

# AnnotationDbi: Introduction To Bioconductor Annotation Packages

Marc Carlson

May 3, 2016

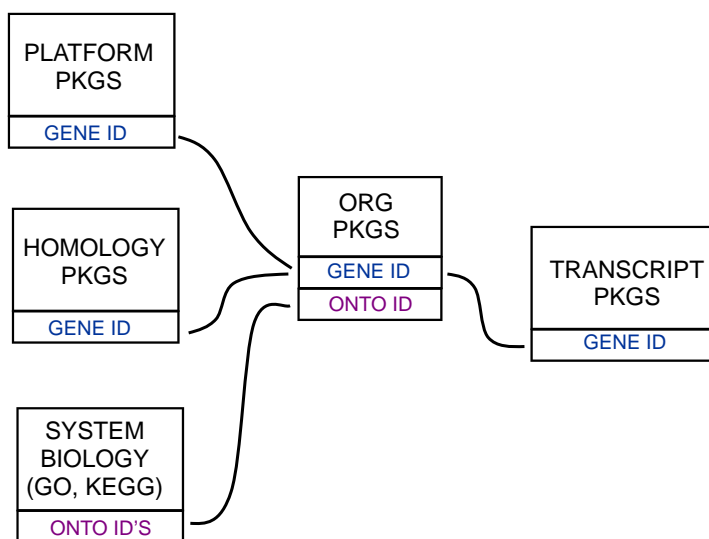


Figure 1: Annotation Packages: the big picture

*Bioconductor* provides extensive annotation resources. These can be *gene centric*, or *genome centric*. Annotations can be provided in packages curated by *Bioconductor*, or obtained from web-based resources. This vignette is primarily concerned with describing the annotation resources that are available as packages. More advanced users who wish to learn about how to make new annotation packages should see the vignette titled "Creating select Interfaces for custom Annotation resources" from the *AnnotationForge* package.

Gene centric *AnnotationDbi* packages include:

- Organism level: e.g. *org.Mm.eg.db*.
- Platform level: e.g. *hgu133plus2.db*, *hgu133plus2.probes*, *hgu133plus2.cdf*.
- Homology level: e.g. *hom.Dm.inp.db*.
- System-biology level: *GO.db*

Genome centric *GenomicFeatures* packages include

- Transcriptome level: e.g. *TxDb.Hsapiens.UCSC.hg19.knownGene*, *EnsDb.Hsapiens.v75*.
- Generic genome features: Can generate via *GenomicFeatures*

One web-based resource accesses [biomart](#), via the *biomaRt* package:

- Query web-based 'biomart' resource for genes, sequence, SNPs, and etc.

The most popular annotation packages have been modified so that they can make use of a new set of methods to more easily access their contents. These four methods are named: `columns`, `keytypes`, `keys` and `select`. And they are described in this vignette. They can currently be used with all chip, organism, and *TxDb* packages along with the popular *GO.db* package.

For the older less popular packages, there are still convenient ways to retrieve the data. The *How to use bimap from the ".db" annotation packages* vignette in the *AnnotationDbi* package is a key reference for learnign about how to use bimap objects.

Finally, all of the '.db' (and most other *Bioconductor* annotation packages) are updated every 6 months corresponding to each release of *Bioconductor*. Exceptions are made for packages where the actual resources that the packages are based on have not themselves been updated.

## 0.1 AnnotationDb objects and the select method

As previously mentioned, a new set of methods have been added that allow a simpler way of extracting identifier based annotations. All the annotation packages that support these new methods expose an object named exactly the same as the package itself. These objects are collectively called *AnntoationDb* objects for the class that they all inherit from. The more specific classes (the ones that you will actually see in the wild) have names like *OrgDb*, *ChipDb* or *TxDb* objects. These names correspond to the kind of package (and underlying schema) being represented. The methods that can be applied to all of these objects are `columns`, `keys`, `keytypes` and `select`.

In addition, another accessor has recently been added which allows extraction of one column at at time. the `mapIds` method allows users to extract data into either a named character vector, a list or even a `SimpleCharacterList`. This method should work with all the different kinds of *AnntoationDb* objects described below.

