Understand Bioconductor Annotation Packages

2018-09-18 | R , Bioconductor | 0 Comments | | 22k | 0:20

This note is to help me figure out the design schema of annotation packages in Bioconductor. And this note is mainly compiled from:

- Annotation Packages: the big picture. Fantastic slide!
- Introduction To Bioconductor Annotation Packages
- Package 'AnnotationDbi' Reference Manual
- Making and Utilizing TxDb Objects

Introduction

Packages in Bioconductor can be divided into three categories: software, annotation and experiment.

> 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.

Schema of Annotation Packages

Here is a very representative graph, but not informative-enough.

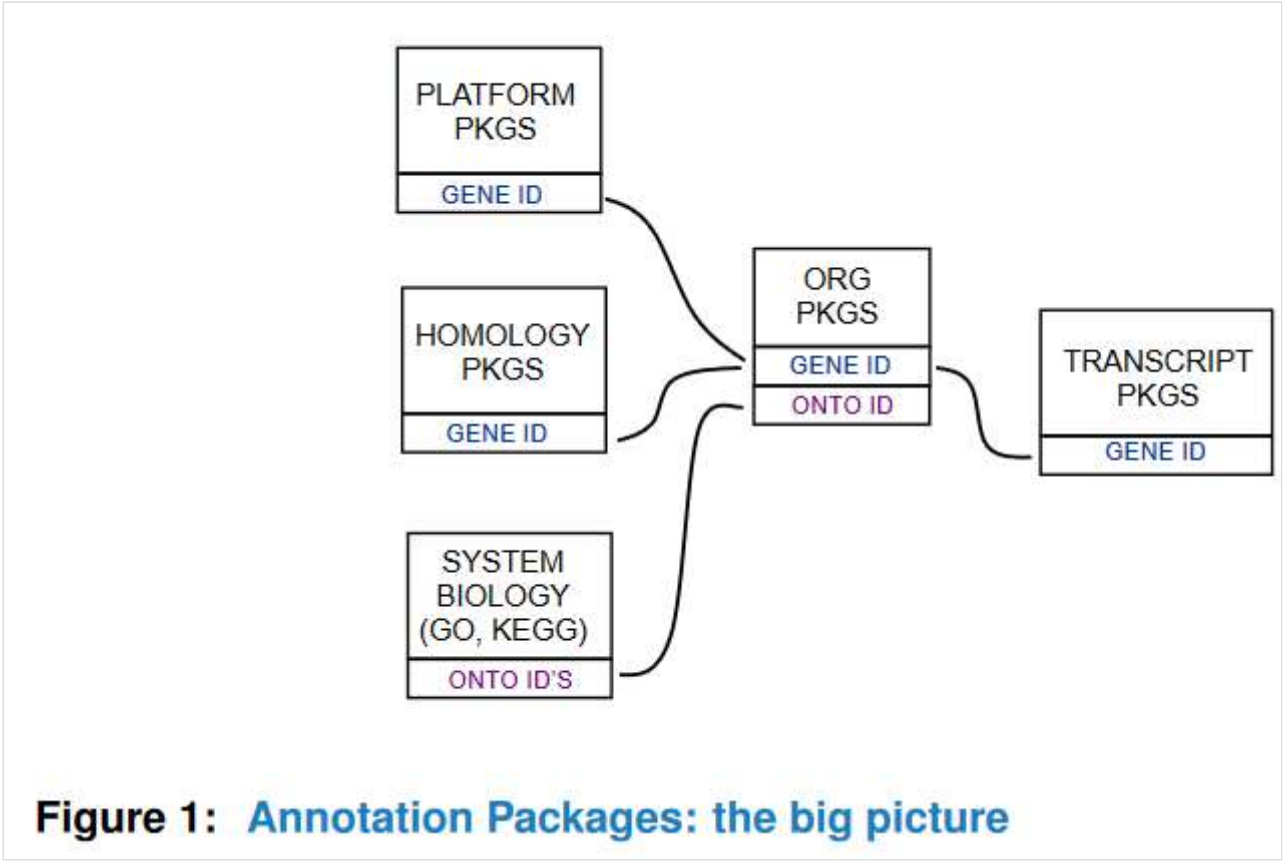


Figure 1: Annotation Packages: the big picture

There are three major types of annotation in Bioconductor:

