Package 'OrganismDbi'

October 20, 2025

Title Software to enable the smooth interfacing of different database packages

Description The package enables a simple unified interface to several annotation packages each of which has its own schema by taking advantage of the fact that each of these packages implements a select methods.

Version 1.51.5

Encoding UTF-8

Depends R (>= 2.14.0), BiocGenerics (>= 0.15.10), AnnotationDbi (>= 1.33.15), Seqinfo, GenomicFeatures (>= 1.61.4)

Imports methods, utils, stats, DBI, BiocManager, Biobase, graph, RBGL, S4Vectors, IRanges, GenomicRanges (>= 1.61.1)

Suggests txdbmaker, GenomeInfoDbData, Homo.sapiens, Rattus.norvegicus, BSgenome.Hsapiens.UCSC.hg19, AnnotationHub, FDb.UCSC.tRNAs, rtracklayer, biomaRt, RUnit, RMariaDB, BiocStyle, knitr

Collate AllGenerics.R AllClasses.R methods-select.R methods-transcripts.R createOrganismPackage.R seqinfo.R test_OrganismDbi_package.R

License Artistic-2.0

biocViews Annotation, Infrastructure

VignetteBuilder knitr

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

git_branch devel

git_last_commit fb8673c

git_last_commit_date 2025-10-16

Repository Bioconductor 3.22

Date/Publication 2025-10-19

Author Marc Carlson [aut],
Martin Morgan [aut],
Valerie Obenchain [aut],
Aliyu Atiku Mustapha [ctb] (Converted 'OrganismDbi' vignette from Sweave to RMarkdown / HTML.),
Bioconductor Package Maintainer [cre]

Maintainer Bioconductor Package Maintainer <maintainer@bioconductor.org>

Contents

`makeOrganismDbFromBiomart` *Make a OrganismDb object from annotations available on a BioMart database*

Description

The `makeOrganismDbFromBiomart` function allows the user to make a OrganismDb object from transcript annotations available on a BioMart database. This object has all the benefits of a TxDb, plus an associated OrgDb and GODb object.

Usage

```
makeOrganismDbFromBiomart(biomart="ENSEMBL_MART_ENSEMBL",
                    dataset="hsapiens_gene_ensembl",
                    transcript_ids=NULL,
                    circ_seqs=NULL,
                    filter="",
                    id_prefix="ensembl_",
                    host="https://www.ensembl.org",
                    port,
                    miRBaseBuild=NA,
                    keytype = "ENSEMBL",
                    orgdb = NA)
```

Arguments

`biomart`	which BioMart database to use. Get the list of all available BioMart databases with the `listMarts` function from the biomaRt package. See the details section below for a list of BioMart databases with compatible transcript annotations.
`dataset`	which dataset from BioMart. For example: `"hsapiens_gene_ensembl"`, `"mmusculus_gene_ensemb` `"dmelanogaster_gene_ensembl"`, `"celegans_gene_ensembl"`, `"scerevisiae_gene_ensembl"`, etc in the ensembl database. See the examples section below for how to discover which datasets are available in a given BioMart database.
`transcript_ids`	optionally, only retrieve transcript annotation data for the specified set of transcript ids. If this is used, then the meta information displayed for the resulting TxDb object will say 'Full dataset: no'. Otherwise it will say 'Full dataset: yes'. This TxDb object will be embedded in the resulting OrganismDb object.

circ_seqs	a character vector to list out which chromosomes should be marked as circular.
filter	Additional filters to use in the BioMart query. Must be a named list. An example is filter=as.list(c(source="entrez"))
host	The host URL of the BioMart. Defaults to www.ensembl.org.
port	Deprecated: The port to use in the HTTP communication with the host.
id_prefix	Specifies the prefix used in BioMart attributes. For example, some BioMarts may have an attribute specified as "ensembl_transcript_id" whereas others have the same attribute specified as "transcript_id". Defaults to "ensembl_".
miRBaseBuild	This argument is defunct.
keytype	This indicates the kind of key that this database will use as a foreign key between it's TxDb object and it's OrgDb object. So basically whatever the column name is for the foreign key from your OrgDb that your TxDb will need to map it's GENEID on to. By default it is "ENSEMBL" since the GENEID's for most biomaRt based TxDbs will be ensembl gene ids and therefore they will need to map to ENSEMBL gene mappings from the associated OrgDb object.
orgdb	By default, makeOrganismDbFromBiomart will use the taxonomyID from your txdb to lookup an appropriate matching OrgDb object but using this you can supply a different OrgDb object.