## 0.2 ChipDb objects and the select method

An extremely common kind of Annotation package is the so called platform based or chip based package type. This package is intended to make the manufacturer labels for a series of probes or probesets to a wide range of gene-based features. A package of this kind will load an *ChipDb* object. Below is a set of examples to show how you might use the standard 4 methods to interact with an object of this type.

First we need to load the package:

```
library(hgu95av2.db)
```

If we list the contents of this package, we can see that one of the many things loaded is an object named after the package "hgu95av2.db":

```
ls("package:hgu95av2.db")
```

```
## [1] "hgu95av2"                "hgu95av2.db"            "hgu95av2ACCNUM"
## [4] "hgu95av2ALIAS2PROBE"     "hgu95av2CHR"           "hgu95av2CHRLNGTHS"
## [7] "hgu95av2CHRLLOC"        "hgu95av2CHRLLOCEND"    "hgu95av2ENSEMBL"
## [10] "hgu95av2ENSEMBL2PROBE"  "hgu95av2ENTREZID"      "hgu95av2ENZYME"
```

```
## [13] "hgu95av2ENZYME2PROBE" "hgu95av2GENENAME" "hgu95av2G0"
## [16] "hgu95av2G02ALLPROBES" "hgu95av2G02PROBE" "hgu95av2MAP"
## [19] "hgu95av2MAPCOUNTS" "hgu95av2OMIM" "hgu95av2ORGANISM"
## [22] "hgu95av2ORGPKG" "hgu95av2PATH" "hgu95av2PATH2PROBE"
## [25] "hgu95av2PFAM" "hgu95av2PMID" "hgu95av2PMID2PROBE"
## [28] "hgu95av2PROSITE" "hgu95av2REFSEQ" "hgu95av2SYMBOL"
## [31] "hgu95av2UNIGENE" "hgu95av2UNIPROT" "hgu95av2_dbInfo"
## [34] "hgu95av2_dbconn" "hgu95av2_dbfile" "hgu95av2_dbschema"
```

We can look at this object to learn more about it:

```
hgu95av2.db

## ChipDb object:
## | DBSCHEMAVERSION: 2.1
## | Db type: ChipDb
## | Supporting package: AnnotationDbi
## | DBSCHEMA: HUMANCHIP_DB
## | ORGANISM: Homo sapiens
## | SPECIES: Human
## | MANUFACTURER: Affymetrix
## | CHIPNAME: Human Genome U95 Set
## | MANUFACTURERURL: http://www.affymetrix.com/support/technical/byproduct.affx?product=hgu95
## | EGSOURCEDATE: 2015-Sep27
## | EGSOURCENAME: Entrez Gene
## | EGSOURCEURL: ftp://ftp.ncbi.nlm.nih.gov/gene/DATA
## | CENTRALID: ENTREZID
## | TAXID: 9606
## | GOSOURCENAME: Gene Ontology
## | GOSOURCEURL: ftp://ftp.geneontology.org/pub/go/godatabase/archive/latest-lite/
## | GOSOURCEDATE: 20150919
## | GOEGSOURCEDATE: 2015-Sep27
## | GOEGSOURCENAME: Entrez Gene
## | GOEGSOURCEURL: ftp://ftp.ncbi.nlm.nih.gov/gene/DATA
## | KEGGSOURCENAME: KEGG GENOME
## | KEGGSOURCEURL: ftp://ftp.genome.jp/pub/kegg/genomes
## | KEGGSOURCEDATE: 2011-Mar15
## | GPSOURCENAME: UCSC Genome Bioinformatics (Homo sapiens)
## | GPSOURCEURL: ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19
## | GPSOURCEDATE: 2010-Mar22
## | ENSOURCEDATE: 2015-Jul16
## | ENSOURCENAME: Ensembl
## | ENSOURCEURL: ftp://ftp.ensembl.org/pub/current_fasta
## | UPSOURCENAME: Uniprot
## | UPSOURCEURL: http://www.uniprot.org/
## | UPSOURCEDATE: Thu Oct 1 23:31:58 2015

##
## Please see: help('select') for usage information
```