- Gene centric `AnnotationDb` packages:
  - 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: e.g. `GO.db`.
- Genome centric `GenomicFeatures` packages include:
  - Transcriptome level: e.g. `TxDb.Hsapiens.UCSC.hg19.knownGene`, `EnsDb.Hsapiens.v75`.
  - Generic genome feature: can generate via `GenomicFeatures`.
- One web-based resource accesss `biomart`, via `biomaRt` package:
  - Query web-based 'biomart' resource for genes, sequence, SNPs, and etc.

Working with AnotationDb objects

`AnnotationDb` is the virtual base class for all annotation packages. It contain a database connection and is meant to be the parent for a set of classes in the Bioconductor annotation packages. These classes will

provide a means of dispatch for a widely available set of select methods and thus allow the easy extraction of data from the annotation packages.

All the annotation packages that base on `AnnotationDb` object expose an object named exactly the same as the package itself.

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

```
[1] "OrgDb"
attr(,"package")
[1] "AnnotationDbi"
```

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.

Like:

- `org.Hs.eg.db` - Genome wide annotation for Human.
- `TxDb.Hsapiens.UCSC.hg19.knownGene` - Annotation package for TxDb object(s).
- `hgu95av2.db` - annotations for the hgu95av2 Affymetrix platform.

Methods

`select`, `columns` and `keys` are used together to extract data from an `AnnotationDb` object (or any object derived from the parent class). Examples of classes derived from the `AnnotationDb` object include (but are not limited to): `ChipDb`, `OrgDb`, `GODb`, `InparanoidDb` and `ReactomeDb`.

- `columns` shows which kinds of data can be returned for the AnnotationDb object.
- `keytypes` allows the user to discover which keytypes can be passed in to select or keys and the keytype argument.
- `keys` returns keys for the database contained in the `AnnotationDb` object . This method is already documented in the keys manual page but is mentioned again here because it’ s usage with select is so intimate. By default it will return the primary keys for the database, but if used with the `keytype` argument, it will return the keys from that `keytype`.

- `select` will retrieve the data as a `data.frame` based on parameters for selected keys `columns` and `keytype` arguments. Users should be warned that if you call `select` and request columns that have multiple matches for your keys, select will return a `data.frame` with one row for each possible match. This has the effect that if you request multiple columns and some of them have a many to one relationship to the keys, things will continue to multiply accordingly. So it's not a good idea to request a large number of columns unless you know that what you are asking for should have a one to one relationship with the initial set of keys. In general, if you need to retrieve a column (like GO) that has a many to one relationship to the original keys, it is most useful to extract that separately.
- `mapIds` gets the mapped ids (column) for a set of keys that are of a particular keytype. Usually returned as a named character vector, a list or even a SimpleCharacterList.
- `saveDb` will take an `AnnotationDb` object and save the database to the file specified by the path passed in to the file argument.
- `loadDb` takes a .sqlite database file as an argument and uses data in the metadata table of that file to return an AnnotationDb style object of the appropriate type.
- `species` shows the genus and species label currently attached to the AnnotationDb objects database.
- `dbfile` gets the database file associated with an object.
- `dbconn` gets the datebase connection associated with an object.
- `taxonomyId` gets the taxonomy ID associated with an object (if available).

ChipDb

Platfom-based or chip-based annotation package are an extremely common kind of Annotation package. The following examples show how to use standard methods to interact with an object of this type.