Details

makeOrganismDbFromBiomart is a convenience function that feeds data from a BioMart database to the lower level OrganismDb constructor. See ?makeOrganismDbFromUCSC for a similar function that feeds data from the UCSC source.

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

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

Not all these datasets have CDS information.

Value

A OrganismDb object.

Author(s)

M. Carlson

See Also

- makeOrganismDbFromUCSC for convenient ways to make a OrganismDb object from UCSC online resources.
- The listMarts, useMart, and listDatasets functions in the **biomaRt** package.
- The OrganismDb class.

Examples

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

## Retrieving an incomplete transcript dataset for Human from the
## "ensembl" BioMart database:
transcript_ids <- c(
    "ENST00000013894",
    "ENST00000268655",
    "ENST00000313243",
    "ENST00000435657",
    "ENST00000384428",
    "ENST00000478783"
)
odb <- makeOrganismDbFromBiomart(transcript_ids=transcript_ids)
odb  # note that these annotations match the GRCh38 genome assembly

if (interactive()) {
  ## Now what if we want to use another mirror?  We might make use of the
  ## new host argument.  But wait!  If we use biomaRt, we can see that
  ## this host has named the mart differently!
  listMarts(host="https://useast.ensembl.org")

  ## Therefore we must also change the name passed into the "mart"
  ## argument thusly:
  makeOrganismDbFromBiomart(
      biomart="ENSEMBL_MART_ENSEMBL",
      transcript_ids=transcript_ids,
      host="https://useast.ensembl.org"
  )
}
```

---

makeOrganismDbFromTxDb *Make an OrganismDb object from an existing TxDb object.*

Description

The makeOrganismDbFromTxDb function allows the user to make a OrganismDb object from an existing TxDb object.

Usage

```
makeOrganismDbFromTxDb(txdb, keytype=NA, orgdb=NA)
```

Arguments

txdb	a TxDb object
.	
keytype	By default, makeOrganismDbFromTxDb will try to guess this information based on the OrgDb object that is inferred to go with your TxDb object... But in some instances, you may need to supply an over-ride and that is what this argument is for. It is the column name of the ID type that your OrgDb will use as a foreign key when connecting to the data from the associated TxDb. So for example, if you looked at the Homo.sapiens package the keytype for org.Hs.eg.db, would be 'ENTREZID' because that is the kind of ID that matches up with it's TxDb GENEID. (Because the GENEID for that specific TxDb is from UCSC and uses entrez gene IDs)
.	
orgdb	By default, makeOrganismDbFromTxDb will use the taxonomyID from your txdb to lookup an appropriate matching OrgDb object but using this you can supply a different OrgDb object.

Details

makeOrganismDbFromTxDb is a convenience function that processes a TxDb object and pairs it up with GO.db and an appropriate OrgDb object to make a OrganismDb object. See ?makeOrganismDbFromBiomart and ?makeOrganismDbFromUCSC for a similar function that feeds data from either a BioMart or UCSC.

Value

A OrganismDb object.

Author(s)

M. Carlson

See Also

- makeOrganismDbFromBiomart for convenient ways to make a OrganismDb object from BioMart online resources.
- The OrganismDb class.

Examples

```
## Not run:
library(txdbmaker)  # for makeTxDbFromUCSC()
## lets start with a txdb object
transcript_ids <- c(
    "uc009uzf.1",
    "uc009uzg.1",
    "uc009uzh.1",
```

```
    "uc009uzi.1",
    "uc009uzj.1"
)
txdbMouse <- makeTxDbFromUCSC(genome="mm9", tablename="knownGene",
                              transcript_ids=transcript_ids)

## Using that, we can call our function to promote it to an OrgDb object:
odb <- makeOrganismDbFromTxDb(txdb=txdbMouse)

columns(odb)

