How to access Annotations in Bioconductor

Marc Carlson

April 27, 2009

1 Introduction

There are two basic ways to get annotations in bioconductor. You can either use an annotation package, or you can use biomaRt. In either case, it is critical to understand where the data comes from and how it relates (or doesn't) to other annotation data. The annotation packages are always gene-centric, and usually this means that they entrez-gene centric. The packaged data is usually based primarily on data from NCBI. This means that the heart of the data is usually an entrez gene ID, and that all the other data in a package will be tied to that central ID. Furthermore, the packaged annotations are frozen at release time, so that they are all updated twice a year.

In contrast, data accessed through biomaRt usually originates from ensembl. So users who try to combine data from the different sources should not be shocked to discover that data from ensembl and NCBI are not always in perfect agreement. Also, biomaRt uses a web-based interface to a set of sources that are always changing. This is an advantage when you want to have the very most current data available, but it can be a disadvantage when you are trying to troubleshoot a process or carefully control variables to make sense of your analysis.

2 Using an annotation package

The organism packages provide a range of different gene-centric annotations about an organism. For most organisms, the packages are entrez gene centric. One such package is the org.Hs.eg.db package.

The org packages provide basic annotation for an entire organism. The data is always gene centric and there is always an identifier that is used as the central ID for the packages database. For most species that we support this is an entrez gene ID. But there are some exceptions. One notable exception is yeast where we use the systematic names as the central ID instead of entrez gene IDs. The structure of the internal database is such that all the other data in an organism package is connected to this central ID.

You can see a few things about an organism package just by looking at its name. The second part of the name is the Genus and species of the package as a two letter abbreviation, and the 3rd part of the name indicates the origin of the data. So while the human organism package listed above is based on entrez gene IDs, the yeast package is named org.Sc.sgd.db because its data comes primarily from SGD.

