How to use bimaps from the ".db" annotation packages

Marc Carlson, Herve Pages, Seth Falcon, Nianhua Li

March 18, 2015

1 Introduction

1.0.1 Purpose

AnnotationDbi is used primarily to create mapping objects that allow easy access from R to underlying annotation databases. As such, it acts as the R interface for all the standard annotation packages. Underlying each AnnotationDbi supported annotation package is at least one (and often two) annotation databases. AnnotationDbi also provides schemas for theses databases. For each supported model organism, a standard gene centric database is maintained from public sources and is packaged up as an appropriate organism or "org" package.

1.0.2 Database Schemas

For developers, a lot of the benefits of having the information loaded into a real database will require some knowledge about the database schema. For this reason the schemas that were used in the creation of each database type are included in AnnotationDbi. The currently supported schemas are listed in the DBschemas directory of AnnotationDbi. But it is also possible to simply print out the schema that a package is currently using by using its "_dbschema" method.

There is one schema/database in each kind of package. These schemas specify which tables and indices will be present for each package of that type. The schema that a particular package is using is also listed when you type the name of the package as a function to obtain quality control information.

The code to make most kinds of the new database packages is also included in AnnotationDbi. Please see the vignette on SQLForge for more details on how to make additional database packages.

1.0.3 Internal schema Design of org packages

The current design of the organism packages is deliberately simple and gene centric. Each table in the database contains a unique kind of information and also an internal identifier called _id. The internal _id has no meaning outside of the context of a single database. But _id does connect all the data within a single database.

As an example if we wanted to connect the values in the genes table with the values in the kegg table, we could simply join the two tables using the internal _id column. It is very important to note however that _id does not have any absolute significance. That is, it has no meaning outside of the context of the database where it is used. It is tempting to think that an _id could have such significance because within a single database,

it looks and behaves similarly to an entrez gene ID. But _id is definitely NOT an entrez gene ID. The entrez gene IDs are in another table entirely, and can be connected to using the internal _id just like all the other meaningful information inside these databases. Each organism package is centered around one type of gene identifier. This identifier is found as the gene_id field in the genes table and is both the central ID for the database as well as the foreign key that chip packages should join to.

The chip packages are 'lightweight', and only contain information about the basic probe to gene mapping. You might wonder how such packages can provide access to all the other information that they do. This is possible because all the other data provided by chip packages comes from joins that are performed by AnnotationDbi behind the scenes at run time. All chip packages have a dependency on at least one organism package. The name of the organism package being depended on can be found by looking at its "ORGPKG" value. To learn about the schema from the appropriate organism package, you will need to look at the " _dbschema" method for that package. In the case of the chip packages, the gene_id that in these packages is mapped to the probe_ids, is used as a foreign key to the appropriate organism package.

Specialized packages like the packages for GO and KEGG, will have their own schemas but will also adhere to the use of an internal _id for joins between their tables. As with the organism packages, this _id is not suitable for use as a foreign key.

For a complete listing of the different schemas used by various packages, users can use the `available.dbschemas` function. This list will also tell you which model organisms are supported.

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

## Loading required package:  org.Hs.eg.db
## Loading required package:  AnnotationDbi
## Loading required package:  stats4
## Loading required package:  BiocGenerics
## Loading required package:  parallel
##
## Attaching package:  'BiocGenerics'
##
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ, clusterExport,
##     clusterMap, parApply, parCapply, parLapply, parLapplyLB, parRapply,
##     parSapply, parSapplyLB
##
## The following object is masked from 'package:stats':
##
##     xtabs
##
## The following objects are masked from 'package:base':
##
##     Filter, Find, Map, Position, Reduce, anyDuplicated, append, as.data.frame,
##     as.vector, cbind, colnames, do.call, duplicated, eval, evalq, get,
##     intersect, is.unsorted, lapply, mapply, match, mget, order, paste, pmax,
##     pmax.int, pmin, pmin.int, rank, rbind, rep.int, rownames, sapply, setdiff,
##     sort, table, tapply, union, unique, unlist, unsplit
##
```

```
## Loading required package:  Biobase
## Welcome to Bioconductor
##
##    Vignettes contain introductory material; view with 'browseVignettes()'.  To
##    cite Bioconductor, see 'citation("Biobase")', and for packages
##    'citation("pkgname")'.
##
## Loading required package:  GenomeInfoDb
## Loading required package:  S4Vectors
## Loading required package:  IRanges
##
## Attaching package:  'AnnotationDbi'
##
## The following object is masked from 'package:GenomeInfoDb':
##
##    species
##
## Loading required package:  DBI

require(AnnotationForge)

## Loading required package:  AnnotationForge

available.dbschemas()
```

2   Examples

2.0.4   Basic information

The *AnnotationDbi* package provides an interface to SQLite-based annotation packages. Each SQLite-based annotation package (identified by a ".db" suffix in the package name) contains a number of *AnnDbBimap* objects in place of the *environment* objects found in the old-style environment-based annotation packages. The API provided by *AnnotationDbi* allows you to treat the *AnnDbBimap* objects like *environment* instances. For example, the functions [[, get, mget, and ls all behave the same as they did with the older environment based annotation packages. In addition, new methods like [, toTable, subset and others provide some additional flexibility in accessing the annotation data.