## End(Not run)
```

`makeOrganismDbFromUCSC` *Make a OrganismDb object from annotations available at the UCSC Genome Browser*

Description

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

Usage

```
makeOrganismDbFromUCSC(
        genome="hg19",
        tablename="knownGene",
        transcript_ids=NULL,
        circ_seqs=NULL,
        url="http://genome.ucsc.edu/cgi-bin/",
        goldenPath.url=getOption("UCSC.goldenPath.url"),
        miRBaseBuild=NA)
```

Arguments

`genome`	genome abbreviation used by UCSC and obtained by `ucscGenomes()[ , "db"]`. For example: `"hg19"`.
`tablename`	name of the UCSC table containing the transcript annotations to retrieve. Use the `supportedUCSCtables` utility function to get the list of supported tables. Note that not all tables are available for all genomes.
`transcript_ids`	optionally, only retrieve transcript annotation data for the specified set of transcript ids. If this is used, then the meta information displayed for the resulting OrganismDb object will say 'Full dataset: no'. Otherwise it will say 'Full dataset: yes'.
`circ_seqs`	a character vector to list out which chromosomes should be marked as circular.
`url`	Deprecated (will be ignored).
`goldenPath.url`	use to specify the location of an alternate UCSC Genome Browser.
`miRBaseBuild`	This argument is defunct.

Details

makeOrganismDbFromUCSC is a convenience function that feeds data from the UCSC source to the lower level OrganismDb function. See ?makeOrganismDbFromBiomart for a similar function that feeds data from a BioMart database.

Value

A OrganismDb object.

Author(s)

M. Carlson

See Also

- makeOrganismDbFromBiomart for convenient ways to make a OrganismDb object from BioMart online resources.
- ucscGenomes in the **rtracklayer** package.
- The OrganismDb class.

Examples

```
## Not run:
## Display the list of genomes available at UCSC:
library(rtracklayer)
library(RMariaDB)
ucscGenomes()[ , "db"]

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

\dontrun{
## Retrieving a full transcript dataset for Yeast from UCSC:
odb1 <- makeOrganismDbFromUCSC(genome="sacCer2", tablename="ensGene")
}

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

odb2 <- makeOrganismDbFromUCSC(genome="mm9", tablename="knownGene",
                           transcript_ids=transcript_ids)
odb2

## End(Not run)
```

makeOrganismPackage *Making OrganismDb packages from annotation packages.*

Description

makeOrganismPackage is a method that generates a package that will load an appropriate annotationOrganismDb object that will in turn point to existing annotation packages.

Usage

```
makeOrganismPackage (pkgname,
                     graphData,
                     organism,
                     version,
                     maintainer,
                     author,
                     destDir,
                     license="Artistic-2.0")
```

Arguments

pkgname	What is the desired package name. Traditionally, this should be the genus and species separated by a ".". So as an example, "Homo.sapiens" would be the package name for human
graphData	A list of short character vectors. Each character vector in the list is exactly two elements long and represents a join relationship between two packages. The names of these character vectors are the package names and the values are the foreign keys that should be used to connect each package. All foreign keys must be values that can be returned by the columns method for each package in question, and obviously they also must be the same kind of identifier as well.
organism	The name of the organism this package represents
version	What is the version number for this package?
maintainer	Who is the package maintainer? (must include email to be valid)
author	Who is the creator of this package?
destDir	A path where the package source should be assembled.
license	What is the license (and it's version)

Details

The purpose of this method is to create a special package that will depend on existing annotation packages and which will load a special annotationOrganismDb object that will allow proper dispatch of special select methods. These methods will allow the user to easily query across multiple annotation resources via information contained by the annotationOrganismDb object. Because the end result will be a package that treats all the data mapped together as a single source, the user is encouraged to take extra care to ensure that the different packages used are from the same build etc.

Value

A special package to load an OrganismDb object.

Author(s)

M. Carlson

See Also

OrganismDb

Examples

```
## set up the list with the relevant relationships:
gd <- list(join1 = c(GO.db="GOID", org.Hs.eg.db="GO"),
           join2 = c(org.Hs.eg.db="ENTREZID",
                     TxDb.Hsapiens.UCSC.hg19.knownGene="GENEID"))