The data in an organism package is organized into a series of mappings that will connect the central gene IDs to various other kinds of information.

```
> ##load the package
> library("org.Hs.eg.db")
> ##look what we just loaded
> ls(2)
```

```
 [1] "org.Hs.eg"                "org.Hs.eg_dbconn"
 [3] "org.Hs.eg_dbfile"         "org.Hs.eg_dbInfo"
 [5] "org.Hs.eg_dbschema"       "org.Hs.egACCNUM"
 [7] "org.Hs.egACCNUM2EG"       "org.Hs.egALIAS2EG"
 [9] "org.Hs.egCHR"             "org.Hs.egCHRLENGTHS"
[11] "org.Hs.egCHRLOC"          "org.Hs.egCHRLOCEND"
[13] "org.Hs.egENSEMBL"         "org.Hs.egENSEMBL2EG"
[15] "org.Hs.egENSEMBLPROT"     "org.Hs.egENSEMBLPROT2EG"
[17] "org.Hs.egENSEMBLTRANS"    "org.Hs.egENSEMBLTRANS2EG"
[19] "org.Hs.egENZYME"          "org.Hs.egENZYME2EG"
[21] "org.Hs.egGENENAME"        "org.Hs.egGO"
[23] "org.Hs.egGO2ALLEGS"       "org.Hs.egGO2EG"
[25] "org.Hs.egMAP"             "org.Hs.egMAP2EG"
[27] "org.Hs.egMAPCOUNTS"       "org.Hs.egOMIM"
[29] "org.Hs.egOMIM2EG"         "org.Hs.egORGANISM"
[31] "org.Hs.egPATH"            "org.Hs.egPATH2EG"
[33] "org.Hs.egPFAM"            "org.Hs.egPMID"
[35] "org.Hs.egPMID2EG"         "org.Hs.egPROSITE"
[37] "org.Hs.egREFSEQ"          "org.Hs.egREFSEQ2EG"
[39] "org.Hs.egSYMBOL"          "org.Hs.egSYMBOL2EG"
[41] "org.Hs.egUNIGENE"         "org.Hs.egUNIGENE2EG"
[43] "org.Hs.egUNIPROT"
```

```
> ##Just like user accessible functions, all Mappings have a manual page which
> ##will show you what to expect as well as where the data came from
> # ?org.Hs.egCHRLOC
>
> ##Have a peak:
> as.list(org.Hs.egCHRLOC[1:4])

$`1`
       19
-63549983

$`10`
       8
18293034

$`100`
       20
-42681576

$`1000`
       18
-23784927

> ##for the stop locations use:
> as.list(org.Hs.egCHRLOCEND[1:4])

$`1`
       19
-63556677

$`10`
       8
18303003

$`100`
       20
-42713790

$`1000`
       18
-24011443
```

```
> ##or can use get, mget etc. with the entrez gene ID
> EGs = c("10","100","1000")
> mget(EGs, org.Hs.egCHRLOC, ifnotfound=NA)

$`10`
       8
18293034

$`100`
       20
-42681576

$`1000`
       18
-23784927

> mget(EGs, org.Hs.egCHRLOCEND, ifnotfound=NA)

$`10`
       8
18303003

$`100`
       20
-42713790

$`1000`
       18
-24011443

> ##You can also retrieve ENSEMBL IDs using this package
> mget(EGs, org.Hs.egENSEMBL, ifnotfound=NA)

$`10`
[1] "ENSG00000156006"

$`100`
[1] "ENSG00000196839"

$`1000`
[1] "ENSG00000170558"
```

```
> ##And GO IDs
> mget(EGs[1], org.Hs.egGO, ifnotfound=NA)

$`10`
$`10`$`GO:0008152`
$`10`$`GO:0008152`$GOID
[1] "GO:0008152"

$`10`$`GO:0008152`$Evidence
[1] "IEA"

$`10`$`GO:0008152`$Ontology
[1] "BP"


$`10`$`GO:0005829`
$`10`$`GO:0005829`$GOID
[1] "GO:0005829"

$`10`$`GO:0005829`$Evidence
[1] "EXP"

$`10`$`GO:0005829`$Ontology
[1] "CC"


$`10`$`GO:0005737`
$`10`$`GO:0005737`$GOID
[1] "GO:0005737"

$`10`$`GO:0005737`$Evidence
[1] "IEA"

$`10`$`GO:0005737`$Ontology
[1] "CC"


$`10`$`GO:0004060`
$`10`$`GO:0004060`$GOID
[1] "GO:0004060"
```

```
$`10`$`GO:0004060`$Evidence
[1] "TAS"

$`10`$`GO:0004060`$Ontology
[1] "MF"


$`10`$`GO:0016407`
$`10`$`GO:0016407`$GOID
[1] "GO:0016407"

$`10`$`GO:0016407`$Evidence
[1] "IEA"

$`10`$`GO:0016407`$Ontology
[1] "MF"


$`10`$`GO:0016740`
$`10`$`GO:0016740`$GOID
[1] "GO:0016740"

$`10`$`GO:0016740`$Evidence
[1] "IEA"

$`10`$`GO:0016740`$Ontology
[1] "MF"

> ##And KEGG pathway IDs etc.
> mget(EGs, org.Hs.egPATH, ifnotfound=NA)

$`10`
[1] "00232" "00983"

$`100`
[1] "00230" "05340"

$`1000`
[1] "04514"
```

```
> ##Other convenient functions
> ##Lkeys, RKeys, mappedLkeys(),mappedRkeys()
> Lkeys(org.Hs.egENZYME)[110:112]

[1] "100033807" "100033808" "100033809"