```
library("hgu95av2.db")

##
```

The same basic set of objects is provided with the db packages:

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

##   [1] "hgu95av2"            "hgu95av2.db"
##   [3] "hgu95av2ACCNUM"      "hgu95av2ALIAS2PROBE"
##   [5] "hgu95av2CHR"         "hgu95av2CHRLENGTHS"
##   [7] "hgu95av2CHRLOC"      "hgu95av2CHRLOCEND"
```

```
##  [9] "hgu95av2ENSEMBL"        "hgu95av2ENSEMBL2PROBE"
## [11] "hgu95av2ENTREZID"       "hgu95av2ENZYME"
## [13] "hgu95av2ENZYME2PROBE"   "hgu95av2GENENAME"
## [15] "hgu95av2GO"             "hgu95av2GO2ALLPROBES"
## [17] "hgu95av2GO2PROBE"       "hgu95av2MAP"
## [19] "hgu95av2MAPCOUNTS"      "hgu95av2OMIM"
## [21] "hgu95av2ORGANISM"       "hgu95av2ORGPKG"
## [23] "hgu95av2PATH"           "hgu95av2PATH2PROBE"
## [25] "hgu95av2PFAM"           "hgu95av2PMID"
## [27] "hgu95av2PMID2PROBE"     "hgu95av2PROSITE"
## [29] "hgu95av2REFSEQ"         "hgu95av2SYMBOL"
## [31] "hgu95av2UNIGENE"        "hgu95av2UNIPROT"
## [33] "hgu95av2_dbInfo"        "hgu95av2_dbconn"
## [35] "hgu95av2_dbfile"        "hgu95av2_dbschema"
```

Exercise 1
Start an R session and use the `library` function to load the hgu95av2.db software package. Use search() to see that an organism package was also loaded and then use the approriate "_dbschema" methods to the schema for the hgu95av2.db and org.Hs.eg.db packages.

It is possible to call the package name as a function to get some QC information about it.

```
qcdata = capture.output(hgu95av2())
head(qcdata, 20)

##  [1] "Quality control information for hgu95av2:"
##  [2] ""
##  [3] ""
##  [4] "This package has the following mappings:"
##  [5] ""
##  [6] "hgu95av2ACCNUM has 12625 mapped keys (of 12625 keys)"
##  [7] "hgu95av2ALIAS2PROBE has 33755 mapped keys (of 103735 keys)"
##  [8] "hgu95av2CHR has 11540 mapped keys (of 12625 keys)"
##  [9] "hgu95av2CHRLENGTHS has 93 mapped keys (of 93 keys)"
## [10] "hgu95av2CHRLOC has 11474 mapped keys (of 12625 keys)"
## [11] "hgu95av2CHRLOCEND has 11474 mapped keys (of 12625 keys)"
## [12] "hgu95av2ENSEMBL has 11460 mapped keys (of 12625 keys)"
## [13] "hgu95av2ENSEMBL2PROBE has 9814 mapped keys (of 28553 keys)"
## [14] "hgu95av2ENTREZID has 11543 mapped keys (of 12625 keys)"
## [15] "hgu95av2ENZYME has 2125 mapped keys (of 12625 keys)"
## [16] "hgu95av2ENZYME2PROBE has 786 mapped keys (of 975 keys)"
## [17] "hgu95av2GENENAME has 11543 mapped keys (of 12625 keys)"
## [18] "hgu95av2GO has 11245 mapped keys (of 12625 keys)"
## [19] "hgu95av2GO2ALLPROBES has 17214 mapped keys (of 18826 keys)"
## [20] "hgu95av2GO2PROBE has 12920 mapped keys (of 14714 keys)"
```

Alternatively, you can get similar information on how many items are in each of the provided maps by looking at the MAPCOUNTs:

```
hgu95av2MAPCOUNTS
```

To demonstrate the *environment* API, we'll start with a random sample of probe set IDs.

```
all_probes <- ls(hgu95av2ENTREZID)
length(all_probes)
```

```
## [1] 12625
```

```
set.seed(0xa1beef)
probes <- sample(all_probes, 5)
probes
```

```
## [1] "31882_at"   "38780_at"   "37033_s_at" "1702_at"    "31610_at"
```

The usual ways of accessing annotation data are also available.

```
hgu95av2ENTREZID[[probes[1]]]
```

```
## [1] "9136"
```

```
hgu95av2ENTREZID$"31882_at"
```

```
## [1] "9136"
```

```
syms <- unlist(mget(probes, hgu95av2SYMBOL))
syms
```

```
##   31882_at   38780_at 37033_s_at    1702_at   31610_at
##     "RRP9"   "AKR1A1"     "GPX1"    "IL2RA" "PDZK1IP1"
```

The annotation packages provide a huge variety of information in each package. Some common types of information include gene symbols (SYMBOL), GO terms (GO), KEGG pathway IDs (KEGG), ENSEMBL IDs (ENSEMBL) and chromosome start and stop locations (CHRLOC and CHRLOCEND). Each mapping will have a manual page that you can read to describe the data in the mapping and where it came from.

```
?hgu95av2CHRLOC
```

Exercise 2
For the probes in 'probes' above, use the annotation mappings to find the chromosome start locations.

2.0.5   Manipulating Bimap Objects

Many filtering operations on the annotation *Bimap* objects require conversion of the *AnnDbBimap* into a *list*. In general, converting to lists will not be the most efficient way to filter the annotation data when using a SQLite-based package. Compare the following two examples for how you could get the 1st ten elements of the hgu95av2SYMBOL mapping. In the 1st case we have to get the entire mapping into list form, but in the second case we first subset the mapping object itself and this allows us to only convert the ten elements that we care about.

```
system.time(as.list(hgu95av2SYMBOL)[1:10])

## vs:
```

```
system.time(as.list(hgu95av2SYMBOL[1:10]))
```

There are many different kinds of *Bimap* objects in AnnotationDbi, but most of them are of class *AnnDbBimap*. All /RclassBimap objects represent data as a set of left and right keys. The typical usage of these mappings is to search for right keys that match a set of left keys that have been supplied by the user. But sometimes it is also convenient to go in the opposite direction.