## sets up a temporary directory for this example
## (users won't need to do this step)
destination <- tempfile()
dir.create(destination)

## makes an Organism package for human called Homo.sapiens
if(interactive()){
  makeOrganismPackage(pkgname = "Homo.sapiens",
   graphData = gd,
   organism = "Homo sapiens",
   version = "1.0.0",
   maintainer = "Bioconductor Package Maintainer <maintainer@bioconductor.org>",
   author = "Bioconductor Core Team",
   destDir = destination,
   license = "Artistic-2.0")
}
```

---

mapToTranscripts *Map range coordinates between transcripts and genome space*

Description

Map range coordinates between features in the transcriptome and genome (reference) space.

See `mapToAlignments` in the **GenomicAlignments** package for mapping coordinates between reads (local) and genome (reference) space using a CIGAR alignment.

Usage

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

Arguments

`x`	GRanges-class object of positions to be mapped. `x` must have names when mapping to the genome.
`transcripts`	The `OrganismDb` object that will be used to extract features using the `extractor.fun`.

`ignore.strand` When TRUE, strand is ignored in overlap operations.

`extractor.fun` Function to extract genomic features from a `TxDb` object.

Valid `extractor` functions:

- transcripts ## default
- exons
- cds
- genes
- promoters
- tRNAs
- transcriptsBy
- exonsBy
- cdsBy
- intronsByTranscript
- fiveUTRsByTranscript
- threeUTRsByTranscript

`...` Additional arguments passed to `extractor.fun` functions.

Details

- `mapToTranscripts` The genomic range in `x` is mapped to the local position in the `transcripts` ranges. A successful mapping occurs when `x` is completely within the `transcripts` range, equivalent to:

```
findOverlaps(..., type="within")
```

  Transcriptome-based coordinates start counting at 1 at the beginning of the `transcripts` range and return positions where `x` was aligned. The seqlevels of the return object are taken from the `transcripts` object and should be transcript names. In this direction, mapping is attempted between all elements of `x` and all elements of `transcripts`.

Value

An object the same class as `x`.

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

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

When present, names from `x` are propagated to the output. When mapping to transcript coordinates, seqlevels of the output are the names on the `transcripts` object; most often these will be transcript names. When mapping to the genome, seqlevels of the output are the seqlevels of `transcripts` which are usually chromosome names.

Author(s)

V. Obenchain, M. Lawrence and H. Pagès; ported to work with OrganismDbi by Marc Carlson

See Also

- mapToTranscripts.

Examples

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

library(Homo.sapiens)
x <- GRanges("chr5",
             IRanges(c(173315331,174151575), width=400,
                     names=LETTERS[1:2]))

## Map to transcript coordinates:
mapToTranscripts(x, Homo.sapiens)
```

`MultiDb-class` *MultiDb and OrganismDb objects*

Description

The OrganismDb class is a container for storing knowledge about existing Annotation packages and the relationships between these resources. The purpose of this object and it's associated methods is to provide a means by which users can conveniently query for data from several different annotation resources at the same time using a familiar interface.

The supporting methods `select`, `columns` and `keys` are used together to extract data from an `OrganismDb` object in a manner that should be consistent with how these are used on the supporting annotation resources.

The family of `seqinfo` style getters (`seqinfo`, `seqlevels`, `seqlengths`, `isCircular`, `genome`, and `seqnameStyle`) is also supported for OrganismDb objects provided that the object in question has an embedded TxDb object.

Methods

In the code snippets below, `x` is a OrganismDb object.

`keytypes(x)`**:** allows the user to discover which keytypes can be passed in to `select` or `keys` and the `keytype` argument.

`keys(x, keytype, pattern, column, fuzzy)`**:** Return keys for the database contained in the TxDb object .

The `keytype` argument specifies the kind of keys that will be returned and is always required.

If `keys` is used with `pattern`, it will pattern match on the `keytype`.

But if the `column` argument is also provided along with the `pattern` argument, then `pattern` will be matched against the values in `column` instead.

If `keys` is called with `column` and no `pattern` argument, then it will return all keys that have corresponding values in the `column` argument.