If we want to know what kinds of data are retrievable via `select`, then we should use the `columns` method like this:

```
columns(hgu95av2.db)

## [1] "ACCNUM"      "ALIAS"       "ENSEMBL"     "ENSEMBLPROT" "ENSEMBLTRANS"
## [6] "ENTREZID"    "ENZYME"      "EVIDENCE"     "EVIDENCEALL"  "GENENAME"
## [11] "GO"          "GOALL"       "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"    "ONTOLOGYALL" "PATH"         "PFAM"         "PMID"
## [21] "PROBEID"     "PROSITE"     "REFSEQ"       "SYMBOL"       "UCSCKG"
## [26] "UNIGENE"     "UNIPROT"
```

If we are further curious to know more about those values for columns, we can consult the help pages. Asking about any of these values will pull up a manual page describing the different fields and what they mean.

```
help("SYMBOL")
```

If we are curious about what kinds of fields we could potentially use as keys to query the database, we can use the `keytypes` method. In a perfect world, this method will return values very similar to what was returned by `columns`, but in reality, some kinds of values make poor keys and so this list is often shorter.

```
keytypes(hgu95av2.db)
## [1] "ACCNUM"      "ALIAS"       "ENSEMBL"     "ENSEMBLPROT" "ENSEMBLTRANS"
## [6] "ENTREZID"    "ENZYME"      "EVIDENCE"     "EVIDENCEALL"  "GENENAME"
## [11] "GO"          "GOALL"       "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"    "ONTOLOGYALL" "PATH"         "PFAM"         "PMID"
## [21] "PROBEID"     "PROSITE"     "REFSEQ"       "SYMBOL"       "UCSCKG"
## [26] "UNIGENE"     "UNIPROT"
```

If we want to extract some sample keys of a particular type, we can use the `keys` method.

```
head(keys(hgu95av2.db, keytype="SYMBOL"))
## [1] "A1BG" "A2M" "A2MP1" "NAT1" "NAT2" "NATP"
```

And finally, if we have some keys, we can use `select` to extract them. By simply using appropriate argument values with `select` we can specify what keys we want to look up values for (`keys`), what we want returned back (`columns`) and the type of keys that we are passing in (`keytype`)

```
#1st get some example keys
k <- head(keys(hgu95av2.db,keytype="PROBEID"))
# then call select
select(hgu95av2.db, keys=k, columns=c("SYMBOL","GENENAME"), keytype="PROBEID")

## 'select()' returned 1:1 mapping between keys and columns
```

	PROBEID	SYMBOL	GENENAME
## 1	1000_at	MAPK3	mitogen-activated protein kinase 3
## 2	1001_at	TIE1	tyrosine kinase with immunoglobulin like and EGF like domains 1
## 3	1002_f_at	CYP2C19	cytochrome P450 family 2 subfamily C member 19
## 4	1003_s_at	CXCR5	chemokine (C-X-C motif) receptor 5
## 5	1004_at	CXCR5	chemokine (C-X-C motif) receptor 5
## 6	1005_at	DUSP1	dual specificity phosphatase 1

And as you can see, when you call the code above, `select` will try to return a `data.frame` with all the things you asked for matched up to each other.

Finally if you wanted to extract only one column of data you could instead use the `mapIds` method like this:

```
#1st get some example keys
k <- head(keys(hgu95av2.db, keytype="PROBEID"))
# then call mapIds
mapIds(hgu95av2.db, keys=k, column=c("GENENAME"), keytype="PROBEID")

## 'select()' returned 1:1 mapping between keys and columns
##
##                                1000_at
##                                "mitogen-activated protein kinase 3"
##                                1001_at
## "tyrosine kinase with immunoglobulin like and EGF like domains 1"
##                                1002_f_at
##                                "cytochrome P450 family 2 subfamily C member 19"
##                                1003_s_at
##                                "chemokine (C-X-C motif) receptor 5"
##                                1004_at
##                                "chemokine (C-X-C motif) receptor 5"
##                                1005_at
##                                "dual specificity phosphatase 1"
```