> Rkeys(org.Hs.egENZYME)[110:112]

[1] "1.14.99.9" "1.15.1.1"  "1.16.1.-"

> ##Left keys and right keys can be mapped or un-mapped.
> length(Lkeys(org.Hs.egPATH))

[1] 40784

> length(Rkeys(org.Hs.egPATH))

[1] 205

> length(mappedLkeys(org.Hs.egPATH))

[1] 4799

> length(mappedRkeys(org.Hs.egPATH))

[1] 205

> ##keys() and mappedkeys() both return the left keys
> length(keys(org.Hs.egPATH))

[1] 40784

> length(mappedkeys(org.Hs.egPATH))

[1] 4799

> ##revmap() can USUALLY be used to reverse the direction of a mapping.
> PATHIDs = unlist(mget(EGs[1], org.Hs.egPATH, ifnotfound=NA))[[1]]
> PATHIDs

[1] "00232"

> mget(as.character(PATHIDs), revmap(org.Hs.egPATH), ifnotfound=NA)
```

```
$`00232`
[1] "9"    "10"   "1544" "1548" "1549" "1553" "7498"

> ##toTable
> toTable(revmap(org.Hs.egPATH))[1:4,]

  gene_id path_id
1       2   04610
2       9   00232
3       9   00983
4      10   00232

> ##special symbols: packagename(), _dbfile(), _dbinfo(), and _dbconn()
> ls(2)

 [1] "org.Hs.eg"                  "org.Hs.eg_dbconn"
 [3] "org.Hs.eg_dbfile"           "org.Hs.eg_dbInfo"
 [5] "org.Hs.eg_dbschema"         "org.Hs.egACCNUM"
 [7] "org.Hs.egACCNUM2EG"         "org.Hs.egALIAS2EG"
 [9] "org.Hs.egCHR"               "org.Hs.egCHRLENGTHS"
[11] "org.Hs.egCHRLOC"            "org.Hs.egCHRLOCEND"
[13] "org.Hs.egENSEMBL"           "org.Hs.egENSEMBL2EG"
[15] "org.Hs.egENSEMBLPROT"       "org.Hs.egENSEMBLPROT2EG"
[17] "org.Hs.egENSEMBLTRANS"      "org.Hs.egENSEMBLTRANS2EG"
[19] "org.Hs.egENZYME"            "org.Hs.egENZYME2EG"
[21] "org.Hs.egGENENAME"          "org.Hs.egGO"
[23] "org.Hs.egGO2ALLEGS"         "org.Hs.egGO2EG"
[25] "org.Hs.egMAP"               "org.Hs.egMAP2EG"
[27] "org.Hs.egMAPCOUNTS"         "org.Hs.egOMIM"
[29] "org.Hs.egOMIM2EG"           "org.Hs.egORGANISM"
[31] "org.Hs.egPATH"              "org.Hs.egPATH2EG"
[33] "org.Hs.egPFAM"              "org.Hs.egPMID"
[35] "org.Hs.egPMID2EG"           "org.Hs.egPROSITE"
[37] "org.Hs.egREFSEQ"            "org.Hs.egREFSEQ2EG"
[39] "org.Hs.egSYMBOL"            "org.Hs.egSYMBOL2EG"
[41] "org.Hs.egUNIGENE"           "org.Hs.egUNIGENE2EG"
[43] "org.Hs.egUNIPROT"

> ##package info
> org.Hs.eg()
```

```
Quality control information for org.Hs.eg:


This package has the following mappings:

org.Hs.egACCNUM has 29687 mapped keys (of 40784 keys)
org.Hs.egACCNUM2EG has 590454 mapped keys (of 590454 keys)
org.Hs.egALIAS2EG has 102986 mapped keys (of 102986 keys)
org.Hs.egCHR has 40539 mapped keys (of 40784 keys)
org.Hs.egCHRLENGTHS has 25 mapped keys (of 25 keys)
org.Hs.egCHRLOC has 20599 mapped keys (of 40784 keys)
org.Hs.egCHRLOCEND has 20599 mapped keys (of 40784 keys)
org.Hs.egENSEMBL has 20255 mapped keys (of 40784 keys)
org.Hs.egENSEMBL2EG has 19903 mapped keys (of 19903 keys)
org.Hs.egENSEMBLPROT has 19927 mapped keys (of 40784 keys)
org.Hs.egENSEMBLPROT2EG has 44871 mapped keys (of 44871 keys)
org.Hs.egENSEMBLTRANS has 19965 mapped keys (of 40784 keys)
org.Hs.egENSEMBLTRANS2EG has 44931 mapped keys (of 44931 keys)
org.Hs.egENZYME has 2015 mapped keys (of 40784 keys)
org.Hs.egENZYME2EG has 870 mapped keys (of 870 keys)
org.Hs.egGENENAME has 40784 mapped keys (of 40784 keys)
org.Hs.egGO has 17482 mapped keys (of 40784 keys)
org.Hs.egGO2ALLEGS has 10438 mapped keys (of 10438 keys)
org.Hs.egGO2EG has 7659 mapped keys (of 7659 keys)
org.Hs.egMAP has 36549 mapped keys (of 40784 keys)
org.Hs.egMAP2EG has 2946 mapped keys (of 2946 keys)
org.Hs.egOMIM has 14080 mapped keys (of 40784 keys)
org.Hs.egOMIM2EG has 16415 mapped keys (of 16415 keys)
org.Hs.egPATH has 4799 mapped keys (of 40784 keys)
org.Hs.egPATH2EG has 205 mapped keys (of 205 keys)