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

Things loaded along with this package

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

```
 [1] "hgu95av2"              "hgu95av2.db"           "hgu95av2_dbconn"       "hgu95av2_
 [6] "hgu95av2_dbschema"     "hgu95av2ACCNUM"        "hgu95av2ALIAS2PROBE"   "hgu95av2
```

```
[11] "hgu95av2CHRLOC"         "hgu95av2CHRLOCEND"      "hgu95av2ENSEMBL"        "hgu95av2
[16] "hgu95av2ENZYME"         "hgu95av2ENZYME2PROBE"   "hgu95av2GENENAME"       "hgu95av2
[21] "hgu95av2GO2PROBE"       "hgu95av2MAP"            "hgu95av2MAPCOUNTS"      "hgu95av2
[26] "hgu95av2ORGPKG"         "hgu95av2PATH"           "hgu95av2PATH2PROBE"     "hgu95av2
[31] "hgu95av2PMID2PROBE"     "hgu95av2PROSITE"        "hgu95av2REFSEQ"         "hgu95av2
[36] "hgu95av2UNIPROT"
```

These packages appear to contain a lot of data but it is an illusion.

```
> library(hgu95av2.db)
> hgu95av2()
> hgu95av2_dbInfo()
> hgu95av2GENENAME
> hgu95av2_dbschema()
```

Use `columns()` to see possible values for columns.

```
columns(hgu95av2.db)
```

```
 [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTRE
 [9] "EVIDENCEALL"  "GENENAME"     "GO"           "GOALL"        "IPI"          "MAP"
[17] "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"         "PROBEID"      "PROSI
[25] "UCSCKG"       "UNIGENE"      "UNIPROT"
```

Use `help("xxx")` to see the description of columns.

```
help('SYMBOL')
```

Use `keytypes()` to see possible values for keytypes. In reality, some kinds of values make poor keys and so this list is shorter than that of above.

```
keytypes(hgu95a.db)
```

```
 [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTRE
 [9] "EVIDENCEALL"  "GENENAME"     "GO"           "GOALL"        "IPI"          "MAP"
```

```
[17] "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"         "PROBEID"      "PROSI
[25] "UCSCKG"       "UNIGENE"      "UNIPROT"
```

Use `keys()` to extract some sample keys back. (default if the primary key.)

```
head(keys(hgu95av2.db))
```

```
[1] "1000_at"   "1001_at"   "1002_f_at" "1003_s_at" "1004_at"   "1005_at"
```

Or for a particular keytype.

```
head(keys(hgu95av2.db, keytype='SYMBOL'))
```

```
[1] "A1BG"  "A2M"   "A2MP1" "NAT1"  "NAT2"  "NATP"
```

Use `select()` to retrieve data.

```
#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                                C-X-C motif chemokine receptor 5
5   1004_at   CXCR5                                C-X-C motif chemokine receptor 5
6   1005_at   DUSP1                                  dual specificity phosphatase 1
```

If one wants to get only one column of data, `mapIds` can be used.

```
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" "tyrosine kinase wit
                                                  1002_f_at
                 "cytochrome P450 family 2 subfamily C member 19"
                                                    1004_at
                              "C-X-C motif chemokine receptor 5"
```

OrgDb

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.<Ab>.<id>.db` (e.g. `org.Sc.sgd.db`) where `<Ab>` is a 2-letter abbreviation of the organism (e.g. `Sc` for *Saccharomyces cerevisiae*) and `<id>` is an abbreviation (in lower-case) describing the type of cen- tral identifier (e.g. `sgd` for gene identifiers assigned by the Saccharomyces Genome Database, or `eg` for Entrez Gene ids).

Using `OrgDb` packages is just like using `ChipDb` packages.

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

> columns(org.Hs.eg.db)
 [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTR
 [9] "EVIDENCEALL"  "GENENAME"     "GO"           "GOALL"        "IPI"          "MAP"
[17] "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"         "PROSITE"      "REFS
[25] "UNIGENE"      "UNIPROT"

> keytypes(org.Hs.eg.db)
 [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTR
 [9] "EVIDENCEALL"  "GENENAME"     "GO"           "GOALL"        "IPI"          "MAP"
[17] "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"         "PROSITE"      "REFS
[25] "UNIGENE"      "UNIPROT"