### 0.3 OrgDb objects and the select method

An organism level package (an 'org' package) uses a central gene identifier (e.g. Entrez Gene id) and contains mappings between this identifier and other kinds of identifiers (e.g. GenBank or Uniprot accession number, RefSeq id, etc.). The name of an org package is always of the form *org.jAb<sub>i</sub>.jid<sub>i</sub>.db* (e.g. *org.Sc.sgd.db*) where *jAb<sub>i</sub>* is a 2-letter abbreviation of the organism (e.g. *Sc* for *Saccharomyces cerevisiae*) and *jid<sub>i</sub>* is an abbreviation (in lower-case) describing the type of central identifier (e.g. *sgd* for gene identifiers assigned by the *Saccharomyces* Genome Database, or *eg* for Entrez Gene ids).

Just as the chip packages load a *ChipDb* object, the org packages will load a *OrgDb* object. The following exercise should acquaint you with the use of these methods in the context of an organism package.

#### Exercise 1

Display the *OrgDb* object for the [org.Hs.eg.db](#) package.

Use the `columns` method to discover which sorts of annotations can be extracted from it. Is this the same as the result from the `keytypes` method? Use the `keytypes` method to find out.

Finally, use the `keys` method to extract UNIPROT identifiers and then pass those keys in to the `select` method in such a way that you extract the gene symbol and KEGG pathway information for each. Use the `help` system as needed to learn which values to pass in to `columns` in order to achieve this.

#### Solution:

```
library(org.Hs.eg.db)
columns(org.Hs.eg.db)
```

```
## [1] "ACCNUM"      "ALIAS"      "ENSEMBL"    "ENSEMBLPROT" "ENSEMBLTRANS"
## [6] "ENTREZID"    "ENZYME"     "EVIDENCE"   "EVIDENCEALL"  "GENENAME"
## [11] "GO"          "GOALL"      "IPI"        "MAP"          "OMIM"
## [16] "ONTOLOGY"    "ONTOLOGYALL" "PATH"       "PFAM"         "PMID"
## [21] "PROSITE"     "REFSEQ"     "SYMBOL"     "UCSCKG"       "UNIGENE"
## [26] "UNIPROT"
```

```
help("SYMBOL") ## for explanation of these columns and keytypes values
```

```
keytypes(org.Hs.eg.db)
```

```
## [1] "ACCNUM"      "ALIAS"      "ENSEMBL"    "ENSEMBLPROT" "ENSEMBLTRANS"
## [6] "ENTREZID"    "ENZYME"     "EVIDENCE"   "EVIDENCEALL"  "GENENAME"
## [11] "GO"          "GOALL"      "IPI"        "MAP"          "OMIM"
## [16] "ONTOLOGY"    "ONTOLOGYALL" "PATH"       "PFAM"         "PMID"
## [21] "PROSITE"     "REFSEQ"     "SYMBOL"     "UCSCKG"       "UNIGENE"
## [26] "UNIPROT"
```

```
uniKeys <- head(keys(org.Hs.eg.db, keytype="UNIPROT"))
cols <- c("SYMBOL", "PATH")
select(org.Hs.eg.db, keys=uniKeys, columns=cols, keytype="UNIPROT")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```
##      UNIPROT SYMBOL  PATH
## 1    P04217  A1BG   <NA>
## 2    V9HWD8  A1BG   <NA>
## 3    P01023  A2M    04610
## 4    P18440  NAT1   00232
## 5    P18440  NAT1   00983
## 6    P18440  NAT1   01100
## 7    Q400J6  NAT1   00232
## 8    Q400J6  NAT1   00983
## 9    Q400J6  NAT1   01100
## 10   F5H5R8  NAT1   00232
## 11   F5H5R8  NAT1   00983
## 12   F5H5R8  NAT1   01100
```

So how could you use select to annotate your results? This next exercise should help you to understand how that should generally work.