org.Hs.egPFAM has 24009 mapped keys (of 40784 keys)
org.Hs.egPMID has 28206 mapped keys (of 40784 keys)
org.Hs.egPMID2EG has 232955 mapped keys (of 232955 keys)
org.Hs.egPROSITE has 24009 mapped keys (of 40784 keys)
org.Hs.egREFSEQ has 28158 mapped keys (of 40784 keys)
org.Hs.egREFSEQ2EG has 90796 mapped keys (of 90796 keys)
org.Hs.egSYMBOL has 40784 mapped keys (of 40784 keys)
org.Hs.egSYMBOL2EG has 40763 mapped keys (of 40763 keys)
org.Hs.egUNIGENE has 24864 mapped keys (of 40784 keys)
org.Hs.egUNIGENE2EG has 25562 mapped keys (of 25562 keys)
org.Hs.egUNIPROT has 20652 mapped keys (of 40784 keys)


Additional Information about this package:

DB schema: HUMAN_DB
DB schema version: 1.0
Organism: Homo sapiens
Date for NCBI data: 2009-Mar11
Date for GO data: 200903
Date for KEGG data: 2009-Mar10
Date for Golden Path data: 2008-Sep3
Date for IPI data: 2009-Mar03
Date for Ensembl data: 2009-Mar6
```

```
> ##location of the database file
> org.Hs.eg_dbfile()
```

```
[1] "/home/mcarlson/arch/x86_64/R-devel/library/org.Hs.eg.db/extdata/org.Hs.eg.sqlite
```

```
> ##Data frame with Information
> org.Hs.eg_dbInfo()
```

```
             name
1  DBSCHEMAVERSION
2         DBSCHEMA
3         ORGANISM
4          SPECIES
5     EGSOURCEDATE
6     EGSOURCENAME
7      EGSOURCEURL
8     GOSOURCENAME
9      GOSOURCEURL
10    GOSOURCEDATE
11  GOEGSOURCEDATE
12  GOEGSOURCENAME
13   GOEGSOURCEURL
14  KEGGSOURCENAME
15   KEGGSOURCEURL
16  KEGGSOURCEDATE
17    GPSOURCENAME
```

```
18     GPSOURCEURL
19    GPSOURCEDATE
20   IPISOURCENAME
21    IPISOURCEURL
22   IPISOURCEDATE
23    ENSOURCEDATE
24    ENSOURCENAME
25     ENSOURCEURL
                                                                  value
1                                                                   1.0
2                                                              HUMAN_DB
3                                                          Homo sapiens
4                                                                 Human
5                                                            2009-Mar11
6                                                           Entrez Gene
7                                   ftp://ftp.ncbi.nlm.nih.gov/gene/DATA
8                                                         Gene Ontology
9            ftp://ftp.geneontology.org/pub/go/godatabase/archive/latest
10                                                               200903
11                                                           2009-Mar11
12                                                          Entrez Gene
13                                 ftp://ftp.ncbi.nlm.nih.gov/gene/DATA
14                                                          KEGG GENOME
15                                 ftp://ftp.genome.jp/pub/kegg/genomes
16                                                           2009-Mar10
17                            UCSC Genome Bioinformatics (Homo sapiens)
18 ftp://hgdownload.cse.ucsc.edu/goldenPath/currentGenomes/Homo_sapiens
19                                                            2008-Sep3
20                                      The International Protein Index
21                       ftp://ftp.ebi.ac.uk/pub/databases/IPI/current
22                                                           2009-Mar03
23                                                            2009-Mar6
24                                                              Ensembl
25                              ftp://ftp.ensembl.org/pub/current_fasta
```

```
> ##Connection object
> org.Hs.eg_dbconn()
```

```
<SQLiteConnection:(19636,0)>
```

