

Generating and using Ensembl based annotation packages

Johannes Rainer

Modified: 11 June, 2015. Compiled: June 11, 2015

Contents

1	Introduction	1
2	Using <code>ensemldb</code> annotation packages to retrieve specific annotations	2
3	Extracting gene/transcript/exon models for RNASeq feature counting	7
4	Retrieving sequences for gene/transcript/exon models	9
5	Important notes	11
6	Building an transcript centric database package based on Ensembl annotation	11
6.1	Requirements	11
6.2	Building an annotation package	11
7	Database layout	15

1 Introduction

The `ensemldb` package provides functions to create and use transcript centric annotation databases/packages. The annotation for the databases are directly fetched from Ensembl ¹ using their Perl API. The functionality and data is similar to that of the `TxDb` packages from the `GenomicFeatures` package, but, in addition to retrieve all gene/transcript models and annotations from the database, the `ensemldb` package provides also a filter framework allowing to retrieve annotations for specific entries like genes encoded on a chromosome region or transcript models of lincRNA genes. In the databases, along with the gene and transcript models and their chromosomal coordinates, additional annotations including the gene name (symbol) and NCBI Entrezgene identifiers as well as the gene and transcript biotypes are stored too (see Section 7 for the database layout and an overview of available attributes/columns).

Another main goal of this package is to generate *versioned* annotation packages, i.e. annotation packages that are build for a specific Ensembl release, and are also named according to that (e.g. `EnsDb.Hsapiens.v75` for human gene definitions of the Ensembl code database version 75). This ensures reproducibility, as it allows to load annotations from a specific Ensembl release also if newer versions of annotation packages/releases are

¹<http://www.ensembl.org>

available. It also allows to load multiple annotation packages at the same time in order to e.g. compare gene models between Ensembl releases.

In the example below we load an Ensembl based annotation package for Homo sapiens, Ensembl version 75. The connection to the database is bound to the variable `EnsDb.Hsapiens.v75`.

```
library(EnsDb.Hsapiens.v75)

## print some informations for this package
EnsDb.Hsapiens.v75

## EnsDb for Ensembl:
## |Db type: EnsDb
## |Type of Gene ID: Ensembl Gene ID
## |Supporting package: ensemblldb
## |Db created by: ensemblldb package from Bioconductor
## |script_version: 0.1.2
## |Creation time: Wed Mar 18 09:30:54 2015
## |ensembl_version: 75
## |ensembl_host: manny.i-med.ac.at
## |Organism: homo_sapiens
## |genome_build: GRCh37
## |DBSCHEMAVERSION: 1.0
## | No. of genes: 64102.
## | No. of transcripts: 215647.

## for what organism was the database generated?
organism(EnsDb.Hsapiens.v75)

## [1] "Homo sapiens"
```

2 Using ensemblldb annotation packages to retrieve specific annotations

The ensemblldb package provides a set of filter objects allowing to specify which entries should be fetched from the database. The complete list of filters, which can be used individually or can be combined, is shown below (in alphabetical order):

- `ExonidFilter`: allows to filter the result based on the (Ensembl) exon identifiers.
- `EntrezidFilter`: allows to filter results based on NCBI Entrezgene identifiers of the genes.
- `GenebiotypeFilter`: allows to filter for the gene biotypes defined in the Ensembl database; use the `listGenebiotypes` method to list all available biotypes.
- `GeneidFilter`: allows to filter based on the Ensembl gene IDs.
- `GenenameFilter`: allows to filter based on the names (symbols) of the genes.
- `SeqendFilter`: filter based on the chromosomal end coordinate of the exons, transcripts or genes (correspondingly set `feature="exon"`, `feature="tx"` or `feature="gene"`).
- `SeqnameFilter`: filter by the name of the chromosomes the genes are encoded on.
- `SeqstartFilter`: filter based on the chromosomal start coordinates of the exons, transcripts or genes (correspondingly set `feature="exon"`, `feature="tx"` or `feature="gene"`).
- `SeqstrandFilter`: filter for the chromosome strand on which the genes are encoded.

- `TxbiotypeFilter`: filter on the transcript biotype defined in Ensembl; use the `listTxbiotypes` method to list all available biotypes.
- `TxidFilter`: filter on the Ensembl transcript identifiers.

Each of the filter classes can take a single value or a vector of values (with the exception of the `SeqendFilter` and `SeqstartFilter`) for comparison. In addition, it is possible to specify the *condition* for the filter, e.g. `condition="="` to retrieve all entries matching the filter value, `condition="!="` to negate the filter or also `condition="like"` to allow partial matching. The `condition` parameter for `SeqendFilter` and `SeqendFilter` can take the values `=`, `>`, `>=`, `<` and `<=` (since these filters base on numeric values).