> head(keys(org.Hs.eg.db))
[1] "1"  "2"  "3"  "9"  "10" "11"
```

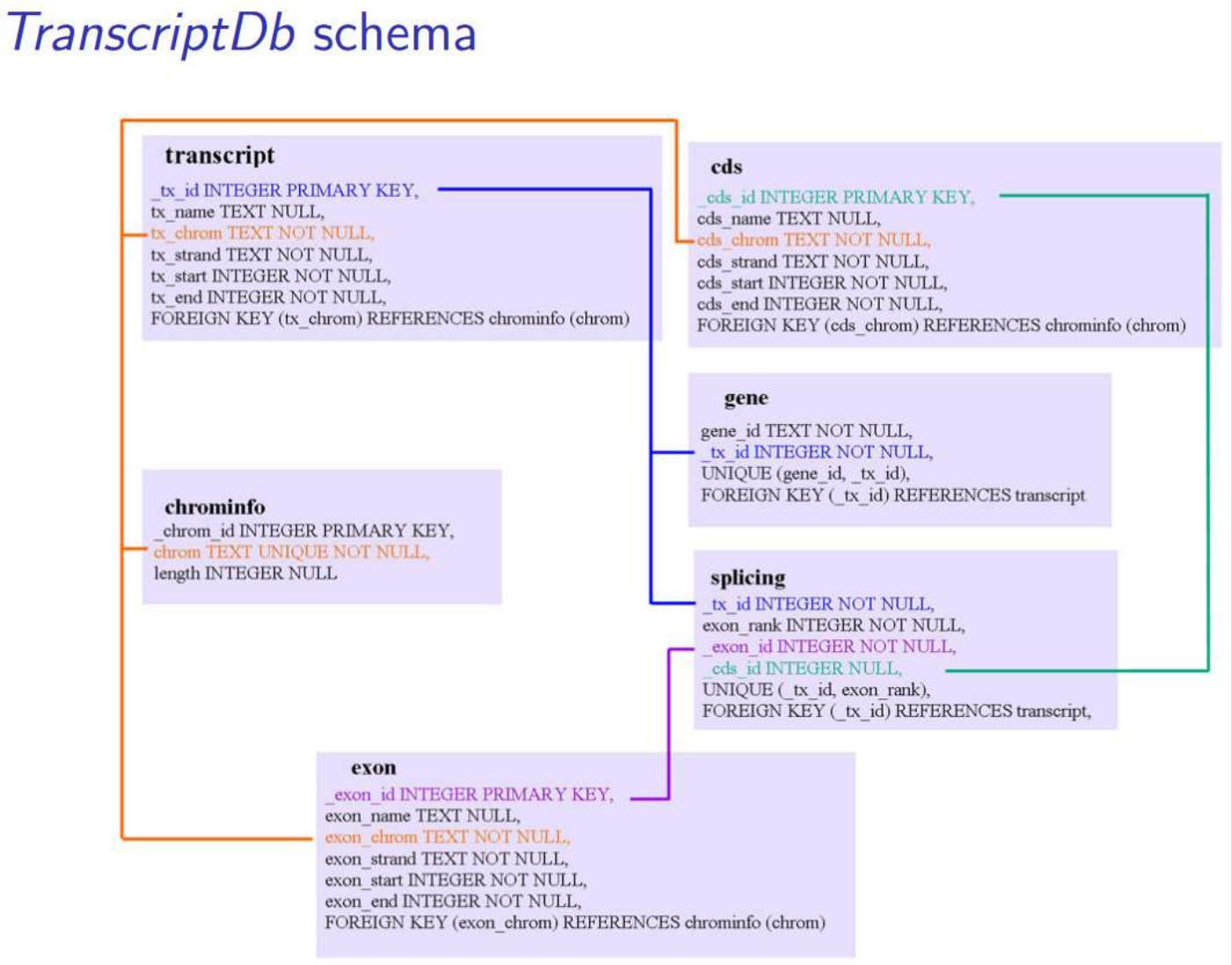
GO.db

```
library(GO.db)

> GO.db
GODb object:
| GOSOURCENAME: Gene Ontology
| GOSOURCEURL: ftp://ftp.geneontology.org/pub/go/godatabase/archive/latest-lite/
| GOSOURCEDATE: 2018-Mar28
| Db type: GODb
| package: AnnotationDbi
| DBSCHEMA: GO_DB
| GOEGSOURCEDATE: 2018-Apr4
| GOEGSOURCENAME: Entrez Gene
| GOEGSOURCEURL: ftp://ftp.ncbi.nlm.nih.gov/gene/DATA
| DBSCHEMAVERSION: 2.1