The annotation packages provide many reverse maps as objects in the package name space for backwards compatibility, but the reverse mappings of almost any map is also available using `revmap`. Since the data are stored as tables, no extra disk space is needed to provide reverse mappings.

```
unlist(mget(syms, revmap(hgu95av2SYMBOL)))
```

```
##        RRP9       AKR1A1        GPX1       IL2RA    PDZK1IP1
##   "31882_at"   "38780_at" "37033_s_at"  "1702_at"  "31610_at"
```

So now that you know about the `revmap` function you might try something like this:

```
as.list(revmap(hgu95av2PATH)["00300"])
```

```
## $`00300`
## [1] "35870_at" "36132_at"
```

Note that in the case of the PATH map, we don't need to use revmap(x) because hgu95av2.db already provides the PATH2PROBE map:

```
x <- hgu95av2PATH
## except for the name, this is exactly revmap(x)
revx <- hgu95av2PATH2PROBE
revx2 <- revmap(x, objName="PATH2PROBE")
revx2
```

```
## PATH2PROBE map for chip hgu95av2 (object of class "ProbeAnnDbBimap")
```

```
identical(revx, revx2)
```

```
## [1] TRUE
```

```
as.list(revx["00300"])
```

```
## $`00300`
## [1] "35870_at" "36132_at"
```

Note that most maps are reversible with `revmap`, but some (such as the more complex GO mappings), are not. Why is this? Because to reverse a mapping means that there has to be a "value" that will always become the "key" on the newly reversed map. And GO mappings have several distinct possibilities to choose from (GO ID, Evidence code or Ontology). In non-reversible cases like this, AnnotationDbi will usually provide a pre-defined reverse map. That way, you will always know what you are getting when you call `revmap`

While we are on the subject of GO and GO mappings, there are a series of special methods for GO mappings that can be called to find out details about these IDs. `Term,GOID`, `Ontology`, `Definition,Synonym`, and `Secondary` are all useful ways of getting additional information about a particular GO ID. For example:

```
Term("GO:0000018")
```

```
## Loading required package:  GO.db
```

```
##                            GO:0000018
## "regulation of DNA recombination"

Definition("GO:0000018")

##
## "Any process that modulates the frequency, rate or extent of DNA recombination, a DNA metaboli
```

Exercise 3
Given the following set of RefSeq IDs: c("NG_005114","NG_007432","NG_008063"), Find the Entrez Gene IDs that would correspond to those. Then find the GO terms that are associated with those entrez gene IDs.

org.Hs.eg.db packages.

2.0.6   The Contents and Structure of Bimap Objects

Sometimes you may want to display or subset elements from an individual map. A *Bimap* interface is available to access the data in table (*data.frame*) format using [and toTable.

```
head(toTable(hgu95av2GO[probes]))

##   probe_id       go_id Evidence Ontology
## 1  1702_at GO:0006915      TAS       BP
## 2  1702_at GO:0006955      TAS       BP
## 3  1702_at GO:0007166      TAS       BP
## 4  1702_at GO:0008283      TAS       BP
## 5  1702_at GO:0038110      TAS       BP
## 6  1702_at GO:0002437      IEA       BP
```

The toTable function will display all of the information in a *Bimap*. This includes both the left and right values along with any other attributes that might be attached to those values. The left and right keys of the *Bimap* can be extracted using Lkeys and Rkeys. If is is necessary to only display information that is directly associated with the left to right links in a *Bimap*, then the links function can be used. The links returns a data frame with one row for each link in the bimap that it is applied to. It only reports the left and right keys along with any attributes that are attached to the edge between these two values.

Note that the order of the cols returned by toTable does not depend on the direction of the map. We refer to it as an 'undirected method':

```
toTable(x)[1:6, ]

##   probe_id path_id
## 1  1000_at   04010
## 2  1000_at   04012
## 3  1000_at   04062
## 4  1000_at   04114
## 5  1000_at   04150
## 6  1000_at   04270

toTable(revx)[1:6, ]

##   probe_id path_id
```

```
## 1   1000_at    04010
## 2   1000_at    04012
## 3   1000_at    04062
## 4   1000_at    04114
## 5   1000_at    04150
## 6   1000_at    04270
```

Notice however that the Lkeys are always on the left (1st col), the Rkeys always in the 2nd col

For length() and keys(), the result does depend on the direction, hence we refer to these as 'directed methods':

```
length(x)
```

```
## [1] 12625
```

```
length(revx)
```

```
## [1] 229
```

```
allProbeSetIds <- keys(x)
allKEGGIds <- keys(revx)
```

There are more 'undirected' methods listed below:

```
junk <- Lkeys(x)          # same for all maps in hgu95av2.db (except pseudo-map
                          # MAPCOUNTS)
Llength(x)                # nb of Lkeys
```

```
## [1] 12625
```

```
junk <- Rkeys(x)          # KEGG ids for PATH/PATH2PROBE maps, GO ids for
                          # GO/GO2PROBE/GO2ALLPROBES maps, etc...
Rlength(x)                # nb of Rkeys
```

```
## [1] 229
```

Notice how they give the same result for x and revmap(x)

You might be tempted to think that `Lkeys` and `Llength` will tell you all that you want to know about the left keys. But things are more complex than this, because not all keys are mapped. Often, you will only want to know about the keys that are mapped (ie. the ones that have a corresponding Rkey). To learn this you want to use the `mappedkeys` or the undirected variants `mappedLkeys` and `mappedRkeys`. Similarly, the `count.mappedkeys`, `count.mappedLkeys` and `count.mappedRkeys` methods are very fast ways to determine how many keys are mapped. Accessing keys like this is usually very fast and so it can be a decent strategy to subset the mapping by 1st using the mapped keys that you want to find.

```
x = hgu95av2ENTREZID[1:10]
## Directed methods
mappedkeys(x)              # mapped keys
```

```
## [1] "1000_at"    "1001_at"    "1002_f_at" "1003_s_at" "1004_at"
## [6] "1005_at"    "1006_at"    "1008_f_at" "1009_at"
```

```
count.mappedkeys(x)       # nb of mapped keys
```

```
## [1] 9
```

```
## Undirected methods
mappedLkeys(x)            # mapped left keys

## [1] "1000_at"    "1001_at"    "1002_f_at" "1003_s_at" "1004_at"
## [6] "1005_at"    "1006_at"    "1008_f_at" "1009_at"

count.mappedLkeys(x)     # nb of mapped Lkeys

## [1] 9
```

If you want to find keys that are not mapped to anything, you might want to use isNA.

```
y = hgu95av2ENTREZID[isNA(hgu95av2ENTREZID)]     # usage like is.na()
Lkeys(y)[1:4]

## [1] "1007_s_at" "1047_s_at" "1089_i_at" "108_g_at"
```

Exercise 4
*How many probesets do not have a GO mapping for the hgu95av2.db package? How many have no mapping?
Find a probeset that has a GO mapping. Now look at the GO mappings for this probeset in table form.*