A simple example would be to get all transcripts for the gene *BCL2L11*. To this end we specify a `GenenameFilter` with the value "BCL2L11". As a result we get a `GRanges` object with start, end, strand and seqname of the `GRanges` object being the start coordinate, end coordinate, chromosome name and strand for the respective transcripts. All additional annotations are available as metadata columns. Alternatively, by setting `return.type="DataFrame"`, or `return.type="data.frame"` the method would return a `DataFrame` object or `data.frame`.

```
Tx <- transcripts(EnsDb.Hsapiens.v75, filter=list(GenenameFilter("BCL2L11")))
```

```
Tx
```

```
## GRanges object with 17 ranges and 5 metadata columns:
```

```
##           seqnames           ranges strand |           tx_id
##           <Rle>           <IRanges> <Rle> | <character>
## ENST00000308659      2 [111878491, 111922625] + | ENST00000308659
## ENST00000337565      2 [111878491, 111886423] + | ENST00000337565
## ENST00000357757      2 [111878491, 111919016] + | ENST00000357757
## ENST00000361493      2 [111881323, 111921808] + | ENST00000361493
## ENST00000393252      2 [111880247, 111881537] + | ENST00000393252
##           ...           ...           ...   ...   ...
## ENST00000436733      2 [111881323, 111921808] + | ENST00000436733
## ENST00000437029      2 [111881323, 111921808] + | ENST00000437029
## ENST00000438054      2 [111881329, 111903861] + | ENST00000438054
## ENST00000439718      2 [111881323, 111922220] + | ENST00000439718
## ENST00000452231      2 [111881323, 111921808] + | ENST00000452231
##           tx_biotype tx_cds_seq_start tx_cds_seq_end      gene_id
##           <character>      <numeric>      <numeric>      <character>
## ENST00000308659      protein_coding      111881323      111921808 ENSG00000153094
## ENST00000337565      protein_coding      111881323      111886328 ENSG00000153094
## ENST00000357757      protein_coding      111881323      111919016 ENSG00000153094
## ENST00000361493      nonsense_mediated_decay      111881323      111887812 ENSG00000153094
## ENST00000393252      protein_coding      111881323      111881537 ENSG00000153094
##           ...           ...           ...           ...           ...
## ENST00000436733      nonsense_mediated_decay      111881323      111911385 ENSG00000153094
## ENST00000437029      nonsense_mediated_decay      111881323      111919016 ENSG00000153094
## ENST00000438054      protein_coding      111881329      111902068 ENSG00000153094
## ENST00000439718      nonsense_mediated_decay      111881323      111909428 ENSG00000153094
## ENST00000452231      nonsense_mediated_decay      111881323      111919016 ENSG00000153094
## -----
## seqinfo: 1 sequence from GRCh37 genome
```

```
## as this is a GRanges object we can access e.g. the start coordinates with
head(start(Tx))

## [1] 111878491 111878491 111878491 111881323 111880247 111878491

## or extract the biotype with
head(Tx$tx_biotype)

## [1] "protein_coding"          "protein_coding"          "protein_coding"
## [4] "nonsense_mediated_decay" "protein_coding"          "protein_coding"
```

The parameter columns of the exons, genes and transcripts method allow to specify which database attributes (columns) should be retrieved. Note that these are not restricted to columns of the corresponding database table (e.g. columns of database table *gene* for genes). To get an overview of database tables and available columns the function `listTables` can be used. The method `listColumns` on the other hand lists columns for the specified database table.

```
## list all database tables along with their columns
listTables(EnsDb.Hsapiens.v75)

## $gene
## [1] "gene_id"          "gene_name"          "entrezid"          "gene_biotype"
## [5] "gene_seq_start"   "gene_seq_end"       "seq_name"          "seq_strand"
## [9] "seq_coord_system"
##
## $tx
## [1] "tx_id"            "tx_biotype"         "tx_seq_start"      "tx_seq_end"
## [5] "tx_cds_seq_start" "tx_cds_seq_end"     "gene_id"
##
## $tx2exon
## [1] "tx_id"    "exon_id" "exon_idx"
##
## $exon
## [1] "exon_id"          "exon_seq_start" "exon_seq_end"
##
## $chromosome
## [1] "seq_name"    "seq_length" "is_circular"
##
## $metadata
## [1] "name"  "value"

## list columns from a specific table
listColumns(EnsDb.Hsapiens.v75, "tx")

## [1] "tx_id"            "tx_biotype"         "tx_seq_start"      "tx_seq_end"
## [5] "tx_cds_seq_start" "tx_cds_seq_end"     "gene_id"
```

Thus, we could retrieve all transcripts of the biotype *nonsense_mediated_decay* (which, according to the definitions by Ensembl are transcribed, but most likely not translated in a protein, but rather degraded after transcription) along with the name of the gene for each transcript. Note that we are changing here the `return.type` to `DataFrame`, so the method will return a `DataFrame` with the results instead of the default