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

> columns(GO.db)
[1] "DEFINITION" "GOID"       "ONTOLOGY"   "TERM"

> keytypes(GO.db)
[1] "DEFINITION" "GOID"       "ONTOLOGY"   "TERM"

> head(keys(GO.db))
[1] "GO:0000001" "GO:0000002" "GO:0000003" "GO:0000006" "GO:0000007" "GO:0000009"

> head(keys(GO.db, keytype='ONTOLOGY'))
[1] "BP"        "CC"        "MF"        "universal"

> head(keys(GO.db, keytype='TERM'))
[1] "mitochondrion inheritance"                  "mitochondrial genome maintenance"
[3] "reproduction"                               "ribosome biogenesis"
[5] "protein binding involved in protein folding" "unfolded protein binding"

> keys <- head(keys(GO.db))
> select(GO.db, keys=keys, columns=c("TERM","ONTOLOGY"), keytype="GOID")
'select()' returned 1:1 mapping between keys and columns
        GOID                                                TERM ONTOLOGY
1 GO:0000001                                mitochondrion inheritance       BP
2 GO:0000002                          mitochondrial genome maintenance       BP
```

```
3 GO:0000003                                                reproduction       BP
4 GO:0000006    high-affinity zinc transmembrane transporter activity       MF
5 GO:0000007 low-affinity zinc ion transmembrane transporter activity       MF
6 GO:0000009                   alpha-1,6-mannosyltransferase activity       MF
```

TxDb

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.



This class maps the 5′ and 3′ untranslated regions (UTRs), protein coding sequences (CDSs) and exons for a set of mRNA transcripts to their associated genome. `TxDb` objects have numerous accessors

functions to allow such features to be retrieved individually or grouped together in a way that reflects the underlying biology.

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.

Package `GenomicFeatures` contain a set of tools and methods to make and manipulate transcript centric annotation.

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene  #shorthand (for convenience
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

> class(txdb)
[1] "TxDb"
attr(,"package")
[1] "GenomicFeatures"
```

In addition to accessors via `select`, `TxDb` objects also provide access via the more familiar `transcripts`, `exons`, `cds`, `transcriptsBy`, `exonsBy` and `cdsBy` methods, and they will return `GRanges` objects.

The 'ungrouped' functions `transcripts`, `exons`, `cds`, `genes` and `promoters` return the coordinate information as a `GRanges` object.

```
GR = transcripts(txdb)
> GR[1:3]
GRanges object with 3 ranges and 2 metadata columns:
      seqnames      ranges strand |     tx_id     tx_name
         <Rle>   <IRanges>  <Rle> | <integer> <character>
  [1]     chr1 11874-14409      + |         1  uc001aaa.3
  [2]     chr1 11874-14409      + |         2  uc010nxq.1
  [3]     chr1 11874-14409      + |         3  uc010nxr.1
  -------
  seqinfo: 93 sequences (1 circular) from hg19 genome

> length(GR)
[1] 82960
```

The 'grouped' function `transcriptsBy`, `exonsBy`, `cdsBy`, `intronsByTranscript`, `fiveUTRsByTranscript` and `threeUTRsByTranscript` extract genomic features of a given type grouped based on another type of genomic feature.

```
> GRList <- transcriptsBy(txdb, by = "gene")
> GRList
GRangesList object of length 23459:
$1
GRanges object with 2 ranges and 2 metadata columns:
      seqnames            ranges strand |     tx_id     tx_name
         <Rle>         <IRanges>  <Rle> | <integer> <character>
  [1]    chr19 58858172-58864865      - |     70455  uc002qsd.4
  [2]    chr19 58859832-58874214      - |     70456  uc002qsf.2

$10
GRanges object with 1 range and 2 metadata columns:
      seqnames            ranges strand | tx_id    tx_name
  [1]     chr8 18248755-18258723      + | 31944 uc003wyw.1

```

```
$100
GRanges object with 1 range and 2 metadata columns:
      seqnames            ranges strand | tx_id    tx_name
  [1]    chr20 43248163-43280376      - | 72132 uc002xmj.3

...
<23456 more elements>
-------
seqinfo: 93 sequences (1 circular) from hg19 genome
```

The `transcriptsBy` function returns a `GRangesList` class object. The `show` method for a `GRangesList` object will display as a list of `GRanges` objects. And, at the bottom the seqinfo will be displayed once for the entire list.