## Exercise 2

Please run the following code snippet (which will load a fake data result that I have provided for the purposes of illustration):

```
load(system.file("extdata", "resultTable.Rda", package="AnnotationDbi"))
head(resultTable)
```

```
##      logConc    logFC LR.statistic      PValue      FDR
## 100418920 -9.639471 -4.679498    378.0732 3.269307e-84 2.613484e-80
## 100419779 -10.638865 -4.264830    291.1028 2.859424e-65 1.142912e-61
```

```
## 100271867 -11.448981 -4.009603      222.3653 2.757135e-50 7.346846e-47
## 100287169 -11.026699 -3.486593      206.7771 6.934967e-47 1.385953e-43
## 100287735 -11.036862  3.064980      204.1235 2.630432e-46 4.205535e-43
## 100421986 -12.276297 -4.695736      190.5368 2.427556e-43 3.234314e-40
```

The rownames of this table happen to provide entrez gene identifiers for each row (for human). Find the gene symbol and gene name for each of the rows in resultTable and then use the merge method to attach those annotations to it.

### Solution:

```
annots <- select(org.Hs.eg.db, keys=rownames(resultTable),
                 columns=c("SYMBOL", "GENENAME"), keytype="ENTREZID")

## 'select()' returned 1:1 mapping between keys and columns

resultTable <- merge(resultTable, annots, by.x=0, by.y="ENTREZID")
head(resultTable)
```

##	Row.names	logConc	logFC	LR.statistic	PValue	FDR	SYMBOL
## 1	100127888	-10.57050	2.758937	182.8937	1.131473e-41	1.130624e-38	SLC04A1-AS1
## 2	100131223	-12.37808	-4.654318	179.2331	7.126423e-41	6.329847e-38	LOC100131223
## 3	100271381	-12.06340	3.511937	188.4824	6.817155e-43	7.785191e-40	RPS28P8
## 4	100271867	-11.44898	-4.009603	222.3653	2.757135e-50	7.346846e-47	MPVQTL1
## 5	100287169	-11.02670	-3.486593	206.7771	6.934967e-47	1.385953e-43	<NA>
## 6	100287735	-11.03686	3.064980	204.1235	2.630432e-46	4.205535e-43	TTY13B

```
##
##                               GENENAME
## 1                               SLC04A1 antisense RNA 1
## 2 ADP ribosylation factor like GTPase 8B pseudogene
## 3                               ribosomal protein S28 pseudogene 8
## 4                               Mean platelet volume QTL1
## 5                               <NA>
## 6          testis-specific transcript, Y-linked 13B
```

## 0.4 Using select with GO.db

When you load the GO.db package, a *GODb* object is also loaded. This allows you to use the columns, keys, keytypes and select methods on the contents of the GO ontology. So if for example, you had a few GO IDs and wanted to know more about it, you could do it like this:

```
library(GO.db)
GOIDs <- c("GO:0042254", "GO:0044183")
select(GO.db, keys=GOIDs, columns="DEFINITION", keytype="GOID")

## 'select()' returned 1:1 mapping between keys and columns

##          GOID
## 1 GO:0042254
## 2 GO:0044183
```

```
##
## 1      A cellular process that results in the biosynthesis of constituent macromolecules, asse
## 2 Interacting selectively and non-covalently with any protein or protein complex (a complex of
```

## 0.5 Using select with TxDb packages

A *TxDb* package (a 'TxDb' package) connects a set of genomic coordinates to various transcript oriented features. The package can also contain Identifiers to features such as genes and transcripts, and the internal schema describes the relationships between these different elements. All TxDb containing packages follow a specific naming scheme that tells where the data came from as well as which build of the genome it comes from.

### Exercise 3

Display the TxDb object for the [TxDb.Hsapiens.UCSC.hg19.knownGene](#) package.

As before, use the *columns* and *keytypes* methods to discover which sorts of annotations can be extracted from it.

Use the *keys* method to extract just a few gene identifiers and then pass those keys in to the *select* method in such a way that you extract the transcript ids and transcript starts for each.