Behind the scenes, all of the annotation packages are really just small

SQLite databases wrapped up with a familiar interface. Because of this, it is possible to connect directly to these databases. SQLite also has the property that multiple databases can be attached which allows the tables of multiple databases to be joined together on the fly. We accomodate this feature by providing annotation packages that are a snapshot from a particular time period. Thus the data from different annotation packages can be combined on the fly in interesting ways as needed.

```
> ##simple examples of using the DBI interface
> library(org.Hs.eg.db)
> dbconn <- org.Hs.eg_dbconn()
> sql <- "SELECT * FROM genes LIMIT 4;"
> result <- dbGetQuery(dbconn, sql)
> result

  _id gene_id
1   1       1
2   2       2
3   3       3
4   4       9

> ##Here is a simple join of the type that generates the mappings
> sql <- "SELECT * FROM genes,kegg WHERE genes._id=kegg._id;"
> result <- dbGetQuery(dbconn, sql)
> result[1:4,]

  _id gene_id _id path_id
1   2       2   2   04610
2   4       9   4   00232
3   4       9   4   00983
4   5      10   5   00232

> ##simple example of a join across DBs
> ##get the entrez genes in humans and in mouse which share a kegg pathway ID.
>
> ## 1st we must get the pathway to the database file
> path = system.file("extdata", "org.Mm.eg.sqlite", package = "org.Mm.eg.db" )
> ##then we have to attach the database
> sql <- paste("ATTACH '",path,"' AS Mm;",sep="")
> dbSendQuery(dbconn, sql)
```

```
<SQLiteResult:(19636,0,63)>

> ##Then we can make our query
> sql <- "SELECT g.gene_id, k.path_id, mk.path_id, mg.gene_id
+         FROM genes AS g, kegg AS k, Mm.kegg AS mk, Mm.genes AS mg
+         WHERE g._id=k._id AND mg._id=mk._id AND k.path_id=mk.path_id
+         limit 1000;"
> result <- dbGetQuery(dbconn, sql)
> result[1:4,]

  gene_id path_id path_id gene_id
1       2   04610   04610   11537
2       2   04610   04610   11905
3       2   04610   04610   12061
4       2   04610   04610   12062

> ##can even use SQLite specific stuff
> sql = "SELECT * FROM sqlite_master;"
> result = dbGetQuery(dbconn, sql)
> head(result)

   type                          name     tbl_name rootpage
1 table                      metadata     metadata        2
2 index   sqlite_autoindex_metadata_1     metadata        3
3 table                  map_metadata map_metadata        4
4 table                    map_counts   map_counts        5
5 index sqlite_autoindex_map_counts_1   map_counts        6
6 table                         genes        genes        7

1
2
3 CREATE TABLE map_metadata (\n      map_name VARCHAR(80) NOT NULL,\n      source_nam
4
5
6                                                                  CREATE TABLE genes
```