GRanges.

```
Tx <- transcripts(EnsDb.Hsapiens.v75,
                  columns=c(listColumns(EnsDb.Hsapiens.v75 , "tx"), "gene_name"),
                  filter=list(TxbiotypeFilter("nonsense_mediated_decay")),
                  return.type="DataFrame")

nrow(Tx)

## [1] 13812

Tx

## DataFrame with 13812 rows and 8 columns
##      gene_name      tx_id      tx_biotype tx_seq_start tx_seq_end
##      <character>    <character>      <character>    <integer>    <integer>
## 1      GCLC  ENST00000504525 nonsense_mediated_decay    53373395    53409886
## 2      GCLC  ENST00000505294 nonsense_mediated_decay    53379005    53387577
## 3     LAS1L  ENST00000484069 nonsense_mediated_decay    64732463    64754655
## 4     ANKIB1 ENST00000439883 nonsense_mediated_decay    91936673    91972431
## 5     ANKIB1 ENST00000413588 nonsense_mediated_decay    91972452    91991574
## ...      ...      ...      ...      ...      ...
## 13808  NDUFA3 ENST00000608867 nonsense_mediated_decay    54606422    54612033
## 13809   TFPT ENST00000609798 nonsense_mediated_decay    54610327    54618665
## 13810  CNOT3 ENST00000609793 nonsense_mediated_decay    54647473    54659419
## 13811  VSTM1 ENST00000608902 nonsense_mediated_decay    54544083    54567207
## 13812  MBOAT7 ENST00000609221 nonsense_mediated_decay    54677109    54693294
##      tx_cds_seq_start tx_cds_seq_end      gene_id
##      <numeric>      <numeric>      <character>
## 1      53379316      53409443  ENSG000000001084
## 2      53380934      53387316  ENSG000000001084
## 3      64744901      64754595  ENSG000000001497
## 4      91936673      91955247  ENSG000000001629
## 5      91972452      91977330  ENSG000000001629
## ...      ...      ...      ...
## 13808      54606422      54609715  ENSG00000273453
## 13809      54611667      54618649  ENSG00000273458
## 13810      54647473      54649358  ENSG00000273459
## 13811      54545493      54567031  ENSG00000273460
## 13812      54687557      54692362  ENSG00000273470
```

To get an overview of allowed/available gene and transcript biotype the functions `listGenebiotypes` and `listTxbiotypes` can be used.

```
## Get all gene biotypes from the database. The GenebiotypeFilter
## allows to filter on these values.
listGenebiotypes(EnsDb.Hsapiens.v75)

## [1] "protein_coding"      "pseudogene"          "processed_transcript"
## [4] "antisense"           "lincRNA"             "polymorphic_pseudogene"
## [7] "IG_V_pseudogene"     "IG_V_gene"           "sense_overlapping"
## [10] "sense_intronic"      "TR_V_gene"           "misc_RNA"
```

```
## [13] "snRNA" "miRNA" "snoRNA"
## [16] "rRNA" "Mt_tRNA" "Mt_rRNA"
## [19] "IG_C_gene" "IG_J_gene" "TR_J_gene"
## [22] "TR_C_gene" "TR_V_pseudogene" "TR_J_pseudogene"
## [25] "IG_D_gene" "IG_C_pseudogene" "TR_D_gene"
## [28] "IG_J_pseudogene" "3prime_overlapping_ncrna" "processed_pseudogene"
## [31] "LRG_gene"

## Get all transcript biotypes from the database.
listTxbiotypes(EnsDb.Hsapiens.v75)

## [1] "protein_coding" "processed_transcript"
## [3] "retained_intron" "nonsense_mediated_decay"
## [5] "unitary_pseudogene" "non_stop_decay"
## [7] "unprocessed_pseudogene" "processed_pseudogene"
## [9] "transcribed_unprocessed_pseudogene" "antisense"
## [11] "lincRNA" "polymorphic_pseudogene"
## [13] "transcribed_processed_pseudogene" "miRNA"
## [15] "pseudogene" "IG_V_pseudogene"
## [17] "snoRNA" "IG_V_gene"
## [19] "sense_overlapping" "sense_intronic"
## [21] "TR_V_gene" "snRNA"
## [23] "misc_RNA" "rRNA"
## [25] "Mt_tRNA" "Mt_rRNA"
## [27] "IG_C_gene" "IG_J_gene"
## [29] "TR_J_gene" "TR_C_gene"
## [31] "TR_V_pseudogene" "TR_J_pseudogene"
## [33] "IG_D_gene" "IG_C_pseudogene"
## [35] "TR_D_gene" "IG_J_pseudogene"
## [37] "3prime_overlapping_ncrna" "translated_processed_pseudogene"
## [39] "LRG_gene"
```

