**AnnotationDbi: Introduction To Bioconductor Annotation Packages**

*Marc Carlson*

March 6, 2019

---

**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 bimaps from the "db" annotation packages` vignette in the `AnnotationDbi` package is a key reference for learning 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.

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 `AnnotationDb` 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 a 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 `AnnotationDb` objects described below.

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:

```r
suppressPackageStartupMessages({
  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")
```

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
## | EGSOURCENNAME: Entrez Gene
## | CENTRALID: ENTREZID
## | TAXID: 9606
## | GOSOURCENAME: Gene Ontology
## | GOSOURCENDATE: 20150919
## | GOEGSOURCENNAME: Entrez Gene
## | KEGGSOURL: KEGG GENOME
```
If we want to know what kinds of data are retrievable via `select`, then we should use the `columns` method like this:

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

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.

```r
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.

```r
keytypes(hgu95av2.db)
```
If we want to extract some sample keys of a particular type, we can use the `keys` method.

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

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).

```r
# 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")
```

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:

```r
# 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")
```
## 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
## "C-X-C motif chemokine receptor 5"
## 1004_at
## "C-X-C motif chemokine receptor 5"
## 1005_at
## "dual specificity phosphatase 1"

### 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.<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 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:**

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

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")

head(resultTable)
head(annots)
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:

```r
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

## 1 GO:0042254
## 2 GO:0044183

## 1 A cellular process that results in the biosynthesis of constituent macromolecules, assembly, and arrangement of constituent parts of ribosome subunits; includes transport to the sites of protein synthesis.
## 2 Interacting selectively and non-covalently with any protein or protein complex (a complex of two or more proteins that may include other nonprotein molecules).

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:

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

## Loading required package: GenomicFeatures

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

```r
taxdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
taxdb
```

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

```r
columns(txdb)
```

```
[1] "CDSCHROM"  "CDSEND"   "CDSID"    "CDSNAME"  "CDSSTART"
[6] "CDSSTRAND" "EXONCHROM" "EXONEND"  "EXONID"    "EXONNAME"
[11] "EXONRANK"  "EXONSTART" "EXONSTRAND" "GENEID"    "TXCHROM"
[16] "TXEND"     "TXID"     "TXNAME"   "TXSTART"   "TXSTRAND"
[21] "TXTYPE"
```

```r
keytypes(txdb)
```

```
[1] "CDSID"  "CDSNAME"  "EXONID"  "EXONNAME"  "GENEID"  "TXID"
[7] "TXNAME"
```

```r
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
```
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.

## 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.

```r
library(EnsDb.Hsapiens.v75)
## Loading required package: ensembldb
## Loading required package: AnnotationFilter
##
## Attaching package: 'ensembldb'
##
## The following object is masked from 'package:stats':
##
## filter
edb <- EnsDb.Hsapiens.v75
edb
## EnsDb for Ensembl:
## |Backend: SQLite
```
AnnotationDbi: Introduction To Bioconductor Annotation Packages

```r
## |Db type: EnsDb
## |Type of Gene ID: Ensembl Gene ID
## |Supporting package: ensembldb
## |Db created by: ensembldb package from Bioconductor
## |script_version: 0.3.0
## |Creation time: Thu May 18 09:15:45 2017
## |ensembl_version: 75
## |ensembl_host: localhost
## |Organism: homo_sapiens
## |taxonomy_id: 9606
## |genome_build: GRCh37
## |DBSCHEMAVERSION: 2.0
## |No. of genes: 64102.
## |No. of transcripts: 215647.
## |Protein data available.

## List all columns

```r
columns(edb)
```

```r
## [1] "ENTREZID"  "EXONID"  "EXONIDX"
## [4] "EXONSEQEND" "EXONSEQSTART" "GENEBIOTYPE"
## [7] "GENEID"     "GENENAME"  "GENESEQEND"
## [10] "GENESEQSTART" "INTERPROACCESSION" "ISCIRCULAR"
## [13] "PROTDOMEND"  "PROTDOMSTART"  "PROTEINDOMAINID"
## [16] "PROTEINDOMAINSOURCE" "PROTEINID" "PROTEINSEQUENCE"
## [19] "SEQCOORDSYSTEM" "SEQLENGTH" "SEQNAME"
## [22] "SEQSTRAND" "SYMBOL" "TXBIOTYPE"
## [25] "TXCDSSEQEND" "TXCDSSEQSTART" "TXID"
## [28] "TXNAME" "TXSEQEND" "TXSEQSTART"
## [31] "UNIPROTDB" "UNIPROTID" "UNIPROTMAPPINGTYPE"
```

## List all keytypes

```r
keytypes(edb)
```

```r
## [1] "ENTREZID" "GENEID" "PROTEINDOMAINID" "PROTEINDOMAINSOURCE"
## [4] "DEXONID" "GENENAME" "PROTEINID" "SEQCOORDSYSTEM"
## [7] "GENESEQEND" "PROTDOMEND" "PROTDOME" "SEQNAME"
## [10] "INTERPROACCESSION" "PROTDOMSTART" "Symbol" "SEQSTRAND"
## [13] "PROTEINDOMAINID" "TXBIOTYPE" "TXID" "TXNAME"
## [16] "UNIPROTID"
```

