## Bioconductor annotation packages

Major types of annotation in Bioconductor.

AnnotationDbi packages:

- Organism level: org.Mm.eg.db.
- Platform level: hgu133plus2.db.
- System-biology level: GO.db or KEGG.db.

biomaRt:

- Query web-based ‘biomart’ resource for genes, sequence, SNPs, and etc.

Other packages:

- rtracklayer – export to UCSC web browsers.
- GenomicFeatures – coming soon for transcripts.

## AnnotationDbi

AnnotationDbi is a software package that enables the package annotations:

- Each supported package contains a database.
- AnnotationDbi allows access to that data via Bimap objects.
- Some databases depend on the databases in other packages.

## Organism-level annotation

There are a number of organism annotation packages with names starting with org, e.g., org.Hs.eg.db – genome-wide annotation for human.

```
> library(org.Hs.eg.db)
> org.Hs.eg()
> org.Hs.eg_dbInfo()
> org.Hs.egGENENAME
> org.Hs.eg_dbschema()
```

## platform based packages (chip packages)

There are a number of platform or chip specific annotation packages named after their respective platforms, e.g. hgu95av2.db annotations for the hgu95av2 Affymetrix platform.

- These packages appear to contain a lot of data but it's an illusion.

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

## Kinds of annotation

What can you hope to extract from an annotation package?

- GO IDs: GO
- KEGG pathway IDs: PATH
- Gene Symbols: SYMBOL
- Chromosome start and stop locs: CHRLOC and CHRLOCEND
- Alternate Gene Symbols: ALIAS
- Associated Pubmed IDs: PMID
- RefSeq IDs: REFSEQ
- Unigene IDs: UNIGENE
- PFAM IDs: PFAM
- Prosite IDs: PROSITE
- ENSEMBL IDs: ENSEMBL

## Basic Bimap structure and getters

Bimaps create a mapping from one set of keys to another. And they can easily be searched.

- `toTable`: converts a Bimap to a data.frame
- `get`: pulls data from a Bimap
- `mget`: pulls data from a Bimap for multiple things at once

```
> head(toTable(hgu95av2SYMBOL))
> get("38187_at",hgu95av2SYMBOL)
> mget(c("38912_at","38187_at"),hgu95av2SYMBOL,ifnotfound=NA)
```

## Reversing and subsetting Bimaps

Bimaps can also be reversed and subsetted:

- `revmap`: reverses a Bimap
- `[[`,`[`: Bimaps are subsettable.

```
> ##revmap
> mget(c("NAT1","NAT2"),revmap(hgu95av2SYMBOL),ifnotfound=NA)
> ##subsetting
> head(toTable(hgu95av2SYMBOL[1:3]))
> hgu95av2SYMBOL[["1000_at"]]
> revmap(hgu95av2SYMBOL)[["MAPK3"]]
> ##Or you can combine things
> toTable(hgu95av2SYMBOL[c("38912_at","38187_at")])
```

## using merge, cbind

sometimes you will want to combine data

- `cbind`: appends multiple columns (blindly by order)
- `merge`: "joins" a pair of data.frames based on a key

```
> ## 1st lets get some data
> symbols = head(toTable(hgu95av2SYMBOL),n=3)
> chrlocs = head(toTable(hgu95av2CHRLOC),n=3)
> pmids = head(toTable(hgu95av2PMID),n=3)
> ##cbind
> cbind(symbols, pmids, chrlocs)
> ##merge
> merge(symbols, pmids, by.x="probe_id", by.y="probe_id")
```

## Annotation exercise 1

Find the gene symbol, chromosome position and KEGG pathway ID for "1003_s_at".