2.0.7 Some specific examples

Lets use what we have learned to get information about the probes that are are not assigned to a chromosome:

```
x <- hgu95av2CHR
Rkeys(x)

##  [1] "19" "12" "8"  "14" "3"  "2"  "17" "16" "9"  "X"  "6"  "1"  "7"
## [14] "10" "11" "22" "5"  "18" "15" "Y"  "20" "21" "4"  "13" "MT" "Un"

chroms <- Rkeys(x)[23:24]
chroms

## [1] "4"  "13"

Rkeys(x) <- chroms
toTable(x)

##         probe_id chromosome
## 1      1029_s_at          4
## 2        1036_at          4
## 3        1058_at         13
## 4        1065_at         13
## 5        1115_at          4
## 6        1189_at         13
## 7        1198_at         13
## 8        1219_at          4
## 9      1220_g_at          4
## 10       1249_at          4
## 11       1285_at          4
## 12       1303_at          4
```

```
## 13      1325_at          4
## 14    1348_s_at         13
## 15    1369_s_at          4
## 16      1377_at          4
## 17    1378_g_at          4
## 18    1451_s_at         13
## 19      1503_at         13
## 20    1507_s_at          4
## 21    1527_s_at         13
## 22      1528_at         13
## 23      1529_at         13
## 24    1530_g_at         13
## 25      1531_at         13
## 26    1532_g_at         13
## 27    1538_s_at          4
## 28      1542_at          4
## 29    1545_g_at         13
## 30      1567_at         13
## 31    1570_f_at         13
## 32    1571_f_at         13
## 33      1593_at          4
## 34      1597_at         13
## 35    1598_g_at         13
## 36       159_at          4
## 37      1600_at          4
## 38      1604_at          4
## 39    1605_g_at          4
## 40      1616_at         13
## 41      1624_at          4
## 42    1629_s_at          4
## 43      1670_at         13
## 44    1672_f_at         13
## 45      1679_at          4
## 46      1708_at          4
## 47    1709_g_at          4
## 48       170_at         13
## 49      1720_at          4
## 50    1721_g_at          4
## 51      1731_at          4
## 52      1732_at          4
## 53      1819_at         13
## 54    1828_s_at          4
## 55      1836_at          4
## 56    1883_s_at          4
## 57    1888_s_at          4
## 58      1900_at         13
## 59    1905_s_at         13
```

```
## 60      1913_at          4
## 61      1914_at         13
## 62      1931_at         13
## 63    1934_s_at          4
## 64      1943_at          4
## 65      1954_at          4
## 66      1963_at         13
## 67    1964_g_at         13
## 68      1987_at          4
## 69      1988_at          4
## 70      1989_at         13
## 71    1990_g_at         13
## 72    2044_s_at         13
## 73      2062_at          4
## 74    2092_s_at          4
## 75       214_at          4
## 76     215_g_at          4
## 77       252_at         13
## 78     253_g_at         13
## 79       260_at          4
## 80     281_s_at          4
## 81     31314_at          4
## 82     31320_at         13
## 83     31333_at          4
## 84     31345_at          4
## 85     31349_at          4
## 86     31356_at          4
## 87   31382_f_at          4
## 88     31404_at         13
## 89     31408_at          4
## 90     31464_at         13
## 91   31465_g_at         13
## 92   31516_f_at         13
## 93     31543_at          4
## 94     31562_at         13
## 95     31584_at         13
## 96     31628_at         13
## 97   31631_f_at          4
## 98   31639_f_at         13
## 99   31640_r_at         13
## 100 31670_s_at          4
## 101    31684_at          4
## 102    31686_at          4
## 103    31706_at          4
## 104    31744_at          4
## 105    31753_at         13
## 106    31790_at         13
```

```
## 107    31792_at         4
## 108    31805_at         4
## 109 31811_r_at         4
## 110    31847_at        13
## 111    31849_at        13
## 112    31851_at        13
## 113 31876_r_at         4
## 114    31894_at         4
## 115 31969_i_at         4
## 116 31970_r_at         4
## 117 32006_r_at         4
## 118 32026_s_at         4
## 119    32080_at         4
## 120    32102_at        13
## 121    32145_at         4
## 122 32146_s_at         4
## 123    32147_at        13
## 124    32148_at        13
## 125 32163_f_at         4
## 126 32180_s_at         4
## 127    32220_at        13
## 128    32299_at         4
## 129    32349_at         4
## 130    32353_at         4
## 131    32357_at         4
## 132    32368_at        13
## 133 32393_s_at         4
## 134    32439_at        13
## 135    32446_at         4
## 136    32449_at         4
## 137    32465_at         4
## 138    32482_at        13
## 139    32506_at         4
## 140    32507_at         4
## 141    32570_at         4
## 142    32580_at         4
## 143    32595_at         4
## 144    32602_at         4
## 145    32641_at        13
## 146    32675_at         4
## 147    32703_at         4
## 148    32768_at        13
## 149    32769_at         4
## 150    32770_at         4
## 151    32771_at         4
## 152    32812_at         4
## 153    32822_at         4
```

```
## 154    32832_at          4
## 155    32862_at         13
## 156    32906_at         13
## 157    32979_at          4
## 158 32986_s_at          13
## 159    32998_at          4
## 160    33013_at          4
## 161    33050_at          4
## 162 33068_f_at           4
## 163 33069_f_at           4
## 164    33100_at          4
## 165    33150_at          4
## 166 33151_s_at           4
## 167    33155_at          4
## 168    33156_at          4
## 169    33168_at         13
## 170 33171_s_at           4
## 171    33172_at          4
## 172 33173_g_at           4
## 173    33199_at         13
## 174    33208_at         13
## 175    33241_at          4
## 176    33249_at          4
## 177    33267_at          4
## 178    33276_at         13
## 179    33299_at          4
## 180    33318_at         13
## 181    33356_at          4
## 182    33359_at          4
## 183    33369_at          4
## 184 33370_r_at           4
## 185    33382_at          4
## 186    33483_at          4
## 187    33488_at          4
## 188    33490_at          4
## 189    33494_at          4
## 190    33519_at          4
## 191    33520_at         13
## 192    33525_at          4
## 193    33526_at          4
## 194    33529_at          4
## 195    33536_at          4
## 196    33544_at          4
## 197    33564_at          4
## 198    33576_at         13
## 199    33584_at          4
## 200    33596_at          4
```

```