Data can be fetched in an analogous way using the `exons` and `genes` methods. In the example below we retrieve `gene_name`, `entrezid` and the `gene_biotype` of all genes in the database which names start with "BCL2".

```
## We're going to fetch all genes which names start with BCL. To this end
## we define a GenenameFilter with partial matching, i.e. condition "like"
## and a % for any character/string.
BCLs <- genes(EnsDb.Hsapiens.v75,
              columns=c("gene_name", "entrezid", "gene_biotype"),
              filter=list(GenenameFilter("BCL%", condition="like")),
              return.type="DataFrame")

nrow(BCLs)

## [1] 25

BCLs

## DataFrame with 25 rows and 4 columns
```

	gene_id	gene_name	entrezid	gene_biotype
	<character>	<character>	<character>	<character>
## 1	ENSG00000029363	BCLAF1	9774	protein_coding
## 2	ENSG00000069399	BCL3	602;102465879	protein_coding
## 3	ENSG00000099385	BCL7C	9274	protein_coding
## 4	ENSG00000099968	BCL2L13	23786	protein_coding
## 5	ENSG00000106635	BCL7B	9275	protein_coding
##
## 21	ENSG00000186174	BCL9L	283149	protein_coding
## 22	ENSG00000188761	BCL2L15	440603	protein_coding
## 23	ENSG00000258643	BCL2L2-PABPN1	599;100529063	protein_coding
## 24	ENSG00000263151	BCL7B	9275	protein_coding
## 25	ENSG00000266095	BCL9	607	protein_coding

Sometimes it might be useful to know the length of genes or transcripts (i.e. the total sum of nucleotides covered by their exons). Below we calculate the mean length of transcripts from protein coding genes on chromosomes X and Y as well as the average length of snoRNA, snRNA and rRNA transcripts encoded on these chromosomes.

```
## determine the average length of snRNA, snoRNA and rRNA genes encoded on
## chromosomes X and Y.
mean(lengthOf(EnsDb.Hsapiens.v75, of="tx",
              filter=list(GenebiotypeFilter(c("snRNA", "snoRNA", "rRNA")),
                          SeqnameFilter(c("X", "Y")))))

## [1] 116.3046

## determine the average length of protein coding genes encoded on the same
## chromosomes.
mean(lengthOf(EnsDb.Hsapiens.v75, of="tx",
              filter=list(GenebiotypeFilter("protein_coding"),
                          SeqnameFilter(c("X", "Y")))))

## [1] 1920
```

Not unexpectedly, transcripts of protein coding genes are longer than those of snRNA, snoRNA or rRNA genes.

3 Extracting gene/transcript/exon models for RNASeq feature counting

For the feature counting step of an RNAseq experiment, the gene or transcript models (defined by the chromosomal start and end positions of their exons) have to be known. To extract these from an Ensembl based annotation package, the `exonsBy`, `genesBy` and `transcriptsBy` methods can be used in an analogous way as in `TxDb` packages generated by the `GenomicFeatures` package. However, the `transcriptsBy` method does not, in contrast to the method in the `GenomicFeatures` package, allow to return transcripts by `"cds"`. While the annotation packages built by the `=ensembl`db contain the chromosomal start and end coordinates of the coding region (for protein coding genes) they do not assign an ID to each CDS.

A simple use case is to retrieve all genes encoded on chromosomes X and Y from the database.

```

TxByGns <- transcriptsBy(EnsDb.Hsapiens.v75, by="gene",
                        filter=list(SeqlenFilter(c("X", "Y"))))
)

TxByGns

## GRangesList object of length 2908:
## $ENSG000000000003
## GRanges object with 3 ranges and 4 metadata columns:
##      seqnames      ranges strand |      tx_id      tx_biotype
##      <Rle>        <IRanges> <Rle> |      <character>      <character>
## [1]      X [99888439, 99894988]   - | ENS00000494424 processed_transcript
## [2]      X [99883667, 99891803]   - | ENS00000373020      protein_coding
## [3]      X [99887538, 99891686]   - | ENS00000496771 processed_transcript
##      tx_cds_seq_start tx_cds_seq_end
##      <numeric>      <numeric>
## [1]      <NA>      <NA>
## [2]      99885795      99891691
## [3]      <NA>      <NA>
##
## $ENSG000000000005
## GRanges object with 2 ranges and 4 metadata columns:
##      seqnames      ranges strand |      tx_id      tx_biotype
## [1]      X [99839799, 99854882]   + | ENS00000373031      protein_coding
## [2]      X [99848621, 99852528]   + | ENS00000485971 processed_transcript
##      tx_cds_seq_start tx_cds_seq_end
## [1]      99840016      99854714
## [2]      <NA>      <NA>
##
## $ENSG000000001497
## GRanges object with 6 ranges and 4 metadata columns:
##      seqnames      ranges strand |      tx_id      tx_biotype
## [1]      X [64732463, 64754655]   - | ENS00000484069 nonsense_mediated_decay
## [2]      X [64732463, 64754636]   - | ENS00000312391      protein_coding
## [3]      X [64732463, 64754636]   - | ENS00000374804      protein_coding
## [4]      X [64732462, 64754636]   - | ENS00000374811      protein_coding
## [5]      X [64732462, 64754634]   - | ENS00000374807      protein_coding
## [6]      X [64740309, 64743497]   - | ENS00000469091      protein_coding
##      tx_cds_seq_start tx_cds_seq_end
## [1]      64744901      64754595
## [2]      64744901      64754595
## [3]      64732655      64754595
## [4]      64732655      64754595
## [5]      64732655      64754595
## [6]      64740535      64743497
##
## ...
## <2905 more elements>
## -----

```



```
## seqinfo: 2 sequences from GRCh37 genome
```

Since Ensembl contains also definitions of genes that are on chromosome variants (supercontigs), it is advisable to specify the chromosome names for which the gene models should be returned.