### Solution:

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)

## Loading required package: GenomicFeatures
## Loading required package: GenomeInfoDb
## Loading required package: GenomicRanges

txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
txdb

## TxDb object:
## # Db type: TxDb
## # Supporting package: GenomicFeatures
## # Data source: UCSC
## # Genome: hg19
## # Organism: Homo sapiens
## # Taxonomy ID: 9606
## # UCSC Table: knownGene
## # Resource URL: http://genome.ucsc.edu/
## # Type of Gene ID: Entrez Gene ID
## # Full dataset: yes
## # miRBase build ID: GRCh37
## # transcript_nrow: 82960
## # exon_nrow: 289969
## # cds_nrow: 237533
## # Db created by: GenomicFeatures package from Bioconductor
## # Creation time: 2015-10-07 18:11:28 +0000 (Wed, 07 Oct 2015)
```



```
## # GenomicFeatures version at creation time: 1.21.30
## # RSQLite version at creation time: 1.0.0
## # DBSCHEMAVERSION: 1.1

columns(txdb)

## [1] "CDSCHROM" "CDSSEND" "CDSID" "CDSNAME" "CDSSTART" "CDSSTRAND"
## [7] "EXONCHROM" "EXONEND" "EXONID" "EXONNAME" "EXONRANK" "EXONSTART"
## [13] "EXONSTRAND" "GENEID" "TXCHROM" "TXEND" "TXID" "TXNAME"
## [19] "TXSTART" "TXSTRAND" "TXTYPE"

keytypes(txdb)

## [1] "CDSID" "CDSNAME" "EXONID" "EXONNAME" "GENEID" "TXID" "TXNAME"

keys <- head(keys(txdb, keytype="GENEID"))
cols <- c("TXID", "TXSTART")
select(txdb, keys=keys, columns=cols, keytype="GENEID")

## 'select()' returned 1:many mapping between keys and columns

##      GENEID  TXID  TXSTART
## 1         1 70455 58858172
## 2         1 70456 58859832
## 3        10 31944 18248755
## 4       100 72132 43248163
## 5      1000 65378 25530930
## 6      1000 65379 25530930
## 7     10000  7895 243651535
## 8     10000  7896 243663021
## 9     10000  7897 243663021
## 10 100008586 75890 49217763
```

As is widely known, in addition to providing access via the `select` method, *TxDb* objects also provide access via the more familiar `transcripts`, `exons`, `cds`, `transcriptsBy`, `exonsBy` and `cdsBy` methods. For those who do not yet know about these other methods, more can be learned by seeing the vignette called: *Making and Utilizing TxDb Objects* in the *GenomicFeatures* package.

## 0.6 Using select with EnsDb packages

Similar to the *TxDb* objects/packages discussed in the previous section, *EnsDb* objects/packages provide genomic coordinates of gene models along with additional annotations (e.g. gene names, biotypes etc) but are tailored to annotations provided by Ensembl. The central methods `columns`, `keys`, `keytypes` and `select` are all implemented for *EnsDb* objects. In addition, these methods allow also the use of the *EnsDb* specific filtering framework to retrieve only selected information from the database (see vignette of the *ensemblDb* package for more information).

In the example below we first evaluate which columns, keys and keytypes are available for *EnsDb* objects and fetch then the transcript ids, genomic start coordinate and transcript biotype for some genes.

```

library(EnsDb.Hsapiens.v75)

## Loading required package: ensembldb

edb <- EnsDb.Hsapiens.v75
edb

## EnsDb for Ensembl:
## |Db type: EnsDb
## |Type of Gene ID: Ensembl Gene ID
## |Supporting package: ensembldb
## |Db created by: ensembldb 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.

## List all columns
columns(edb)

## [1] "ENTREZID"      "EXONID"      "EXONIDX"      "EXONSEQEND"   "EXONSEQSTART"
## [6] "GENEBIOTYPE"   "GENEID"      "GENENAME"     "GENESEQEND"   "GENESEQSTART"
## [11] "ISCIRCULAR"    "SEQCOORDSYSTEM" "SEQLength"    "SEQNAME"      "SEQSTRAND"
## [16] "TXBIOTYPE"     "TXCDSSEQEND"  "TXCDSSEQSTART" "TXID"         "TXSEQEND"
## [21] "TXSEQSTART"