## 201    33657_at            4
## 202 33672_f_at            4
## 203 33673_r_at            4
## 204    33687_at           13
## 205    33700_at           13
## 206    33733_at            4
## 207    33791_at           13
## 208    33823_at            4
## 209    33827_at           13
## 210    33837_at            4
## 211    33859_at           13
## 212    33975_at            4
## 213    33990_at            4
## 214 33991_g_at            4
## 215    33992_at            4
## 216    33997_at            4
## 217    34021_at            4
## 218    34022_at            4
## 219    34026_at           13
## 220    34029_at            4
## 221    34048_at            4
## 222    34051_at           13
## 223    34058_at            4
## 224    34075_at            4
## 225    34122_at            4
## 226    34131_at            4
## 227    34144_at            4
## 228    34145_at            4
## 229    34149_at            4
## 230 34170_s_at            4
## 231    34181_at            4
## 232    34198_at            4
## 233    34211_at           13
## 234    34239_at           13
## 235 34240_s_at           13
## 236    34247_at            4
## 237    34248_at            4
## 238 34275_s_at            4
## 239    34284_at           13
## 240    34307_at           13
## 241    34319_at            4
## 242    34324_at           13
## 243    34334_at           13
## 244    34335_at           13
## 245    34341_at            4
## 246 34342_s_at            4
## 247    34353_at            4
```

```
## 248    34398_at          13
## 249    34411_at           4
## 250    34423_at           4
## 251    34459_at          13
## 252 34476_r_at           4
## 253    34482_at           4
## 254    34512_at           4
## 255    34551_at           4
## 256    34564_at           4
## 257    34565_at           4
## 258    34578_at          13
## 259    34583_at          13
## 260    34596_at           4
## 261 34637_f_at           4
## 262 34638_r_at           4
## 263    34657_at          13
## 264    34672_at          13
## 265    34745_at           4
## 266    34803_at          13
## 267    34898_at           4
## 268 34953_i_at           4
## 269 34954_r_at           4
## 270    34955_at          13
## 271    34973_at           4
## 272    34984_at           4
## 273    34988_at           4
## 274    35020_at           4
## 275    35021_at           4
## 276    35025_at           4
## 277    35028_at           4
## 278    35039_at           4
## 279    35053_at           4
## 280    35061_at           4
## 281    35063_at           4
## 282    35081_at          13
## 283    35105_at          13
## 284    35107_at          13
## 285    35110_at          13
## 286    35131_at           4
## 287    35134_at           4
## 288    35140_at          13
## 289    35147_at          13
## 290    35164_at           4
## 291    35181_at           4
## 292 35182_f_at           4
## 293    35193_at          13
## 294    35213_at          13
```

```
## 295    35214_at             4
## 296    35215_at             4
## 297    35220_at             4
## 298    35285_at             4
## 299    35306_at             4
## 300    35344_at            13
## 301    35356_at             4
## 302    35357_at             4
## 303    35371_at             4
## 304 35372_r_at             4
## 305    35400_at            13
## 306    35410_at             4
## 307 35435_s_at             4
## 308    35437_at             4
## 309    35469_at            13
## 310    35470_at            13
## 311 35471_g_at            13
## 312    35481_at            13
## 313    35507_at             4
## 314    35523_at             4
## 315 35554_f_at            13
## 316 35555_r_at            13
## 317    35591_at             4
## 318    35656_at            13
## 319    35662_at             4
## 320    35664_at             4
## 321    35678_at             4
## 322    35689_at             4
## 323    35698_at             4
## 324    35725_at            13
## 325    35730_at             4
## 326    35777_at             4
## 327    35793_at             4
## 328    35827_at             4
## 329    35837_at             4
## 330    35845_at             4
## 331 35871_s_at             4
## 332    35877_at            13
## 333    35904_at            13
## 334 35939_s_at            13
## 335    35940_at            13
## 336    35949_at            13
## 337    35972_at            13
## 338    35989_at             4
## 339    35991_at             4
## 340    36012_at            13
## 341    36013_at             4
```

```
## 342    36017_at         13
## 343    36021_at          4
## 344    36031_at         13
## 345    36046_at          4
## 346    36047_at          4
## 347    36065_at          4
## 348    36080_at          4
## 349    36143_at          4
## 350    36157_at          4
## 351    36188_at         13
## 352    36194_at          4
## 353    36212_at         13
## 354    36243_at          4
## 355 36247_f_at          4
## 356    36269_at          4
## 357    36274_at         13
## 358    36358_at          4
## 359    36363_at          4
## 360    36433_at          4
## 361 36434_r_at          4
## 362    36510_at         13
## 363    36521_at         13
## 364    36606_at          4
## 365    36622_at          4
## 366    36627_at          4
## 367    36659_at         13
## 368    36717_at          4
## 369    36788_at         13
## 370      367_at         13
## 371    36814_at          4
## 372    36830_at         13
## 373    36913_at          4
## 374    36914_at          4
## 375    36915_at          4
## 376    36918_at          4
## 377    36939_at          4
## 378 36968_s_at         13
## 379    36990_at          4
## 380    37006_at          4
## 381    37019_at          4
## 382    37023_at         13
## 383    37056_at          4
## 384    37058_at          4
## 385    37062_at          4
## 386    37067_at         13
## 387    37079_at         13
## 388    37099_at         13
```

```
## 389    37109_at          13
## 390    37154_at          13
## 391    37170_at           4
## 392    37172_at          13
## 393    37173_at           4
## 394    37187_at           4
## 395    37206_at           4
## 396    37219_at           4
## 397    37223_at           4
## 398    37243_at           4
## 399    37244_at          13
## 400    37280_at           4
## 401    37282_at           4
## 402 37291_r_at           4
## 403    37303_at          13
## 404 37322_s_at           4
## 405 37323_r_at           4
## 406 37356_r_at           4
## 407    37366_at           4
## 408    37404_at           4
## 409    37416_at           4
## 410    37472_at           4
## 411    37518_at          13
## 412    37520_at           4
## 413 37521_s_at           4
## 414 37522_r_at           4
## 415    37571_at          13
## 416    37578_at           4
## 417    37593_at          13
## 418    37619_at           4
## 419    37658_at          13
## 420 37707_i_at           4
## 421 37708_r_at           4
## 422    37723_at           4
## 423    37747_at           4
## 424    37748_at           4
## 425    37752_at           4
## 426    37757_at          13
## 427    37767_at           4
## 428    37840_at           4
## 429    37852_at           4
## 430    37926_at          13
## 431    37930_at          13
## 432    37964_at           4
## 433    38008_at           4
## 434    38016_at           4
## 435    38024_at           4
```