Then standard `GRanges` and `GRangesList` accessors can be used to deal with the returnings. And one can also leverage many nice `IRanges` methods.

EnsDb

Similar to the `TxDb` objects/packages, `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 implemented for `EnsDb` objects allow also the use of the `EnsDb` specific filtering framework to retrieve only selected information from the database.

```
library(EnsDb.Hsapiens.v86)
edb = EnsDb.Hsapiens.v86

> class(edb)
[1] "EnsDb"
attr(,"package")
[1] "ensembldb"
```

`key()` function has an additional `filter` parameter, which accepts `AnnotationFilter` object.

```
keys <- head(keys(edb, keytype="GENEID"))

```

```
keys(edb, filter=list(GeneBiotypeFilter("lincRNA"), SeqNameFilter("Y")))
```

```
 [1] "ENSG00000129816" "ENSG00000129845" "ENSG00000131538" "ENSG00000147753" "ENSG0000
 [8] "ENSG00000183385" "ENSG00000184991" "ENSG00000185700" "ENSG00000212855" "ENSG0000
[15] "ENSG00000223641" "ENSG00000224075" "ENSG00000224989" "ENSG00000225516" "ENSG0000
[22] "ENSG00000227439" "ENSG00000228240" "ENSG00000228296" "ENSG00000228379" "ENSG0000
[29] "ENSG00000229308" "ENSG00000229643" "ENSG00000230663" "ENSG00000231141" "ENSG0000
[36] "ENSG00000233522" "ENSG00000233699" "ENSG00000233864" "ENSG00000235059" "ENSG0000
[43] "ENSG00000237069" "ENSG00000237563" "ENSG00000239225" "ENSG00000240450" "ENSG0000
[50] "ENSG00000277930" "ENSG00000278847" "ENSG00000280961"
```

`keys` in `mapIds` and `select` also accepts `AnnotationFilter` object.

```
txs <- select(edb, keys=list(GeneBiotypeFilter("lincRNA"), SeqNameFilter("Y")), column

> head(txs, n=3)
             TXID TXSEQSTART TXBIOTYPE GENEBIOTYPE SEQNAME
1 ENST00000250776    6390431   lincRNA     lincRNA       Y
2 ENST00000250805    9753156   lincRNA     lincRNA       Y
3 ENST00000253838   22439593   lincRNA     lincRNA       Y
```

Other Questions

Question: Difference between GO.db, biomaRt, and org.Hs.eg.db in GO annotations

> `GO.db` and `org.Hs.eg.db` are copies of the GO annotations. `GO.db` is updated every 6 months with each release of Bioconductor. `org.Hs.eg.db` is also updated at the same time and using `GO.db`. `biomaRt` connects to the server where the informations is stored, so it will be the most up to date. If you want a stable release you can use either `GO.db` or `org.Hs.eg.db`, if you want the most up to date (from yesterday) data every time you do an analysis you can use `biomaRt`.

Question: org.Hs.eg.db - hg38 build?

> The orgDb packages don’ t really contain any positional annotation. They used to, but these days you will be directed to a TxDb package if you try to get positional info. And the TxDb have the build in the package name. The orgDb packages mostly contain mappings between various

> databases and some functional annotation, none of which is based on any build. In fact, most of that stuff is updated weekly or monthly, so the orgDb packages get outdated to a certain extent rather quickly.

Change log

- 20180918: create the note.

R # Bioconductor # Bioconductor package # annotation package

‹ Biological ID Conversion

Calculate FRiP score ›