## Annotation exercise 1 solution

```
> get("1003_s_at",hgu95av2SYMBOL)
> get("1003_s_at",hgu95av2CHRLOC)
> get("1003_s_at",hgu95av2PATH)
```

## Bimap keys

Bimaps create a mapping from one set of keys to another. Some important methods include:

- `keys`: centralID for the package (directional)
- `Lkeys`: centralID for the package (probe ID or gene ID)
- `Rkeys`: centralID for the package (attached data)

```
> keys(hgu95av2SYMBOL[1:4])
> Lkeys(hgu95av2SYMBOL[1:4])
> Rkeys(hgu95av2SYMBOL)[1:4]
```

## More Bimap structure

Not all keys have a partner (or are mapped)

- `mappedkeys`: which of the key are mapped (directional)
- `mappedLkeys` `mappedRkeys`: which keys are mapped (absolute reference)
- `count.mappedkeys`: Number of mapped keys (directional)
- `count.mappedLkeys`,`count.mappedRkeys`: Number of mapped keys (absolute)

```
> mappedkeys(hgu95av2SYMBOL[1:10])
> mappedLkeys(hgu95av2SYMBOL[1:10])
> mappedRkeys(hgu95av2SYMBOL[1:10])
> count.mappedkeys(hgu95av2SYMBOL[1:100])
> count.mappedLkeys(hgu95av2SYMBOL[1:100])
> count.mappedRkeys(hgu95av2SYMBOL[1:100])
```

## Bimap Conversions

How to handle conversions from Bimaps to lists

- `as.list`: converts a Bimap to a list
- `unlist2`: unlists a list minus the name-mangling.

```
> as.list(hgu95av2SYMBOL[c("38912_at","38187_at")])
> unlist(as.list(hgu95av2SYMBOL[c("38912_at","38187_at")]))
> unlist2(as.list(hgu95av2SYMBOL[c("38912_at","38187_at")]))
> ##but what happens when there are
> ##repeating values for the left key?
> unlist(as.list(revmap(hgu95av2SYMBOL)[c("STAT1","PTGER3")]))
> ##unlist2 can help with this
> unlist2(as.list(revmap(hgu95av2SYMBOL)[c("STAT1","PTGER3")]))
```

## Annotation exercise 2

Gene symbols are often recycled by other genes making them a poor choice for identifiers. Using what you have learned, the SYMBOL Bimap, along with the `lapply`, `length` and `sort` functions, determine which gene symbols in hgu95av2 are the worst offenders.

## Annotation exercise 2 solution

```
> badRank <- lapply(as.list(revmap(hgu95av2SYMBOL)), length)
> tail(sort(unlist(badRank)))
```

## toggleProbes

How to hide/unhide ambiguous probes.

- `toggleProbes`: hides or displays the probes that have multiple mappings to genes.

```
> ## How many probes?
> dim(hgu95av2ENTREZID)
> ## Make a mapping with multiple probes exposed
> multi <- toggleProbes(hgu95av2ENTREZID, "all")
> ## How many probes?
> dim(multi)
> ## Make a mapping with ONLY multiple probes exposed
> multiOnly <- toggleProbes(multi, "multiple")
> ## How many probes?
> dim(multiOnly)
> ## Then make a mapping with ONLY single mapping probes
> singleOnly <- toggleProbes(multiOnly, "single")
> ## How many probes?
> dim(singleOnly)
```

## Annotation exercise 3

Using the knowledge that Entrez IDs are good IDs that can be used to define genes uniquely, find the probe that maps to the largest number of different genes on hgu95av2.

## Annotation exercise 3 solution

```
> mult <- toggleProbes(hgu95av2ENTREZID, "multi")
> dim(mult)
> multRank <- lapply(as.list(mult), length)
> tail(sort(unlist(multRank)))
```

## GO

Some important considerations about the Gene Ontology

- GO is actually 3 ontologies (CC, BP and MF)
- Each ontology is a directed acyclic graph.
- The structure of GO is maintained separarately from the genes that these GO IDs are usually used to annotate.

## GO to gene mappings are stored in other packages

Mapping Entrez IDs to GO

- Each ENTREZ ID is associated with up to three GO categories.
- The objects returned from an ordinary GO mapping are complex.

```
> go <- org.Hs.egGO[["1000"]]
> length(go)
> go[[2]]$GOID
> go[[2]]$Ontology
```

## Annotation exercise 4

Use what they have learned to write a function that gets the GOIDs for a particular entrez gene ID, and then returns only their GOID as a named vector. Use `lapply` and `(names)`.