Thus, the behavior of `keys` all depends on how many arguments are specified.

Use of the `fuzzy` argument will toggle fuzzy matching to TRUE or FALSE. If `pattern` is not used, fuzzy is ignored.

`columns(x)`: shows which kinds of data can be returned for the `OrganismDb` object.

`select(x, keys, columns, keytype)`: When all the appropriate arguments are specifiedm `select` will retrieve the matching data as a data.frame based on parameters for selected `keys` and `columns` and `keytype` arguments.

`mapIds(x, keys, columns, keytype, ..., multiVals)`: When all the appropriate arguments are specifiedm `mapIds` will retrieve the matching data as a vector or list based on parameters for selected `keys` and `columns` and `keytype` arguments. The multiVals argument can be used to choose the format of the values returned. Possible values for multiVals are:

first: This value means that when there are multiple matches only the 1st thing that comes back will be returned. This is the default behavior

list: This will just returns a list object to the end user

filter: This will remove all elements that contain multiple matches and will therefore return a shorter vector than what came in whenever some of the keys match more than one value

asNA: This will return an NA value whenever there are multiple matches

CharacterList: This just returns a SimpleCharacterList object

FUN: You can also supply a function to the `multiVals` argument for custom behaviors. The function must take a single argument and return a single value. This function will be applied to all the elements and will serve a 'rule' that for which thing to keep when there is more than one element. So for example this example function will always grab the last element in each result: `last <- function(x){x[[length(x)]]}`

`selectByRanges(x, ranges, columns, overlaps, ignore.strand)`: When all the appropriate arguments are specified, `selectByRanges` will return an annotated GRanges object that has been generated based on what you passed in to the ranges argument and whether that overlapped with what you specified in the overlaps argument. Internally this function will get annotation features and overlaps by calling the appropriate annotation methods indicated by the overlaps argument. The value for overlaps can be any of: gene, tx, exons, cds, 5utr, introns or 3utr. The default value is 'tx' which will return to you, your annotated ranges based on whether the overlapped with the transcript ranges of any gene in the associated TxDb object based on the gene models it contains. Also: the number of ranges returned to you will match the number of genes that your ranges argument overlapped for the type of overlap that you specified. So if some of your ranges are large and overlap several features then you will get many duplicated ranges returned with one for each gene that has an overlapping feature. The columns values that you request will be returned in the mcols for the annotated GRanges object that is the return value for this function. Finally, the ignore.strand argument is provided to indicate whether or not findOverlaps should ignore or respect the strand.

`selectRangesById(x, keys, columns, keytype, feature)`: When all the appropriate arguments are specified, `selectRangesById` will return a GRangesList object that correspond to gene models GRanges for the keys that you specify with the keys and keytype arguments. The annotation ranges retrieved for this will be specified by the feature argument and can be: gene, tx, exon or cds. The default is 'tx' which will return the transcript ranges for each gene as a GRanges object in the list. Extra data can also be returned in the mcols values for those GRanges by using the columns argument.

`resources(x)`: shows where the db files are for resources that are used to store the data for the `OrganismDb` object.

`TxDb(x)`: Accessor for the TxDb object of a `OrganismDb` object.

`TxDb(x) <- value`: Allows you to swap in an alternative TxDb for a given `OrganismDb` object. This is most often useful when combined with `saveDb(TxDb, file)`, which returns the saved TxDb, so that you can save a TxDb to disc and then assign the saved version right into your `OrganismDb` object.

Author(s)

Marc Carlson

See Also

- AnnotationDb-class for more descriptsion of methods `select`,`keytypes`,`keys` and `columns`.
- makeOrganismPackage for functions used to generate an `OrganismDb` based package.
- rangeBasedAccessors for the range based methods used in extracting data from a `OrganismDb` object.
- Topics in the `Seqinfo` package:
  - `seqinfo`
  - `seqlevels`
  - `seqlengths`
  - `isCircular`
  - `genome`