```
## 436 38025_r_at          4
## 437    38035_at         13
## 438    38065_at          4
## 439    38102_at         13
## 440    38120_at          4
## 441    38168_at          4
## 442    38254_at          4
## 443 38304_r_at          13
## 444    38353_at         13
## 445    38375_at         13
## 446    38438_at          4
## 447    38485_at          4
## 448 38488_s_at           4
## 449    38489_at          4
## 450    38587_at          4
## 451    38606_at          4
## 452    38615_at         13
## 453    38643_at          4
## 454    38649_at         13
## 455    38714_at          4
## 456    38715_at          4
## 457    38736_at          4
## 458 38751_i_at           4
## 459 38752_r_at           4
## 460    38767_at          4
## 461    38768_at          4
## 462    38778_at          4
## 463    38821_at          4
## 464    38825_at          4
## 465    38838_at          4
## 466    38854_at          4
## 467    38891_at          4
## 468    38957_at         13
## 469    38972_at         13
## 470    38988_at          4
## 471    39028_at         13
## 472    39032_at         13
## 473    39037_at          4
## 474    39056_at          4
## 475    39083_at          4
## 476    39131_at         13
## 477    39132_at          4
## 478 39208_i_at           4
## 479 39209_r_at           4
## 480    39224_at          4
## 481    39256_at         13
## 482    39257_at         13
```

```
## 483   39269_at        13
## 484 39295_s_at         4
## 485   39297_at        13
## 486   39333_at        13
## 487   39337_at         4
## 488   39355_at         4
## 489   39369_at         4
## 490   39380_at         4
## 491   39382_at         4
## 492   39405_at        13
## 493 39469_s_at        13
## 494   39475_at         4
## 495   39481_at         4
## 496   39488_at        13
## 497 39489_g_at        13
## 498   39535_at         4
## 499   39536_at         4
## 500   39554_at         4
## 501   39555_at         4
## 502   39576_at         4
## 503   39579_at        13
## 504   39600_at         4
## 505   39634_at         4
## 506 39662_s_at         4
## 507   39665_at         4
## 508   39680_at         4
## 509   39690_at         4
## 510   39698_at         4
## 511   39734_at         4
## 512   39746_at         4
## 513   39748_at        13
## 514 39758_f_at        13
## 515   39777_at        13
## 516   39786_at         4
## 517   39847_at         4
## 518   39850_at         4
## 519   39851_at         4
## 520   39852_at        13
## 521   39878_at        13
## 522   39897_at         4
## 523   39924_at        13
## 524   39929_at         4
## 525   39955_at        13
## 526   39960_at         4
## 527   39979_at        13
## 528   40018_at        13
## 529 40058_s_at         4
```

```
## 530 40059_r_at          4
## 531 40060_r_at          4
## 532    40067_at        13
## 533    40072_at        13
## 534    40082_at         4
## 535      400_at        13
## 536    40114_at         4
## 537    40121_at         4
## 538    40148_at         4
## 539    40180_at        13
## 540 40181_f_at         13
## 541    40199_at         4
## 542 40217_s_at          4
## 543    40218_at         4
## 544    40225_at         4
## 545    40226_at         4
## 546    40272_at         4
## 547    40310_at         4
## 548    40312_at        13
## 549    40323_at         4
## 550    40349_at         4
## 551    40354_at         4
## 552    40392_at        13
## 553 40404_s_at         13
## 554    40449_at         4
## 555    40454_at         4
## 556    40456_at         4
## 557    40473_at        13
## 558    40492_at         4
## 559    40530_at         4
## 560    40570_at        13
## 561 40576_f_at          4
## 562    40633_at        13
## 563    40681_at        13
## 564    40697_at         4
## 565    40710_at         4
## 566    40711_at         4
## 567    40727_at         4
## 568    40746_at         4
## 569 40770_f_at          4
## 570    40772_at         4
## 571    40773_at         4
## 572    40818_at         4
## 573    40828_at        13
## 574    40839_at        13
## 575    40853_at         4
## 576 40880_r_at          4
```

```
## 577    40893_at          13
## 578     408_at           4
## 579 40908_r_at          13
## 580    40943_at          4
## 581    40970_at         13
## 582    40989_at          4
## 583    40990_at          4
## 584    40991_at          4
## 585 40992_s_at          4
## 586 40993_r_at          4
## 587 41014_s_at          4
## 588 41024_f_at          4
## 589 41025_r_at          4
## 590 41026_f_at          4
## 591    41069_at         13
## 592    41071_at          4
## 593    41104_at          4
## 594    41118_at         13
## 595 41119_f_at          13
## 596    41145_at          4
## 597    41148_at          4
## 598    41182_at         13
## 599    41191_at          4
## 600    41276_at         13
## 601    41277_at         13
## 602 41300_s_at          13
## 603    41301_at         13
## 604    41308_at          4
## 605 41309_g_at          4
## 606    41317_at         13
## 607 41318_g_at          13
## 608    41319_at         13
## 609 41376_i_at          4
## 610 41377_f_at          4
## 611    41391_at          4
## 612    41392_at          4
## 613    41402_at          4
## 614    41434_at          4
## 615    41436_at         13
## 616    41456_at          4
## 617    41459_at         13
## 618    41470_at          4
## 619 41491_s_at          13
## 620 41492_r_at          13
## 621    41493_at         13
## 622    41534_at          4
## 623    41555_at          4
```

```
## 624 41556_s_at         4
## 625   41585_at         4
## 626 41667_s_at        13
## 627 41668_r_at        13
## 628   41697_at         4
## 629   41801_at         4
## 630   41806_at         4
## 631   41860_at        13
## 632     431_at         4
## 633     504_at         4
## 634   507_s_at         4
## 635     579_at         4
## 636     618_at         4
## 637     630_at         4
## 638   631_g_at         4
## 639     655_at         4
## 640   690_s_at         4
## 641   692_s_at         4
## 642   764_s_at         4
## 643     820_at         4
## 644     886_at         4
## 645     931_at        13
## 646   936_s_at         4
## 647   948_s_at         4
## 648     963_at        13
## 649     975_at         4
## 650     990_at        13
## 651   991_g_at        13
```

