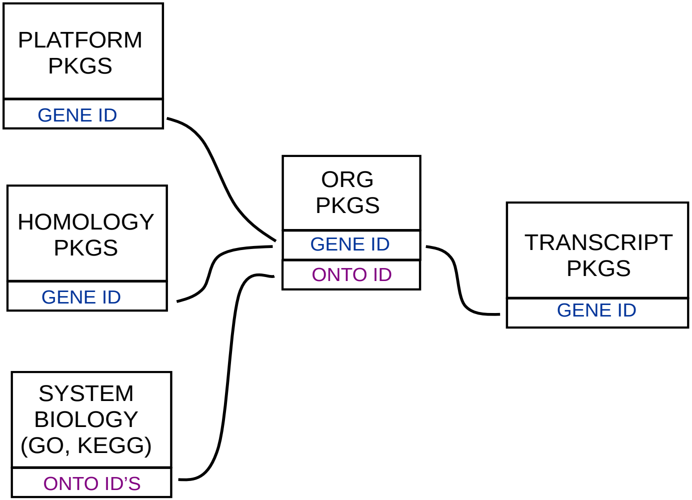
Annotation Packages: the big picture



Bioconductor annotation packages

Major types of annotation in Bioconductor.
Gene centric 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.

Genome centric GenomicFeatures packages:

- Transriptome level: TxDb.Hsapiens.UCSC.hg19.knownGene
- Generic features: Can generate via GenomicFeatures

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()
```

Gene centric annotations

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")
```

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.data.frame`: converts a Bimap to a data.frame
- `toTable`: converts a Bimap to a data.frame

```
> 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")]))
```

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)
```

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
```

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)
```

Working with other packages

- will contain unique kinds of data.
- there should be manual pages for all the different mappings.

```
> library("targetscan.Hs.eg.db")
> # help
> # ?targetscan.Hs.egTARGETS
> tab = toTable(targetscan.Hs.egTARGETS)
> head(tab[tab[,"name"]=="miR-187",])
> ## or you could just use the get method
> geneTargets <- get("miR-187", revmap(targetscan.Hs.egTARGETS))
```

Connecting data between packages

- pay attention to the foreign keys (geneTargets was an EG ID)
- then use those keys as input for the next piece of data you seek
- for advanced users: it is possible to join between packages

```
> library(org.Hs.eg.db)
> gos <- toTable(org.Hs.egGO)
> head(gos[gos[,"gene_id"] %in% geneTargets,])
> ## or alternatively you can generate lists of answers:
> unlist(mget(geneTargets, org.Hs.egGO)[1])[1:6]
> unlist2(mget(geneTargets, org.Hs.egGO)[1])[1:6]
```

Creating packages

- `available.dbschemas` to discover supported organisms
- `makeDBPackage` to create new chip packages
- `makeDBPackage` requires probe-gene mapping data

```
> ## Discover available schemas
> available.dbschemas()
> ## Create a package
> makeDBPackage("HUMANCHIP_DB",
+               affy = TRUE,
+               prefix = "hgu95av2",
+               fileName = "/srcFiles/hgu95av2/HG_U95Av2_annot.c
+               otherSrc = c(
+                 EA="/srcFiles/hgu95av2/hgu95av2.EA.txt",
+                 UMICH="/sqliteGen/srcFiles/hgu95av2/hgu95av2_U
+               baseMapType = "gbNRef",
+               version = "1.0.0",
+               manufacturer = "Affymetrix",
+               chipName = "hgu95av2",
+               manufacturerUrl = "http://www.affymetrix.com")
```

makeOrgPackageFromNCBI

- `makeOrgPackageFromNCBI` generates an org package
- Requires that you have an NCBI Taxonomy ID

```
> makeOrgPackageFromNCBI(version = "0.1",
+                        author = "Some One <so@someplace.org>",
+                        maintainer = "Some One <so@someplace.or
+                        outputDir = ".",
+                        tax_id = "59729",
+                        genus = "Taeniopygia",
+                        species = "guttata")
```

AnnotationDb Objects

- Loading the package will create this
- Will be named after the package
- Some useful accesors include:
  - `keytypes`
  - `keys`
  - `cols`
  - `select`

AnnotationDb Objects

```
> org.Hs.eg.db
```

```
OrgDb object:
| DBSCHEMAVERSION: 2.1
| Db type: OrgDb
| package: AnnotationDbi
| DBSCHEMA: HUMAN_DB
| ORGANISM: Homo sapiens
| SPECIES: Human
| EGSOURCEDATE: 2011-Sep14
| EGSOURCENAME: Entrez Gene
| EGSOURCEURL: ftp://ftp.ncbi.nlm.nih.gov/gene/DATA
| CENTRALID: EG
| TAXID: 9606
| GOSOURCENAME: Gene Ontology
| GOSOURCEURL: ftp://ftp.geneontology.org/pub/go/godatabase/arch
| GOSOURCEDATE: 20110910
| GOEGSOURCEDATE: 2011-Sep14
| GOEGSOURCENAME: Entrez Gene
| GOEGSOURCEURL: ftp://ftp.ncbi.nlm.nih.gov/gene/DATA
| KEGGSOURCENAME: KEGG GENOME
```

Using AnnotationDb accessors

```
> head( cols(org.Hs.eg.db) )

[1] "ACCNUM"   "ALIAS2EG" "CHR"      "ENZYME"   "GENENAME"
[6] "MAP"

> head( keytypes(org.Hs.eg.db) )

[1] "ACCNUM"   "ALIAS2EG" "CHR"      "ENZYME"   "GENENAME"
[6] "MAP"

> head( keys(org.Hs.eg.db) )

[1] "1"  "2"  "3"  "9"  "10" "11"

> c = c("SYMBOL", "CHR")
> k = keys(org.Hs.eg.db)[1:4]
> select(org.Hs.eg.db, keys=k, cols=c)

  gene_id symbol chromosome
1       1   A1BG         19
2       2    A2M         12
3       3  A2MP1         12
4       9   NAT1          8
```

Using Select with another keytype

```
> c = c("SYMBOL", "CHR")
> kt = c("SYMBOL")
> k = head(Rkeys(org.Hs.egSYMBOL))
> select(org.Hs.eg.db, keys=k, cols=c, keytype=kt)

      gene_id symbol chromosome
1           1   A1BG         19
16693       2    A2M         12
21882       3  A2MP1         12
40756       9   NAT1          8
2          10   NAT2          8
12837      11   AACP          8
```

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)
```