Examples

```
## load a package that creates an OrganismDb
library(Homo.sapiens)
ls(2)
## then the methods can be used on this object.
columns <- columns(Homo.sapiens)[c(7,10,11,12)]
keys <- head(keys(org.Hs.eg.db, "ENTREZID"))
keytype <- "ENTREZID"
res <- select(Homo.sapiens, keys, columns, keytype)
head(res)
res <-  mapIds(Homo.sapiens, keys=c('1','10'), column='ALIAS',
               keytype='ENTREZID',  multiVals="CharacterList")

## get symbols for ranges in question:
ranges <-  GRanges(seqnames=Rle(c('chr11'), c(2)),
                   IRanges(start=c(107899550, 108025550),
                           end=c(108291889, 108050000)), strand='*',
                   seqinfo=seqinfo(Homo.sapiens))
selectByRanges(Homo.sapiens, ranges, 'SYMBOL')

## Or extract the gene model for the 'A1BG' gene:
selectRangesById(Homo.sapiens, 'A1BG', keytype='SYMBOL')


## Get the DB connections or DB file paths associated with those for
## each.
dbconn(Homo.sapiens)
dbfile(Homo.sapiens)

## extract the taxonomyId
taxonomyId(Homo.sapiens)

##extract the resources
resources(Homo.sapiens)
```

rangeBasedAccessors *Extract genomic features from an object*

Description

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

Usage

```
## S4 method for signature 'MultiDb'
transcripts(x, columns=c("TXID", "TXNAME"), filter=NULL)

## S4 method for signature 'MultiDb'
exons(x, columns="EXONID", filter=NULL)

## S4 method for signature 'MultiDb'
cds(x, columns="CDSID", filter=NULL)

## S4 method for signature 'MultiDb'
genes(x, columns="GENEID", filter=NULL)

## S4 method for signature 'MultiDb'
transcriptsBy(x, by, columns, use.names=FALSE,
                                    outerMcols=FALSE)

## S4 method for signature 'MultiDb'
exonsBy(x, by, columns, use.names=FALSE, outerMcols=FALSE)

## S4 method for signature 'MultiDb'
cdsBy(x, by, columns, use.names=FALSE, outerMcols=FALSE)

## S4 method for signature 'MultiDb'
getTxDbIfAvailable(x, ...)

## S4 method for signature 'MultiDb'
asBED(x)
## S4 method for signature 'MultiDb'
asGFF(x)

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

## S4 method for signature 'GenomicRanges,MultiDb'
distance(x, y, ignore.strand=FALSE,
    ..., id, type=c("gene", "tx", "exon", "cds"))

## S4 method for signature 'BSgenome'
extractTranscriptSeqs(x, transcripts, strand = "+")
```

```
## S4 method for signature 'MultiDb'
extractUpstreamSeqs(x, genes, width=1000, exclude.seqlevels=NULL)

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

## S4 method for signature 'MultiDb'
isActiveSeq(x)
```

Arguments

x	A MultiDb object, except in the extractTranscriptSeqs method where it is a BSgenome object and the second argument is a MultiDb object.
...	Arguments to be passed to or from methods.
by	One of "gene", "exon", "cds" or "tx". Determines the grouping.
columns	The columns or kinds of metadata that can be retrieved from the database. All possible columns are returned by using the columns method.
filter	Either NULL or a named list of vectors to be used to restrict the output. Valid names for this list are: "gene_id", "tx_id", "tx_name", "tx_chrom", "tx_strand", "exon_id", "exon_name", "exon_chrom", "exon_strand", "cds_id", "cds_name", "cds_chrom", "cds_strand" and "exon_rank".
use.names	Controls how to set the names of the returned GRangesList object. These functions return all the features of a given type (e.g. all the exons) grouped by another feature type (e.g. grouped by transcript) in a GRangesList object. By default (i.e. if use.names is FALSE), the names of this GRangesList object (aka the group names) are the internal ids of the features used for grouping (aka the grouping features), which are guaranteed to be unique. If use.names is TRUE, then the names of the grouping features are used instead of their internal ids. For example, when grouping by transcript (by="tx"), the default group names are the transcript internal ids ("tx_id"). But, if use.names=TRUE, the group names are the transcript names ("tx_name"). Note that, unlike the feature ids, the feature names are not guaranteed to be unique or even defined (they could be all NAs). A warning is issued when this happens. See ?id2name for more information about feature internal ids and feature external names and how to map the formers to the latters. Finally, use.names=TRUE cannot be used when grouping by gene by="gene". This is because, unlike for the other features, the gene ids are external ids (e.g. Entrez Gene or Ensembl ids) so the db doesn't have a "gene_name" column for storing alternate gene names.
upstream	For promoters : An integer(1) value indicating the number of bases upstream from the transcription start site. For additional details see ?`promoters,GRanges-method`.
downstream	For promoters : An integer(1) value indicating the number of bases downstream from the transcription start site. For additional details see ?`promoters,GRanges-method`.
y	For distance, a MultiDb instance. The id is used to extract ranges from the MultiDb which are then used to compute the distance from x.

