AnnotationDbi: Introduction To Bioconductor Annotation Packages

Marc Carlson

March 18, 2015

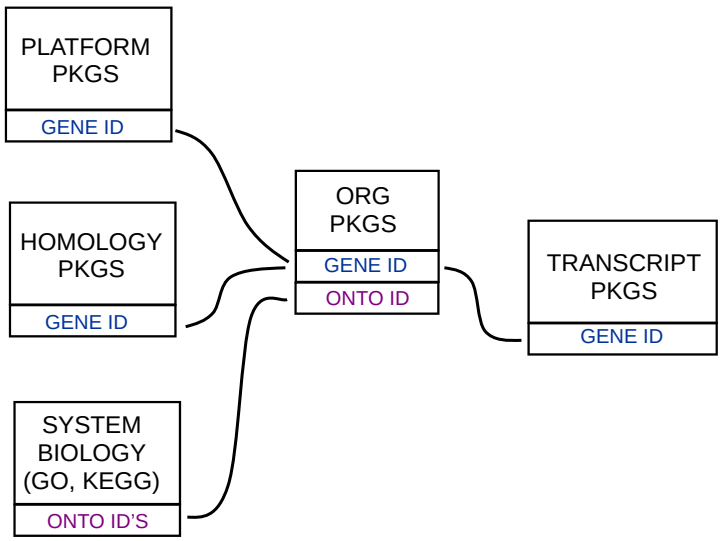

Figure 1: Annotation Packages: the big picture

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*
- 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 conventient 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 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`.

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"            "hgu95av2CHRLENGTHS"
##  [7] "hgu95av2CHRLOC"         "hgu95av2CHRLOCEND"      "hgu95av2ENSEMBL"
## [10] "hgu95av2ENSEMBL2PROBE" "hgu95av2ENTREZID"       "hgu95av2ENZYME"
## [13] "hgu95av2ENZYME2PROBE"  "hgu95av2GENENAME"       "hgu95av2GO"
## [16] "hgu95av2GO2ALLPROBES"  "hgu95av2GO2PROBE"       "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: 2014-Sep19
## | 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: 20140913
## | GOEGSOURCEDATE: 2014-Sep19
## | 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: 2014-Aug6
## | ENSOURCENAME: Ensembl
## | ENSOURCEURL: ftp://ftp.ensembl.org/pub/current_fasta
## | UPSOURCENAME: Uniprot
## | UPSOURCEURL: http://www.UniProt.org/
## | UPSOURCEDATE: Tue Sep 23 15:46:34 2014

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

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

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

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

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

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")
```

```
##     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, polypeptide 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.

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.¡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:

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

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

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

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

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

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

```
## Warning in .generateExtraRows(tab, keys, jointype): 'select' resulted in 1:many mapping
between keys and return rows
```

```
##    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 hlep 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")
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  SLCO4A1-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      TTTY13B
##                                    GENENAME
```

```
## 1                    SLCO4A1 antisense RNA 1
## 2 ADP-ribosylation factor-like 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")
```

```
##         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:  GenomicRanges
```

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

```
## TxDb object:
## | Db type: TxDb
## | Supporting package: GenomicFeatures
## | Data source: UCSC
## | Genome: hg19
## | Organism: Homo sapiens
## | 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: 2014-09-26 11:16:12 -0700 (Fri, 26 Sep 2014)
## | GenomicFeatures version at creation time: 1.17.17
## | RSQLite version at creation time: 0.11.4
## | DBSCHEMAVERSION: 1.0
```

```
columns(txdb)
```

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

```
keytypes(txdb)
```

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

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

```
## Warning in .generateExtraRows(tab, keys, jointype): 'select' resulted in 1:many mapping
between keys and return rows
```

```
##       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.

The version number of R and packages loaded for generating the vignette were:

```
## R version 3.1.3 (2015-03-09)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.9.5 (Mavericks)
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets  methods
## [9] base
##
## other attached packages:
##  [1] TxDb.Hsapiens.UCSC.hg19.knownGene_3.0.0 GenomicFeatures_1.18.3
##  [3] GenomicRanges_1.18.4                    GO.db_3.0.0
##  [5] hgu95av2.db_3.0.0                       AnnotationForge_1.8.2
##  [7] org.Hs.eg.db_3.0.0                      RSQLite_1.0.0
##  [9] DBI_0.3.1                               AnnotationDbi_1.28.2
## [11] GenomeInfoDb_1.2.4                      IRanges_2.0.1
## [13] S4Vectors_0.4.0                         Biobase_2.26.0
## [15] BiocGenerics_0.12.1                     knitr_1.9
##
## loaded via a namespace (and not attached):
##  [1] BBmisc_1.9              BatchJobs_1.5           BiocParallel_1.0.3
##  [4] BiocStyle_1.4.1         Biostrings_2.34.1       GenomicAlignments_1.2.2
##  [7] RCurl_1.95-4.5          Rsamtools_1.18.3        XML_3.98-1.1
## [10] XVector_0.6.0           base64enc_0.1-2         biomaRt_2.22.0
## [13] bitops_1.0-6            brew_1.0-6              checkmate_1.5.1
## [16] codetools_0.2-11        digest_0.6.8            evaluate_0.5.5
## [19] fail_1.2                foreach_1.4.2           formatR_1.0
## [22] highr_0.4               iterators_1.0.7         rtracklayer_1.26.2
## [25] sendmailR_1.2-1         stringr_0.6.2           tools_3.1.3
## [28] zlibbioc_1.12.0
```