In a real use case, we might thus want to retrieve all genes encoded on the *standard* chromosomes. In addition it is advisable to use a `GeneidFilter` to restrict to Ensembl genes only, as also *LRG* (Locus Reference Genomic) genes² are defined in the database, which are partially redundant with Ensembl genes.

```
## will just get exons for all genes on chromosomes 1 to 22, X and Y.
## Note: want to get rid of the "LRG" genes!!!
EnsGenes <- exonsBy(EnsDb.Hsapiens.v75, by="gene",
                    filter=list(SeqnameFilter(c(1:22, "X", "Y")),
                                GeneidFilter("ENSG%", "like")))
```

The code above returns a `GRangesList` that can be used directly as an input for the `summarizeOverlaps` function from the `GenomicAlignments` package³.

Alternatively, the above `GRangesList` can be transformed to a `data.frame` in *SAF* format that can be used as an input to the `featureCounts` function of the `Rsubread` package⁴.

```
## Transforming the GRangesList into a data.frame in SAF format
EnsGenes.SAF <- toSAF(EnsGenes)
```

Note that the ID by which the `GRangesList` is split is used in the *SAF* formatted `data.frame` as the `GeneID`. In the example below this would be the Ensembl gene IDs, while the start, end coordinates (along with the strand and chromosomes) are those of the exons.

In addition, the `disjointExons` function (similar to the one defined in `GenomicFeatures`) can be used to generate a `GRanges` of non-overlapping exon parts which can be used in the `DEXSeq` package.

```
## Create a GRanges of non-overlapping exon parts.
DJE <- disjointExons(EnsDb.Hsapiens.v75,
                     filter=list(SeqnameFilter(c(1:22, "X", "Y")),
                                 GeneidFilter("ENSG%", "like")))
```

4 Retrieving sequences for gene/transcript/exon models

The methods to retrieve exons, transcripts and genes (i.e. exons, transcripts and genes) return by default `GRanges` objects that can be used to retrieve sequences using the `getSeq` method e.g. from `BSgenome` packages. The basic workflow is thus identical to the one for `TxDb` packages, however, it is not straight forward to identify the `BSgenome` package with the matching genomic sequence. Most `BSgenome` packages are named according to the genome build identifier used in UCSC which does not (always) match the genome build name used by Ensembl. Using the Ensembl version provided by the `EnsDb`, the correct genomic sequence can however be retrieved easily from the `AnnotationHub`.

²<http://www.lrg-sequence.org>

³<http://www.ncbi.nlm.nih.gov/pubmed/23950696>

⁴<http://www.ncbi.nlm.nih.gov/pubmed/24227677>

First we load the AnnotationHub data and list all resources available for the specific Ensembl version and species. Next we load the `dna.toplevel.fa` fasta file and retrieve the sequences for all genes defined in the EnsDb package. These sequences represent the sequence between the chromosomal start and end coordinates of the gene. To retrieve the (exonic) sequence of transcripts (i.e. without introns) we first fetch all exons grouped by transcripts and then extract and paste the sequence of all of the transcripts' exons.

```
## load the AnnotationHub data
library(AnnotationHub)
library(EnsDb.Hsapiens.v75)
library(Rsamtools)
ah <- AnnotationHub()

edb <- EnsDb.Hsapiens.v75

## get the Ensembl version
eVersion <- metadata(edb)[metadata(edb)$name=="ensembl_version", "value"]
## query all available files for the Ensembl version
eData <- query(ah, c(organism(edb), paste0("release-", eVersion)))
eData

## retrieve the *dna.toplevel.fa file; this might take some time.
Dna <- ah[["AH20439"]]
## generate an index if none is available
if(is.na(index(Dna))){
  indexFa(Dna)
  Dna <- FaFile(path(Dna))
}

## get start/end coordinates of all genes
genes <- genes(edb)
## subset to all genes that are encoded on chromosomes for which
## we do have DNA sequence available.
genes <- genes[seqnames(genes) %in% seqnames(seqinfo(Dna))]
## get the gene sequences, i.e. the sequence including the sequence of
## all of the gene's exons and introns
geneSeqs <- getSeq(Dna, genes)

## to get the sequence of all transcripts (i.e. only their exonic sequence) we
## fetch the exons grouped by transcripts.
## get all exons by transcript for all genes defined by Ensembl. This excludes
## eventual "LRG" genes, that might be encoded on a sequence for which we don't
## have a DNA sequence.
txExons <- exonsBy(edb, "tx", filter=GeneidFilter("ENS%", condition="like"))
## extract sequence of all of each transcripts' exons and join them into a single
## sequence; this takes quite some time, so we just run it on the first 100.
txSeqs <- lapply(txExons[1:100], function(x){unlist(getSeq(Dna, x))})
```

5 Important notes

These notes might explain eventually unexpected results (and, more importantly, help avoiding them):

- The ordering of the results returned by the `genes`, `exons`, `transcripts` methods can be specified with the `order.by` parameter. The ordering of the results does however **not** correspond to the ordering of values in submitted filter objects.
- Results of `exonsBy`, `transcriptsBy` are always ordered by the `by` argument.