3 Using a chip based annotation package

Sometimes you will want a more customized experience. That is what chip packages are for. Internally, a chip packages is just a subest of an org package with an extra table to capture the probe-gene relationships. Thus when you use the mappings in a chip package, AnnotationDbi will automatically map through the entrez gene ID to connect your probe to the data of interest for you.

```
> ##Things work very similarly to an org package
> library(hgu95av2.db)
> ls(2)

 [1] "hgu95av2"              "hgu95av2_dbconn"       "hgu95av2_dbfile"
 [4] "hgu95av2_dbInfo"       "hgu95av2_dbschema"     "hgu95av2ACCNUM"
 [7] "hgu95av2ALIAS2PROBE"   "hgu95av2CHR"           "hgu95av2CHRLENGTHS"
[10] "hgu95av2CHRLOC"        "hgu95av2CHRLOCEND"     "hgu95av2ENSEMBL"
[13] "hgu95av2ENSEMBL2PROBE" "hgu95av2ENTREZID"      "hgu95av2ENZYME"
[16] "hgu95av2ENZYME2PROBE"  "hgu95av2GENENAME"      "hgu95av2GO"
[19] "hgu95av2GO2ALLPROBES"  "hgu95av2GO2PROBE"      "hgu95av2MAP"
[22] "hgu95av2MAPCOUNTS"     "hgu95av2OMIM"          "hgu95av2ORGANISM"
[25] "hgu95av2PATH"          "hgu95av2PATH2PROBE"    "hgu95av2PFAM"
[28] "hgu95av2PMID"          "hgu95av2PMID2PROBE"    "hgu95av2PROSITE"
[31] "hgu95av2REFSEQ"        "hgu95av2SYMBOL"        "hgu95av2UNIGENE"
[34] "hgu95av2UNIPROT"

> ##Gene symbols and aliases
> probes = keys(hgu95av2SYMBOL)[1:4]
> probes

[1] "1000_at"   "1001_at"   "1002_f_at" "1003_s_at"

> ##You can use the probes in the same way you would have used Entrez Genes
> ##Here is a mapping that retrieves NCBIs official gene symbols
> mget(probes, hgu95av2SYMBOL, ifnotfound=NA)

$`1000_at`
[1] "MAPK3"

$`1001_at`
[1] "TIE1"

$`1002_f_at`
[1] "CYP2C19"
```

```
$`1003_s_at`
[1] "CXCR5"

> ##And here is a mapping that retrieves all known gene symbols
> mget(probes, revmap(hgu95av2ALIAS2PROBE), ifnotfound=NA)

$`1000_at`
[1] "ERK1"     "HS44KDAP" "HUMKER1A" "MAPK3"    "MGC20180" "P44ERK1"  "P44MAPK"
[8] "PRKM3"

$`1001_at`
[1] "JTK14" "TIE"   "TIE1"

$`1002_f_at`
[1] "CPCJ"      "CYP2C"     "CYP2C19"   "P450C2C"   "P450IIC19"

$`1003_s_at`
[1] "BLR1"      "CD185"     "CXCR5"     "MDR15"     "MGC117347"

> ##Be careful with Aliases as they are NOT unique
> ##This is easier to demonstrate with an org package.
> ##But its even more of a problem in a chip package.
> library(org.Hs.eg.db)
> EG2AliasList = as.list(org.Hs.egALIAS2EG)
> EG2AliasList["KAT"]

$KAT
[1] "10300"     "50848"     "100134860"

> ##We can calculate out how many things match each Alias like this:
> lengths = unlist(lapply(EG2AliasList, length))
> lengths["KAT"]

KAT
  3

> table(lengths)

lengths
    1     2     3     4     5     6     7     8     9    10    12    15    36
99536  2933   364    97    25    10    10     4     3     1     1     1     1
```

```
> ##And just out of curiosity:
> lengths[lengths==36]

VH
36

> EG2AliasList["VH"]

$VH
 [1] "3507"  "6545"  "28385" "28388" "28391" "28392" "28394" "28395" "28400"
[10] "28401" "28408" "28409" "28410" "28412" "28414" "28420" "28423" "28424"
[19] "28426" "28429" "28432" "28434" "28439" "28444" "28445" "28447" "28448"
[28] "28450" "28451" "28452" "28454" "28455" "28457" "28464" "28466" "28467"
```

4 Creating a custom chip based package

Sometimes you may wish that you had a package that mapped the probes for a currently unsupported platform. When this happens you can make a custom package that conforms to the standard chip package database schemas by using SQLForge. To guarantee that the package you make matches the other annotation packages in a given release, you have to 1st install the .db0 package that goes with the organism you wish to make a package for. Then you just have to call the appropriate function in AnnotationDbi

```
> ##1st you need the appropriate DB0 package
> library(AnnotationDbi)
> available.db0pkgs()

 [1] "arabidopsis.db0" "bovine.db0"      "canine.db0"      "chicken.db0"
 [5] "ecoliK12.db0"    "ecoliSakai.db0"  "fly.db0"         "human.db0"
 [9] "malaria.db0"     "mouse.db0"       "pig.db0"         "rat.db0"
[13] "worm.db0"        "yeast.db0"       "zebrafish.db0"

> ##Then you need to get that package
> ##But Don't actually do this step (if you are copy/pasting along)
> # source("http://bioconductor.org/BiocLite.R")
> # biocLite("human.db0")
>
> ##Then you can get a tab-delimited file that has your probes paired with IDs
> hcg110_IDs = system.file("extdata", "hcg110_ID", package="AnnotationDbi")
> head(read.delim(hcg110_IDs,header=FALSE))
```

```
          V1     V2
1   1000_at X60188
2   1001_at X60957
3 1002_f_at X65962
4 1003_s_at X68149
5   1004_at X68149
6   1005_at X68277
```

```
> ##For this example lets not actually write anything to the file sys.
> tmpout = tempdir()
> ##Then you can make the package
> makeHUMANCHIP_DB(affy=FALSE,
+                  prefix="hcg110",
+                  fileName=hcg110_IDs,
+                  baseMapType="gb",
+                  outputDir = tmpout,
+                  version="1.0.0",
+                  manufacturer = "Affymetrix",
+                  chipName = "Human Cancer G110 Array",
+                  manufacturerUrl = "http://www.affymetrix.com")
```

```
baseMapType is gb or gbNRefPrepending MetadataCreating Genes tableAppending ProbesFou

Creating package in /tmp/RtmppH4kiv/hcg110.db
```