`id` A `character` vector the same length as `x`. The `id` must be identifiers in the MultiDb object. `type` indicates what type of identifier `id` is.

`type` A `character(1)` describing the `id`. Must be one of 'gene', 'tx', 'exon' or 'cds'.

`ignore.strand` A `logical` indicating if the strand of the ranges should be ignored. When `TRUE`, strand is set to `'+'`.

`outerMcols` A `logical` indicating if the the 'outer' mcols (metadata columns) should be populated for some range based accesors which return a GRangesList object. By default this is FALSE, but if TRUE then the outer list object will also have it's metadata columns (mcols) populated as well as the mcols for the 'inner' GRanges objects.

`transcripts` An object representing the exon ranges of each transcript to extract. It must be a GRangesList or MultiDb object while the `x` is a `BSgenome` object. Internally, it's turned into a GRangesList object with `exonsBy(transcripts, by="tx", use.names=TRUE)`.

`strand` Only supported when `x` is a `DNAString` object.

Can be an atomic vector, a factor, or an Rle object, in which case it indicates the strand of each transcript (i.e. all the exons in a transcript are considered to be on the same strand). More precisely: it's turned into a factor (or factor-Rle) that has the "standard strand levels" (this is done by calling the `strand` function on it). Then it's recycled to the length of IntegerRangesList object `transcripts` if needed. In the resulting object, the i-th element is interpreted as the strand of all the exons in the i-th transcript.

`strand` can also be a list-like object, in which case it indicates the strand of each exon, individually. Thus it must have the same *shape* as IntegerRangesList object `transcripts` (i.e. same length plus `strand[[i]]` must have the same length as `transcripts[[i]]` for all `i`).

`strand` can only contain `"+"` and/or `"-"` values. `"*"` is not allowed.

`genes` An object containing the locations (i.e. chromosome name, start, end, and strand) of the genes or transcripts with respect to the reference genome. Only GenomicRanges and MultiDb objects are supported at the moment. If the latter, the gene locations are obtained by calling the `genes` function on the MultiDb object internally.

`width` How many bases to extract upstream of each TSS (transcription start site).

`exclude.seqlevels`
A character vector containing the chromosome names (a.k.a. sequence levels) to exclude when the genes are obtained from a MultiDb object.

Details

These are the range based functions for extracting transcript information from a MultiDb object.

Value

a GRanges or GRangesList object

Author(s)

M. Carlson

See Also

- MultiDb-class for how to use the simple "select" interface to extract information from a MultiDb object.
- transcripts for the original transcripts method and related methods.
- transcriptsBy for the original transcriptsBy method and related methods.

Examples

```
## extracting all transcripts from Homo.sapiens with some extra metadata
library(Homo.sapiens)
cols = c("TXNAME","SYMBOL")
res <- transcripts(Homo.sapiens, columns=cols)

## extracting all transcripts from Homo.sapiens, grouped by gene and
## with extra metadata
res <- transcriptsBy(Homo.sapiens, by="gene", columns=cols)

## list possible values for columns argument:
columns(Homo.sapiens)

## Get the TxDb from an MultiDb object (if it's available)
getTxDbIfAvailable(Homo.sapiens)

## Other functions listed above should work in way similar to their TxDb
## counterparts.  So for example:
promoters(Homo.sapiens)
## Should give the same value as:
promoters(getTxDbIfAvailable(Homo.sapiens))
```

Index