To get this in the classic named-list format:

```
z <- as.list(revmap(x)[chroms])
names(z)
```

```
## [1] "4"  "13"
```

```
z[["Y"]]
```

```
## NULL
```

Many of the common methods for accessing *Bimap* objects return things in list format. This can be convenient. But you have to be careful about this if you want to use unlist(). For example the following will return multiple probes for each chromosome:

```
chrs = c("12","6")
mget(chrs, revmap(hgu95av2CHR[1:30]), ifnotfound=NA)
```

```
## $`12`
## [1] "1018_at"   "1019_g_at" "101_at"    "1021_at"
##
## $`6`
```

```
## [1] "1026_s_at" "1027_at"
```

But look what happens here if we try to unlist that:

```
unlist(mget(chrs, revmap(hgu95av2CHR[1:30]), ifnotfound=NA))
```

```
##          121          122          123          124           61           62
##     "1018_at" "1019_g_at"     "101_at"    "1021_at" "1026_s_at"    "1027_at"
```

Yuck! One trick that will sometimes help is to use Rfunctionunlist2. But be careful here too. Depending on what step comes next, Rfunctionunlist2 may not really help you...

```
unlist2(mget(chrs, revmap(hgu95av2CHR[1:30]), ifnotfound=NA))
```

```
##           12           12           12           12            6            6
##     "1018_at" "1019_g_at"     "101_at"    "1021_at" "1026_s_at"    "1027_at"
```

Lets ask if the probes in 'pbids' mapped to cytogenetic location "18q11.2"?

```
x <- hgu95av2MAP
pbids <- c("38912_at", "41654_at", "907_at", "2053_at", "2054_g_at",
           "40781_at")
x <- subset(x, Lkeys=pbids, Rkeys="18q11.2")
toTable(x)
```

```
##     probe_id cytogenetic_location
## 1    2053_at              18q11.2
## 2 2054_g_at              18q11.2
```

To coerce this map to a named vector:

```
  pb2cyto <- as.character(x)
  pb2cyto[pbids]
```

```
##        <NA>        <NA>        <NA>     2053_at 2054_g_at        <NA>
##          NA          NA          NA "18q11.2" "18q11.2"          NA
```

The coercion of the reverse map works too but issues a warning because of the duplicated names for the reasons stated above:

```
  cyto2pb <- as.character(revmap(x))
```

```
## Warning in .local(x, ...):  returned vector has duplicated names
```

2.0.8 Accessing probes that map to multiple targets

In many probe packages, some probes are known to map to multiple genes. The reasons for this can be biological as happens in the arabidopsis packages, but usually it is due to the fact that the genome builds that chip platforms were based on were less stable than desired. Thus what may have originally been a probe designed to measure one thing can end up measuring many things. Usually you don't want to use probes like this, because if they manufacturer doesn't know what they map to then their usefullness is definitely suspect. For this reason, by default all chip packages will normally hide such probes in the standard mappings. But sometimes you may want access to the answers that the manufacturer says such a probe will map to. In such

cases, you will want to use the `toggleProbes` method. To use this method, just call it on a standard mapping and copy the result into a new mapping (you cannot alter the original mapping). Then treat the new mapping as you would any other mapping.

```
## How many probes?
dim(hgu95av2ENTREZID)
```

```
## [1] 11543     2
```

```
## Make a mapping with multiple probes exposed
multi <- toggleProbes(hgu95av2ENTREZID, "all")
## How many probes?
dim(multi)
```

```
## [1] 13372     2
```

If you then decide that you want to make a mapping that has only multiple mappings or you wish to revert one of your maps back to the default state of only showing the single mappings then you can use `toggleProbes` to switch back and forth.

```
## Make a mapping with ONLY multiple probes exposed
multiOnly <- toggleProbes(multi, "multiple")
## How many probes?
dim(multiOnly)
```

```
## [1] 1829     2
```

```
## Then make a mapping with ONLY single mapping probes
singleOnly <- toggleProbes(multiOnly, "single")
## How many probes?
dim(singleOnly)
```

```
## [1] 11543     2
```

Finally, there are also a pair of test methods `hasMultiProbes` and `hasSingleProbes` that can be used to see what methods a mapping presently has exposed.

```
## Test the multiOnly mapping
hasMultiProbes(multiOnly)
```

```
## [1] TRUE
```

```
hasSingleProbes(multiOnly)
```

```
## [1] FALSE
```

```
## Test the singleOnly mapping
hasMultiProbes(singleOnly)
```

```
## [1] FALSE
```

```
hasSingleProbes(singleOnly)
```

```
## [1] TRUE
```

2.0.9   Using SQL to access things directly

While the mapping objects provide a lot of convenience, sometimes there are definite benefits to writing a simple SQL query. But in order to do this, it is necessary to know a few things. The 1st thing you will need to know is some SQL. Fortunately, it is quite easy to learn enough basic SQL to get stuff out of a database. Here are 4 basic SQL things that you may find handy:

First, you need to know about SELECT statements. A simple example would look something like this:

SELECT * FROM genes;

Which would select everything from the genes table.

SELECT gene_id FROM genes;

Will select only the gene_id field from the genes table.

Second you need to know about WHERE clauses:

SELECT gene_id,_id FROM genes WHERE gene_id=1;

Will only get records from the genes table where the gene_id is = 1.