6 Building an transcript centric database package based on Ensembl annotation

The code in this section is not supposed to be automatically executed when the vignette is built, as this would require a working installation of the Ensembl Perl API, which is not expected to be available on each system. Also, fetching data from the Ensembl database takes quite some time, thus, in this section only the code is displayed, but not executed.

6.1 Requirements

The `fetchTablesFromEnsembl` function of the package uses the Ensembl Perl API to retrieve the required annotations from an Ensembl database (e.g. from the main site ensemblldb.ensembl.org). Thus, to use the functionality to built databases, the Ensembl Perl API needs to be installed (see ⁵ for details).

Alternatively, the `ensDbFromGRanges` and `ensDbFromGtf` functions allow to build `EnsDb` SQLite files from a `GRanges` object or an Ensembl GTF file and thus doesn't depend on the Ensembl Perl API. Such `GRanges` objects could for example be retrieved with the `AnnotationHub` package.

6.2 Building an annotation package

The functions below use the Ensembl Perl API to fetch the required data directly from the Ensembl core databases. Thus, the path to the Perl API specific for the desired Ensembl version needs to be added to the `PERL5LIB` environment variable.

An annotation package containing all human genes for Ensembl version 75 can be created using the code in the block below.

```
library(ensemblldb)

## get all human gene/transcript/exon annotations from Ensembl (75)
## the resulting tables will be stored by default to the current working
## directory
fetchTablesFromEnsembl(75, species="human")

## These tables can then be processed to generate a SQLite database
```

⁵http://www.ensembl.org/info/docs/api/api_installation.html

```
## containing the annotations (again, the function assumes the required
## txt files to be present in the current working directory)
DBFile <- makeEnsemblSQLiteFromTables()

## and finally we can generate the package
makeEnsemblDbPackage(ensdb=DBFile, version="0.99.12",
                     maintainer="Johannes Rainer <johannes.rainer@eurac.edu>",
                     author="J Rainer")
```

The generated package can then be build using R CMD build EnsDb.Hsapiens.v75 and installed with R CMD INSTALL EnsDb.Hsapiens.v75*. Note that we could directly generate an EnsDb instance by loading the database file, i.e. by calling `edb <- EnsDb(DBFile)` and work with that annotation object.

To fetch and build annotation packages for plant genomes (e.g. *arabidopsis thaliana*), the *Ensembl genomes* should be specified as a host, i.e. setting `host="mysql-eg-publicsql.ebi.ac.uk"`, `port=4157` and species to e.g. `species="arabidopsis thaliana"`.

An alternative way to build the required annotation database is to use the `ensDbFromGtf` function, that extracts most of the required data from a GTF file that can be downloaded from Ensembl (e.g. from ftp://ftp.ensembl.org/pub/release-75/gtf/homo_sapiens/ for human gene definitions from Ensembl version 75; for plant genomes etc files can be retrieved from <ftp://ftp.ensemblgenomes.org>). All information except the chromosome lengths and the NCBI Entrezgene IDs can be extracted from these GTF files. The function also tries to retrieve chromosome length information automatically from Ensembl.

Below we create the annotation from a gtf file that we fetch directly from Ensembl.

```
library(ensembl)

## 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=gtffile, verbose=TRUE)

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

## alternatively, build the annotation package
## and finally we can generate the package
makeEnsemblDbPackage(ensdb=DB, version="0.99.12",
                     maintainer="Johannes Rainer <johannes.rainer@eurac.edu>",
                     author="J Rainer")
```