## Get the first

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

## Get the data

```r
select(edb, keys=keys, columns=c("TXID", "TXSEQSTART", "TXBIOTYPE"), keytype="GENEID")
```

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.

```r
## Retrieve all gene IDs of all lincRNAs encoded on chromosome Y
linkY <- keys(edb,
           filter=list(GeneBiotypeFilter("lincRNA"), SeqNameFilter("Y")))
length(linkY)
```
We get now all transcripts for these genes.
```r
txs <- select(edb, keys=linkY, columns=c("TXID", "TXSEQSTART", "TXBIOTYPE"), keytype="GENEID")
```
```
> nrow(txs)
1 [1] 66
```
Alternatively, we could specify/pass the filters with the keys argument.
```r
txs <- select(edb, keys=list(GeneBiotypeFilter("lincRNA"), SeqNameFilter("Y")), columns=c("TXID", "TXSEQSTART", "TXBIOTYPE"))
```
```
> nrow(txs)
1 [1] 66
```

The version number of R and packages loaded for generating the vignette were:
```r
## R Under development (unstable) (2019-01-21 r75999)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.1 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.9-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.9-bioc/R/lib/libRlapack.so
##
## locale:
## [1] LC_CTYPE=en_US.UTF-8    LC_NUMERIC=C
## [2] LC_CTYPE=en_US.UTF-8    LC_CTYPE=C
## [3] LC_TIME=en_US.UTF-8     LC_COLLATE=C
## [4] LC_TIME=en_US.UTF-8     LC_CTYPE=C
## [5] LC_MONETARY=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
## [6] LC_MONETARY=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
## [7] LC_PAPER=en_US.UTF-8    LC_NAME=C
## [8] LC_ADDRESS=C            LC_ADDRESS=C
## [9] LC_TELEPHONE=C          LC_TELEPHONE=C
## [10] LC_MEASUREMENT=en_US.UTF-8 LC_MEASUREMENT=en_US.UTF-8
## [12] LC_IDENTIFICATION=C    LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4   stats    graphics grDevices utils    datasets
## [8] methods   base
##
## other attached packages:
## [1] EnsDb.Hsapiens.v75_2.99.0
## [2] ensembldb_2.7.10
## [3] AnnotationFilter_1.7.0
## [5] GenomicFeatures_1.35.7
## [6] GenomicRanges_1.35.1
AnnotationDbi: Introduction To Bioconductor Annotation Packages

```r
## [7] GenomeInfoDb_1.19.2
## [8] G0.db_3.7.0
## [9] hgu95av2.db_3.2.3
## [10] AnnotationForge_1.25.0
## [11] org.Hs.eg.db_3.7.0
## [12] AnnotationDbi_1.45.1
## [13] IRanges_2.17.4
## [14] S4Vectors_0.21.10
## [15] Biobase_2.43.1
## [16] BiocGenerics_0.29.2
## [17] DBI_1.0.0
## [18] knitr_1.21
##
## loaded via a namespace (and not attached):
## [1] SummarizedExperiment_1.13.0 progress_1.2.0
## [3] xfun_0.5 lattice_0.20-38
## [5] htmltools_0.3.6 rtracklayer_1.43.3
## [7] yam1_2.2.0 blob_1.1.1
## [9] XML_3.98-1.19 rlang_0.3.1
## [11] BiocParallel_1.17.17 bit64_0.9-7
## [13] matrixStats_0.54.0 GenomeInfoDbData_1.2.0
## [15] ProtGenerics_1.15.0 stringr_1.4.0
## [17] zlibbioc_1.29.0 Biostrings_2.51.2
## [19] memoise_1.1.0 evaluate_0.13
## [21] bioMaRt_2.39.2 curl_3.3
## [23] highr_0.7 Rcpp_1.0.0
## [25] BiocManager_1.30.4 DelayedArray_0.9.8
## [27] XVector_0.23.0 bit_1.1-14
## [29] Rsamtools_1.99.2 BiocStyle_2.11.0
## [31] hms_0.4.2 digest_0.6.18
## [33] stringi_1.3.1 grid_3.6.0
## [35] tools_3.6.0 bitops_1.0-6
## [37] magrittr_1.5 lazyeval_0.2.1
## [39] RTools_1.95-4.12 RSQLite_2.1.1
## [41] crayon_1.3.4 pkgconfig_2.0.2
## [43] Matrix_1.2-15 prettyunits_1.0.2
## [45] assertthat_0.2.0 rmarkdown_1.11
## [47] httr_1.4.0 R6_2.4.0
## [49] GenomicAlignments_1.19.1 compiler_3.6.0
```