Thirdly, you will want to know about an inner join:

SELECT * FROM genes,chromosomes WHERE genes._id=chromosomes._id;

This is only slightly more complicated to understand. Here we want to get all the records that are in both the 'genes' and 'chromosomes' tables, but we only want ones where the '_id' field is identical. This is known as an inner join because we only want the elements that are in both of these tables with respect to '_id'. There are other kinds of joins that are worth learning about, but most of the time, this is all you will need to do.

Finally, it is worthwhile to learn about the AS keyword which is useful for making long queries easier to read. For the previous example, we could have written it this way to save space:

SELECT * FROM genes AS g,chromosomes AS c WHERE g._id=c._id;

In a simple example like this you might not see a lot of savings from using AS, so lets consider what happens when we want to also specify which fields we want:

SELECT g.gene_id,c.chromosome FROM genes AS g,chromosomes AS c WHERE g._id=c._id;

Now you are most of the way there to being able to query the databases directly. The only other thing you need to know is a little bit about how to access these databases from R. With each package, you will also get a method that will print the schema for its database, you can view this to see what sorts of tables are present etc.

```
org.Hs.eg_dbschema()
```

To access the data in a database, you will need to connect to it. Fortunately, each package will automatically give you a connection object to that database when it loads.

```
org.Hs.eg_dbconn()
```

You can use this connection object like this:

```
query <- "SELECT gene_id FROM genes LIMIT 10;"
result = dbGetQuery(org.Hs.eg_dbconn(), query)
result
```

Exercise 5
Retrieve the entrez gene ID and chromosome by using a database query. Show how you could do the same thing by using `toTable`

2.0.10   Combining data from multiple annotation packages at the SQL level

For a more complex example, consider the task of obtaining all gene symbols which are probed on a chip that have at least one GO BP ID annotation with evidence code IMP, IGI, IPI, or IDA. Here is one way to extract this using the environment-based packages:

```
## Obtain SYMBOLS with at least one GO BP
## annotation with evidence IMP, IGI, IPI, or IDA.
system.time({
bpids <- eapply(hgu95av2GO, function(x) {
    if (length(x) == 1 && is.na(x))
      NA
    else {
        sapply(x, function(z) {
            if (z$Ontology == "BP")
              z$GOID
            else
              NA
        })
    }
})
bpids <- unlist(bpids)
bpids <- unique(bpids[!is.na(bpids)])
g2p <- mget(bpids, hgu95av2GO2PROBE)
wantedp <- lapply(g2p, function(x) {
    x[names(x) %in% c("IMP", "IGI", "IPI", "IDA")]
})
wantedp <- wantedp[sapply(wantedp, length) > 0]
wantedp <- unique(unlist(wantedp))
ans <- unlist(mget(wantedp, hgu95av2SYMBOL))
})
length(ans)
ans[1:10]
```

All of the above code could have been reduced to a single SQL query with the SQLite-based packages. But to put together this query, you would need to look 1st at the schema to know what tables are present:

```
hgu95av2_dbschema()
```

This function will give you an output of all the create table statements that were used to generate the hgu95av2 database. In this case, this is a chip package, so you will also need to see the schema for the organism package

that it depends on. To learn what package it depends on, look at the ORGPKG value:

```
hgu95av2ORGPKG
```

Then you can see that schema by looking at its schema method:

```
org.Hs.eg_dbschema()
```

So now we can see that we want to connect the data in the go_bp, and symbol tables from the org.Hs.eg.sqlite database along with the probes data in the hgu95av2.sqlite database. How can we do that?

It turns out that one of the great conveniences of SQLite is that it allows other databases to be 'ATTACHed'. Thus, we can keep our data in many differnt databases, and then 'ATTACH' them to each other in a modular fashion. The databases for a given build have been built together and frozen into a single version specifically to allow this sort of behavoir. To use this feature, the SQLite ATTACH command requires the filename for the database file on your filesystem. Fortunately, R provides a nice system independent way of getting that information. Note that the name of the database is always the same as the name of the package, with the suffix '.sqlite'.:

```
orgDBLoc = system.file("extdata", "org.Hs.eg.sqlite", package="org.Hs.eg.db")
attachSQL = paste("ATTACH '", orgDBLoc, "' AS orgDB;", sep = "")
dbGetQuery(hgu95av2_dbconn(), attachSQL)
```

Finally, you can assemble a cross-db sql query and use the helper function as follows. Note that when we want to refer to tables in the attached database, we have to use the 'orgDB' prefix that we specified in the 'ATTACH' query above.:

```
system.time({
SQL <- "SELECT DISTINCT probe_id,symbol FROM probes, orgDB.gene_info AS gi, orgDB.genes AS g, org
zz <- dbGetQuery(hgu95av2_dbconn(), SQL)
})

##    user  system elapsed
##   0.220   0.000   0.245

#its a good idea to always DETACH your database when you are finished...
dbGetQuery(hgu95av2_dbconn(), "DETACH orgDB"          )
```

Exercise 6
Retrieve the entrez gene ID, chromosome location information and cytoband infomration by using a single database query.

Exercise 7
Expand on the example in the text above to combine data from the hgu95av2.db and org.Hs.eg.db with the GO.db package so as to include the GO ID, and term definition in the output.

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

```
## R version 3.1.3 (2015-03-09)
## Platform: x86_64-unknown-linux-gnu (64-bit)
## Running under: Ubuntu precise (12.04.5 LTS)
##
## locale:
```

```
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils
## [7] datasets  methods   base
##
## other attached packages:
##  [1] GO.db_3.0.0          hgu95av2.db_3.0.0     AnnotationForge_1.8.2
##  [4] org.Hs.eg.db_3.0.0   RSQLite_1.0.0         DBI_0.3.1
##  [7] AnnotationDbi_1.28.2 GenomeInfoDb_1.2.4    IRanges_2.0.1
## [10] S4Vectors_0.4.0      Biobase_2.26.0        BiocGenerics_0.12.1
## [13] knitr_1.9
##
## loaded via a namespace (and not attached):
## [1] BiocStyle_1.4.1 evaluate_0.5.5  formatR_1.0     highr_0.4
## [5] stringr_0.6.2   tools_3.1.3
```