5 Using specialized annotation packages

Sometimes more specialized data is needed that is not necessarily affiliated with a particular organism. Examples of this are GO.db, KEGG.db and PFAM.db. We will demonstrate using GO.db. Before we start it is important to remember that GO is the gene ontology. Which means that there are parent-child relationships between terms within the ontology.

```
> ##You may have already noticed that the organism packages have some GO
> ##information in them already.  This mapping represents the relationship
> ##between these EG IDs and the GO IDs.  For example: org.Hs.egGO,
> ##org.Hs.egGO2EG, and org.Hs.egGO2ALLEGS
> ls("package:org.Hs.eg.db")[22:24]
```

```
[1] "org.Hs.egGO"        "org.Hs.egGO2ALLEGS" "org.Hs.egGO2EG"
```

```
> ##There are two types of such mappings. org.Hs.egGO will map GO terms to
> ##entrez gene IDs, while org.Hs.egGO2ALLEGS maps GO terms and relevant child
> ##terms to specific entrez gene IDs The man pages will help you remember which
> ##is which.
>
>
> ##All the other GO information is found in GO.db
> library(GO.db)
> ls("package:GO.db")

 [1] "GO"            "GO_dbconn"     "GO_dbfile"     "GO_dbInfo"
 [5] "GO_dbschema"   "GOBPANCESTOR"  "GOBPCHILDREN"  "GOBPOFFSPRING"
 [9] "GOBPPARENTS"   "GOCCANCESTOR"  "GOCCCHILDREN"  "GOCCOFFSPRING"
[13] "GOCCPARENTS"   "GOMAPCOUNTS"   "GOMFANCESTOR"  "GOMFCHILDREN"
[17] "GOMFOFFSPRING" "GOMFPARENTS"   "GOOBSOLETE"    "GOSYNONYM"
[21] "GOTERM"

> ##The mapping that you usually want is the one that describes all the terms
> keys = keys(GOTERM[1:500])
> x = mget(as.character(keys), GOTERM, ifnotfound=NA)
> x[30]

$`GO:0000038`
GOID: GO:0000038
Term: very-long-chain fatty acid metabolic process
Ontology: BP
Definition: The chemical reactions and pathways involving fatty acids
    with a chain length of C18 or greater.
Synonym: very-long-chain fatty acid metabolism

> ##GO is a directed acyclic graph, so there are many parent and child
> ##relationships among terms.
> ##Therefore GO.db also has mappings to tell you about the parent and child
> ##terms as well as all the ancestor or offspring terms
> mget(as.character(names(x[30])),GOBPCHILDREN, ifnotfound=NA)

$`GO:0000038`
        isa          isa
"GO:0042760" "GO:0042761"
```

6 Using biomaRt

If you can't find what you are looking for in the annotation packages, you can also consider trying biomaRt. biomaRt is slower, not versioned, and requires a greater level of knowledge to use, but sometimes there is information there that is not included in the annoation packages yet. One thing to pay attention to is that the biomaRt ensembl database used in this example sometimes a different source of annotations from the annotation packages above for sequence data. We therefore recommend against mixing and matching these two annotation sets as there might be disagreements.

