsStyle)*, 54
seqlevelsStyle<-,EnsDb-method *(seqlevelsStyle)*, 54
SeqnameFilter *(Deprecated)*, 7
seqnames,GRangesFilter-method *(Filter-classes)*, 22
SeqstartFilter *(Deprecated)*, 7
SeqstrandFilter *(Deprecated)*, 7
show *(EnsDb-class)*, 10
show,EnsDb-method *(EnsDb-class)*, 10
supportedFilters, *15*, *19*, *43*
supportedFilters(), *22*, *25*
supportedFilters,EnsDb-method *(Filter-classes)*, 22
supportedSeqlevelsStyles *(seqlevelsStyle)*, 54
supportedSeqlevelsStyles,EnsDb-method *(seqlevelsStyle)*, 54

threeUTRsByTranscript,EnsDb-method *(exonsBy)*, 13
toSAF *(exonsBy)*, 13
toSAF,GRangesList-method *(exonsBy)*, 13
transcriptLengths, *36*
transcripts, *12*, *32*, *34*, *36*, *43*, *52*, *53*, *56*
transcripts *(exonsBy)*, 13
transcripts(), *5*, *24*, *25*, *27*, *57*, *61*
transcripts,EnsDb-method *(exonsBy)*, 13
transcriptsBy, *15*
transcriptsBy *(exonsBy)*, 13
transcriptsBy(), *24*
transcriptsBy,EnsDb-method *(exonsBy)*, 13
transcriptsByOverlaps, *18*
transcriptsByOverlaps,EnsDb-method *(exonsBy)*, 13
transcriptToCds, *5*, *28*, *30*, *46*, *49*, 56, *59*, *61*
transcriptToCds(), *59*, *61*
transcriptToGenome, *5*, *28*, *30*, *46*, *49*, *57*, 58, *61*
transcriptToProtein, *5*, *28*, *30*, *46*, *49*, *57*, *59*, 60
transcriptToProtein(), *27*, *28*
TxbiotypeFilter *(Deprecated)*, 7
TxExternalNameFilter *(Filter-classes)*, 22
TxExternalNameFilter-class *(Filter-classes)*, 22

TxidFilter (Deprecated), 7
TxIsCanonicalFilter (Filter-classes), 22
TxIsCanonicalFilter-class
        (Filter-classes), 22
TxSupportLevelFilter (Filter-classes),
        22
TxSupportLevelFilter-class
        (Filter-classes), 22

UniprotDbFilter (Filter-classes), 22
UniprotDbFilter-class (Filter-classes),
        22
UniprotMappingTypeFilter
        (Filter-classes), 22
UniprotMappingTypeFilter-class
        (Filter-classes), 22
updateEnsDb (EnsDb-class), 10
updateEnsDb,EnsDb-method (EnsDb-class),
        10
useMySQL, *9*, *37, 38*
useMySQL (useMySQL,EnsDb-method), 62
useMySQL,EnsDb-method, 62