## List all keytypes
keytypes(edb)

## [1] "ENTREZID"      "EXONID"      "GENEBIOTYPE" "GENEID"      "GENENAME"    "SEQNAME"
## [7] "SEQSTRAND"     "TXBIOTYPE"   "TXID"

## Get the first
keys <- head(keys(edb, keytype="GENEID"))

## Get the data
select(edb, keys=keys, columns=c("TXID", "TXSEQSTART", "TXBIOTYPE"),
       keytype="GENEID")

##           GENEID           TXID TXSEQSTART           TXBIOTYPE
## 1 ENSG00000000460 ENST00000286031 169764550 protein_coding
## 2 ENSG00000000460 ENST00000359326 169764190 protein_coding
## 3 ENSG00000000457 ENST00000367770 169822215 protein_coding
## 4 ENSG00000000457 ENST00000367771 169818772 protein_coding
## 5 ENSG00000000457 ENST00000367772 169821804 protein_coding
## 6 ENSG00000000419 ENST00000371582  49551404 protein_coding
## 7 ENSG00000000419 ENST00000371583  49551490 protein_coding

```

```
## 8  ENSG000000000419 ENST00000371584 49551482 protein_coding
## 9  ENSG000000000419 ENST00000371588 49551404 protein_coding
## 10 ENSG000000000003 ENST00000373020 99883667 protein_coding
## 11 ENSG000000000005 ENST00000373031 99839799 protein_coding
## 12 ENSG000000000938 ENST00000374003 27939180 protein_coding
## 13 ENSG000000000938 ENST00000374004 27938811 protein_coding
## 14 ENSG000000000938 ENST00000374005 27938575 protein_coding
## 15 ENSG000000000938 ENST00000399173 27938803 protein_coding
## 16 ENSG000000000419 ENST00000413082 49552685 protein_coding
## 17 ENSG000000000460 ENST00000413811 169764181 protein_coding
## 18 ENSG000000000457 ENST00000423670 169823652 protein_coding
## 19 ENSG000000000460 ENST00000456684 169764220 protein_coding
## 20 ENSG000000000938 ENST00000457296 27942025 protein_coding
## 21 ENSG000000000460 ENST00000459772 169764220 processed_transcript
## 22 ENSG000000000419 ENST00000466152 49551404 processed_transcript
## 23 ENSG000000000460 ENST00000466580 169764220 processed_transcript
## 24 ENSG000000000938 ENST00000468038 27949918 processed_transcript
## 25 ENSG000000000457 ENST00000470238 169828260 processed_transcript
## 26 ENSG000000000460 ENST00000472795 169763871 processed_transcript
## 27 ENSG000000000938 ENST00000475472 27949895 processed_transcript
## 28 ENSG000000000460 ENST00000481744 169764184 processed_transcript
## 29 ENSG000000000005 ENST00000485971 99848621 processed_transcript
## 30 ENSG000000000003 ENST00000494424 99888439 processed_transcript
## 31 ENSG000000000419 ENST00000494752 49551433 processed_transcript
## 32 ENSG000000000003 ENST00000496771 99887538 processed_transcript
## 33 ENSG000000000460 ENST00000496973 169764184 processed_transcript
## 34 ENSG000000000460 ENST00000498289 169631245 processed_transcript
## 35 ENSG000000000938 ENST00000545953 27938803 protein_coding
```

We can modify the queries above to retrieve only genes encoded on chromosome Y. To this end we use filter objects available for *EnsDb* objects and its methods.

```
## Retrieve all gene IDs of all lincRNAs encoded on chromosome Y
linkY <- keys(edb,
              filter=list(GenebiotypeFilter("lincRNA"), SeqnameFilter("Y")))
length(linkY)

## [1] 48

## We get now all transcripts for these genes.
txs <- select(edb, keys=linkY, columns=c("TXID", "TXSEQSTART", "TXBIOTYPE"),
              keytype="GENEID")
nrow(txs)

## [1] 66

## Alternatively, we could specify/pass the filters with the keys argument.
txs <- select(edb, keys=list(GenebiotypeFilter("lincRNA"), SeqnameFilter("Y")),
              columns=c("TXID", "TXSEQSTART", "TXBIOTYPE"))
nrow(txs)
```