## Annotation exercise 4 solution

```
> ##get GOIDs from Hs package.
> getGOIDs <- function(ids){
+     require(org.Hs.eg.db)
+     GOs = mget(ids, org.Hs.egGO, ifnotfound=NA)
+     unlist2(lapply(GOs,names))
+ }
> ##usage example:
> getGOIDs(c("1","10"))
```

## Working with GO.db

- Encodes the hierarchical structure of GO terms.
- The mapping between GO terms and individual genes is maintained in the GO mappings from the other packages.
- the difference between children and offspring is how many generations are represented. Children only nets you one step down the graph.

```
> library(GO.db)
> ls("package:GO.db")
> ## find children
> as.list(GOMFCHILDREN["GO:0008094"])
> ## all the descendants (children, grandchildren, and so on)
> as.list(GOMFOFFSPRING["GO:0008094"])
```

## GO helper methods

Using the GO helper methods

- The GO terms are described in detail in the GOTERM mapping.
- The objects returned by GO.db are GOTerms objects, which can make use of helper methods like `GOID`, `Term`, `Ontology` and `Definition` to retrieve various details.
- You can also pass GOIDs to these helper methods.

```
> ##Mapping a GOTerms object
> go <- GOTERM[1]
> GOID(go)
> Term(go)
> ##OR you can supply GO IDs
> id = c("GO:0007155","GO:0007156")
> GOID(id)
> Term(id)
> Ontology(id)
> Definition(id)
```

## Annotation exercise 5

Use what you have learned to write a function that calls your previous function so that it can get GOIDs and then returns the GO definitions.

## Annotation exercise 5 solution

```
> ##get GOIDs from Hs package.
> getGODefs <- function(ids){
+     GOids <- getGOIDs(ids)
+     defs <- Definition(GOids)
+     names(defs) <- names(GOids)
+     defs
+ }
> ##usage example:
> getGODefs(c("1","10"))
```

## Using biomaRt

Setting up a biomaRt object

- biomaRt offers several "marts" to get data from
- each "mart" can have several datasets
- the mart object has to be configured with your choices

```
> library(biomaRt)
> ##list the marts
> head(listMarts())
> ## list the Datasets for a mart
> head(listDatasets(useMart("ensembl")))
> ## now set up the fully qualified mart object
> ensembl <- useMart("ensembl", dataset = "hsapiens_gene_ensembl
```

## Using biomaRt

Choosing biomaRt options

- filters are used to limit the query
- values are the values available for a specified filter
- attributes are information we want to retrieve

```
> ## need to be able to list filters
> head(listFilters(ensembl))
> myFilter <- "chromosome_name"
> ## and list values that you expect back
> head(filterOptions(myFilter, ensembl))
> myValues <- c("21", "22")
> ## and list attributes
> head(listAttributes(ensembl))
> myAttributes <- c("ensembl_gene_id","chromosome_name")
```

## Using biomaRt

Calling `getBM` will extract the information

- `getBM` takes the information we have just shown you how to obtain as its parameters.
- With the exception of the mart object all these parameters are vectors so you can request multiple values back if they are available etc.
- If you should need to specify multiple filters, then you will need to pass the values parameter in as a list of vectors instead of just a vector.

```
> ## then you can assemble a query
> res <- getBM(attributes =  myAttributes,
+              filters =  myFilter,
+              values =  myValues,
+              mart = ensembl)
> head(res)
```

## Annotation exercise 6

Use what you have learned about biomaRt to find the gene symbol and name for the entrez gene IDs 1, 10 and 100.

## Annotation exercise 6 solution

```
> res <- getBM(attributes =  c("entrezgene","hgnc_symbol"),
+              filters =  "entrezgene",
+              values =  c("1","10","100"),
+              mart = ensembl)
> head(res)
```

## Annotation exercise 7

Use what you have learned to add annotations to the dataset that you used in the earlier session with Chao Jen with gene symbols. The code below will re-load the dataset into your R session. The annotations for this package are found in the hgu95av2.db chip package.

```
> load(system.file("data", "result.rda",
+                  package = "SeattleIntro2010"))
```

## Annotation exercise 7 solution

```
> ids <- result[,1]
> library(hgu95av2.db)
> merge(result, toTable(hgu95av2SYMBOL[ids]), by.x="ID", by.y="p
```

## Annotation exercise 8 (bonus round)

Use what you have learned about Annotations to come up with a more useful exercise to YOU, and a solution. We will be here to help.