The third way to generate an EnsDb database is *via* a GRanges object that contains all the required information. Such GRanges can for example be loaded using the AnnotationHub package. In the example below we load a GRanges containing gene definitions for genes encoded on chromosome Y and generate a EnsDb SQLite database from that information.

```
## Generate a sqlite database from a GRanges object specifying
## genes encoded on chromosome Y
```

```
load(system.file("YGRanges.RData", package="ensemldb"))
Y

## GRanges object with 7155 ranges and 16 metadata columns:
##           seqnames           ranges strand |           source           type
##           <Rle>           <IRanges> <Rle> |           <factor>   <factor>
##      [1]          Y [2652790, 2652894]   + |           snRNA         gene
##      [2]          Y [2652790, 2652894]   + |           snRNA transcript
##      [3]          Y [2652790, 2652894]   + |           snRNA         exon
##      [4]          Y [2654896, 2655740]   - | protein_coding         gene
##      [5]          Y [2654896, 2655740]   - | protein_coding transcript
##      ...      ...      ...      ...      ...      ...
## [7151]          Y [28772667, 28773306]   - | processed_pseudogene transcript
## [7152]          Y [28772667, 28773306]   - | processed_pseudogene         exon
## [7153]          Y [59001391, 59001635]   + |           pseudogene         gene
## [7154]          Y [59001391, 59001635]   + | processed_pseudogene transcript
## [7155]          Y [59001391, 59001635]   + | processed_pseudogene         exon
##           score      phase      gene_id gene_name gene_source gene_biotype
##           <numeric> <integer>   <character> <character>   <character>   <character>
##      [1]          <NA>          <NA> ENSG00000251841 RNU6-1334P      ensembl      snRNA
##      [2]          <NA>          <NA> ENSG00000251841 RNU6-1334P      ensembl      snRNA
##      [3]          <NA>          <NA> ENSG00000251841 RNU6-1334P      ensembl      snRNA
##      [4]          <NA>          <NA> ENSG00000184895          SRY ensembl_havana protein_coding
##      [5]          <NA>          <NA> ENSG00000184895          SRY ensembl_havana protein_coding
##      ...      ...      ...      ...      ...      ...
## [7151]          <NA>          <NA> ENSG00000231514 FAM58CP      havana      pseudogene
## [7152]          <NA>          <NA> ENSG00000231514 FAM58CP      havana      pseudogene
## [7153]          <NA>          <NA> ENSG00000235857 CTBP2P1      havana      pseudogene
## [7154]          <NA>          <NA> ENSG00000235857 CTBP2P1      havana      pseudogene
## [7155]          <NA>          <NA> ENSG00000235857 CTBP2P1      havana      pseudogene
##           transcript_id transcript_name transcript_source exon_number      exon_id
##           <character>   <character>       <character>   <numeric>   <character>
##      [1]          <NA>          <NA>          <NA>          <NA>          <NA>
##      [2] ENST00000516032 RNU6-1334P-201      ensembl          <NA>          <NA>
##      [3] ENST00000516032 RNU6-1334P-201      ensembl           1 ENSE00002088309
##      [4]          <NA>          <NA>          <NA>          <NA>          <NA>
##      [5] ENST00000383070          SRY-001 ensembl_havana <NA>          <NA>
##      ...      ...      ...      ...      ...      ...
## [7151] ENST00000435741 FAM58CP-001      havana          <NA>          <NA>
## [7152] ENST00000435741 FAM58CP-001      havana           1 ENSE00001616687
## [7153]          <NA>          <NA>          <NA>          <NA>          <NA>
## [7154] ENST00000431853 CTBP2P1-001      havana          <NA>          <NA>
## [7155] ENST00000431853 CTBP2P1-001      havana           1 ENSE00001794473
##           tag      ccds_id protein_id
##           <character> <character> <character>
##      [1]          <NA>          <NA>          <NA>
##      [2]          <NA>          <NA>          <NA>
##      [3]          <NA>          <NA>          <NA>
```

```
##      [4]      <NA>      <NA>      <NA>
##      [5]      CCDS      CCDS14772      <NA>
##      ...      ...      ...      ...
##     [7151]      <NA>      <NA>      <NA>
##     [7152]      <NA>      <NA>      <NA>
##     [7153]      <NA>      <NA>      <NA>
##     [7154]      <NA>      <NA>      <NA>
##     [7155]      <NA>      <NA>      <NA>
## -----
##      seqinfo: 1 sequence from GRCh37 genome

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

edb <- EnsDb(DB)
edb

## EnsDb for Ensembl:
## |Db type: EnsDb
## |Type of Gene ID: Ensembl Gene ID
## |Supporting package: ensemblldb
## |Db created by: ensemblldb package from Bioconductor
## |script_version: 0.0.1
## |Creation time: Thu Jun 11 22:07:51 2015
## |ensembl_version: 75
## |ensembl_host: unknown
## |Organism: Homo_sapiens
## |genome_build: GRCh37
## |DBSCHEMAVERSION: 1.0
## |source_file: GRanges object
## | No. of genes: 495.
## | No. of transcripts: 731.
```

In the next example we create an EnsDb database using the AnnotationHub package and load also the corresponding genomic DNA sequence matching the Ensembl version. We thus first query the AnnotationHub package for all resources available for *Mus musculus* and the Ensembl release 77. Next we load the gtf file for the transcript definitions and the dna.toplevel.fa file for the DNA sequence. From the GRanges object representing the gtf file we can build and load an EnsDb. At last we retrieve the sequences of all exons using the getSeq method.

```
## load the AnnotationHub data
library(AnnotationHub)
ah <- AnnotationHub()

## query all available files from Ensembl release 77 for
## Mus musculus
query(ah, c("Mus musculus", "release-77"))

## get the gtf file
```

```
Gtf <- ah[["AH28822"]]
## create a EnsDb database file from the Gtf
DbFile <- ensDbFromGRanges(Gtf, organism="Mus_musculus", version=77)
## we can either generate a database package, or directly load the data
Edb <- EnsDb(DbFile)

## retrieve the toplevel DNA
Dna <- ah[["AH22042"]]

## we next retrieve the sequence of all exons
library(Rsamtools)
exons <- exons(Edb)
exonSeq <- getSeq(Dna, exons)
```