Remember also when using biomaRt, that it has to talk to an external server most of the time. So you may have to repeat some of the following steps if the internet is not cooperating.

```
> ##Getting the data from biomaRt:
>
> library("biomaRt")
> ##Choose a database
> listMarts()[1:5,]
```

```
  biomart                                                   version
1 ensembl                               ENSEMBL 53 GENES (SANGER UK)
2     snp                          ENSEMBL 53 VARIATION  (SANGER UK)
3    vega                                      VEGA 34  (SANGER UK)
4     msd                                   MSD PROTOTYPE (EBI UK)
5    htgt HIGH THROUGHPUT GENE TARGETING AND TRAPPING (SANGER UK)
```

```
> ##Get the current ensembl database.
> ensembl = useMart("ensembl")
> ##List the datasets therein
> listDatasets(ensembl)[1:10,]
```

```
                 dataset                                 description
1   oanatinus_gene_ensembl    Ornithorhynchus anatinus genes (OANA5)
2    tguttata_gene_ensembl Taeniopygia guttata genes (ZEBRA_FINCH_1)
3  cporcellus_gene_ensembl          Cavia porcellus genes (cavPor3)
4  gaculeatus_gene_ensembl    Gasterosteus aculeatus genes (BROADS1)
5   lafricana_gene_ensembl        Loxodonta africana genes (loxAfr2)
6    agambiae_gene_ensembl          Anopheles gambiae genes (AgamP3)
7  mlucifugus_gene_ensembl        Myotis lucifugus genes (MICROBAT1)
8    hsapiens_gene_ensembl               Homo sapiens genes (NCBI36)
```

```
9  choffmanni_gene_ensembl       Choloepus hoffmanni genes (SLOTH_1)
10   aaegypti_gene_ensembl              Aedes aegypti genes (AaegL1)
         version
1          OANA5
2  ZEBRA_FINCH_1
3         cavPor3
4        BROADS1
5        loxAfr2
6         AgamP3
7      MICROBAT1
8         NCBI36
9        SLOTH_1
10        AaegL1
```

```
> ##Then set up so that you use that for this session
> ##(we will choose the mouse one from NCBI build 37.1):
> ensembl = useDataset("mmusculus_gene_ensembl",mart=ensembl)
> ##List attributes
> attributes = listAttributes(ensembl)
> attributes[1:10,]
```

```
                             name                      description
1                 ensembl_gene_id                  Ensembl Gene ID
2           ensembl_transcript_id            Ensembl Transcript ID
3              ensembl_peptide_id               Ensembl Protein ID
4  canonical_transcript_stable_id Canonical transcript stable ID(s)
5                     description                      Description
6                 chromosome_name                  Chromosome Name
7                  start_position                  Gene Start (bp)
8                    end_position                    Gene End (bp)
9                          strand                           Strand
10                           band                             Band
```

```
> ##And filters
> filters = listFilters(ensembl)
> filters[1:10,]
```

```
               name                              description
1      chromosome_name                          Chromosome name
2               start                          Gene Start (bp)
3                 end                            Gene End (bp)
```

```
4          band_start                                  Band Start
5            band_end                                    Band End
6        marker_start                                Marker Start
7          marker_end                                  Marker End
8              strand                                      Strand
9   chromosomal_region                         Chromosome Regions
10 with_affy_mu11ksuba with Affymetrix Microarray mu11ksuba ID(s)
```

```
> ##Some entrez gene IDs
> EGs = c("18392","18414","56513")
> ##1st a Simple example to just get some gene names:
> getBM(attributes = "external_gene_id",
+       filters = "entrezgene",
+       values = EGs,
+       mart=ensembl)
```

```
  external_gene_id
1           Orc1l
2            Osmr
3          Pard6a
```

```
> ##Transcript starts and ends:
> getBM(attributes = c("entrezgene","transcript_start","transcript_end"),
+       filters = "entrezgene",
+       values = EGs,
+       mart=ensembl)
```

```
  entrezgene transcript_start transcript_end
1      18392        108252066      108288633
2      18414          6763590        6824283
3      56513        108225054      108227393
4      56513        108225571      108227393
5      56513        108225571      108227262
```

7 Session Information

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

```
R version 2.10.0 Under development (unstable) (2009-04-16 r48329)
x86_64-unknown-linux-gnu
```

```
locale:
LC_CTYPE=en_US.UTF-8;LC_NUMERIC=C;LC_TIME=en_US.UTF-8;LC_COLLATE=en_US.UTF-8;LC_MONET

attached base packages:
[1] stats     graphics  grDevices datasets  utils     methods   base

other attached packages:
[1] biomaRt_1.99.9      GO.db_2.2.11        hgu95av2.db_2.2.11
[4] org.Hs.eg.db_2.2.11 RSQLite_0.7-1       DBI_0.2-4
[7] AnnotationDbi_1.7.0 Biobase_2.3.11

loaded via a namespace (and not attached):
[1] RCurl_0.94-1 tools_2.10.0 XML_2.3-0
```