7 Database layout

The database consists of the following tables and attributes (the layout is also shown in Figure 1):

- **gene**: all gene specific annotations.
 - gene_id: the Ensembl ID of the gene.
 - gene_name: the name (symbol) of the gene.
 - entrezid: the NCBI Entrezgene ID(s) of the gene. Note that this can be a ; separated list of IDs for genes that are mapped to more than one Entrezgene.
 - gene_biotype: the biotype of the gene.
 - gene_seq_start: the start coordinate of the gene on the sequence (usually a chromosome).
 - gene_seq_end: the end coordinate of the gene on the sequence.
 - seq_name: the name of the sequence (usually the chromosome name).
 - seq_strand: the strand on which the gene is encoded.
 - seq_coord_system: the coordinate system of the sequence.
- **tx**: all transcript related annotations.
 - tx_id: the Ensembl transcript ID.
 - tx_biotype: the biotype of the transcript.
 - tx_seq_start: the start coordinate of the transcript.
 - tx_seq_end: the 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 of the transcript.
 - gene_id: the gene to which the transcript belongs.
- **exon**: all exon related annotation.
 - exon_id: the Ensembl exon ID.
 - exon_seq_start: the start coordinate of the exon.
 - exon_seq_end: the end coordinate of the exon.
- **tx2exon**: provides the n:m mapping between transcripts and exons.
 - tx_id: the Ensembl transcript ID.
 - exon_id: the Ensembl exon ID.

- `exon_idx`: the index of the exon in the corresponding transcript, always from 5' to 3' of the transcript.
- **chromosome**: provides some information about the chromosomes.
 - `seq_name`: the name of the sequence/chromosome.
 - `seq_length`: the length of the sequence.
 - `is_circular`: whether the sequence is circular.
- **information**: some additional, internal, informations (Genome build, Ensembl version etc).
 - `key`
 - `value`

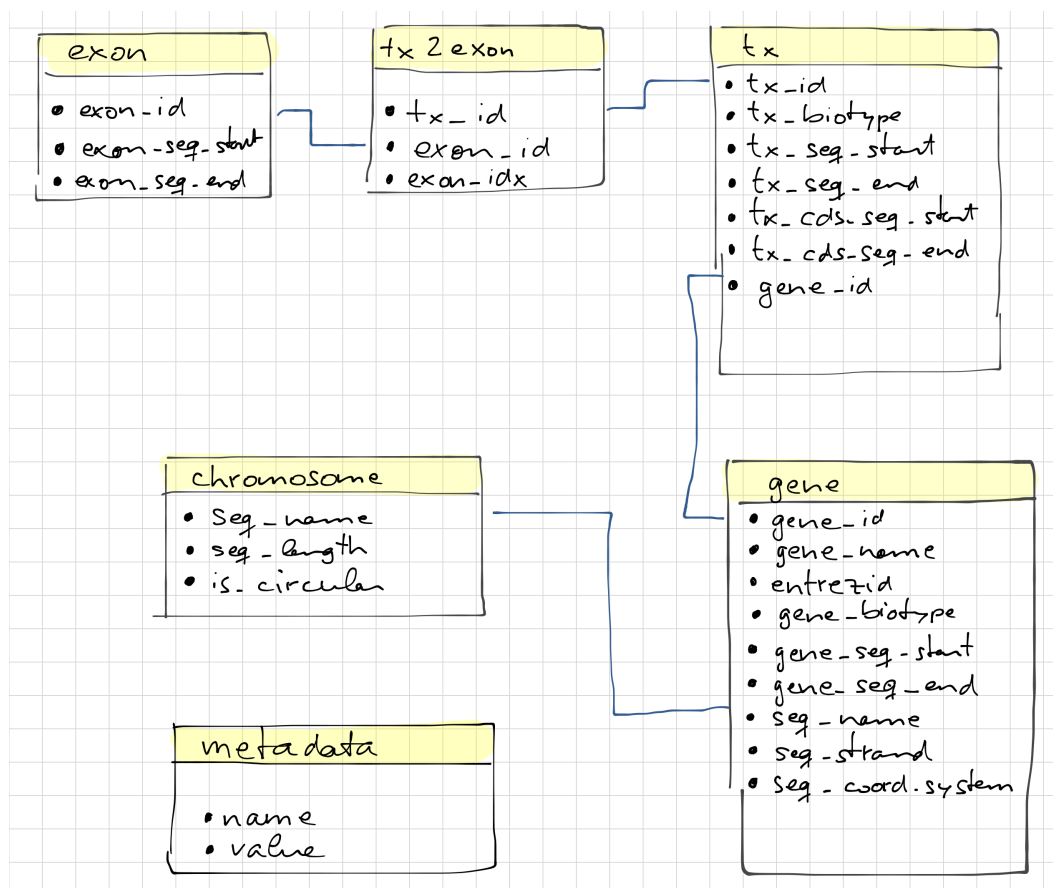


Figure 1: Database layout.