Bioconductor: Annotation Package Overview

April 30, 2018

1 Overview

In its current state the basic purpose of annotate is to supply interface routines that support user actions that rely on the different meta-data packages provided through the Bioconductor Project. There are currently four basic categories of functions that are contained in annotate.

- Interface functions for getting data out of specific meta-data libraries.
- Functions that support querying the different web services provided by the National Library of Medicine (NLM), and the National Center for Biotechnology Information (NCBI).
- Functions that support organizing and structuring chromosomal location data to support some of the gene plotting and gene finding routines in geneplotter.
- Functions that produce HTML output, hyperlinked to different web resources, from gene lists.

We will briefly describe the second and third of these different aspects and then for the remainder of this vignette concentrate on the first category. The other three have their own vignettes.

There are two different, but complementary strategies for accessing meta-data. One is to use highly curated data that have been assembled from many different sources and a second is to rely on on-line sources. The former tends to be less current but more comprehensive while the latter tends to be current but can be less comprehensive and difficult to reproduce as the sources themselves are constantly evolving. To address the second of these we develop and distribute software that can take advantage of the web services that are provided. Perhaps the richest source of these is provided by the National Library of Medicine. Further details on accessing these resources are provided in Gentleman and Gentry (2002) and Gentry (2004b).

While the chromosomal location is not always of interest, in certain situations, especially the study of genetic diseases it is important that we be able to associate particular genes with locations on chromosomes. We provide a
complete set of functions that map from LocusLink identifiers to chromosomal location. Examples and further discussion are provided in Gentry (2004a).

Producing output that helps users navigate and understand the results of an analysis is a very important aspect of any data analysis. Since one of the primary tasks that is carried out when analysing gene expression data is to create lists of interesting genes we have provided some simple infrastructure that will help produce a hyperlinked output page for any given set of genes. A more substantial and comprehensive approach has been taken by C. Smith in the annaffy package. A vignette for using the functions provided in the annotate package is provided in Gentleman (2003).

We now turn our attention to interfaces to the meta-data packages and how and when they will be useful. The annotation library provides interface support for the different meta-data packages that are available through the Bioconductor Project. We have tried to make these different meta-data packages modular, in the sense that all of them should have similar sets of mappings from manufacturer IDs to specific biological data such as chromosomal location, GO, and LocusLink identifiers.

Annotation in the Bioconductor Project is handled by two systems. One, AnnBuilder is a system for assembling and relating the data from various sources. It is much more industrial and takes advantage of many different non-R tools such as Perl and XML. The second package is annotate. This package is designed to provide experiment level annotation resources and to help users associate the outputs of AnnBuilder with their particular data.

There will be some need for the structure of the meta-data packages to evolve over time and by making use of the functions provided in annotate users and developers should shield themselves from many of the changes. We will endeavor to keep the annotate interfaces constant.

Any given experiment typically involves a set of known identifiers (probes in the case of a microarray experiment). These identifiers are typically unique (for any manufacturer). This holds true for any of the standard databases such as LocusLink. However, when the identifiers from one source are linked to the identifiers from another there does not need to be a one-to-one relationship. For example, several different Affymetrix accession numbers correspond to a single LocusLink identifier. Thus, when going one direction (Affymetrix to LocusLink) we have no problem, but when going the other we need some mechanism for dealing with the multiplicity of matches.

A Short Example

We will consider the Affymetrix human gene chip, hgu95av2, for our example. We first load this chip’s package and annotate.

```r
> library("annotate")
> library("hgu95av2.db")
> ls("package:hgu95av2.db")
```
We see the listing of twenty or thirty different R objects in this package. Most of them represent mappings from the identifiers on the Affymetrix chip to the different biological resources and you can find out more about them by using the R help system, since each has a manual page that describes the data together with other information such as where, when and what files were used to construct the mappings. Also, each meta-data package has one object that has the same name as the package basename, in this case it is `hgu95av2`. This is function and it can be invoked to find out some of the different statistics regarding the mappings that were done. Its manual page lists all data resources that were used to create the meta-data package.

```r
> hgu95av2()
```

Quality control information for hgu95av2:

This package has the following mappings:

- `hgu95av2ACCNUM` has 12625 mapped keys (of 12625 keys)
- `hgu95av2ALIAS2PROBE` has 34238 mapped keys (of 121608 keys)
- `hgu95av2CHR` has 11472 mapped keys (of 12625 keys)
- `hgu95av2CHRLENGTHS` has 93 mapped keys (of 455 keys)
- `hgu95av2CHRLOC` has 11423 mapped keys (of 12625 keys)
- `hgu95av2CHRLOCEND` has 11423 mapped keys (of 12625 keys)
- `hgu95av2ENSEMBL` has 11365 mapped keys (of 12625 keys)
- `hgu95av2ENSEMBL2PROBE` has 9545 mapped keys (of 28589 keys)
- `hgu95av2ENTREZID` has 11474 mapped keys (of 12625 keys)
- `hgu95av2ENZYME` has 2097 mapped keys (of 12625 keys)
- `hgu95av2ENZYME2PROBE` has 779 mapped keys (of 975 keys)
- `hgu95av2GENENAME` has 11474 mapped keys (of 12625 keys)
- `hgu95av2GO` has 11229 mapped keys (of 12625 keys)
- `hgu95av2GO2ALLPROBES` has 18521 mapped keys (of 22436 keys)
- `hgu95av2GO2PROBE` has 13971 mapped keys (of 17669 keys)
- `hgu95av2MAP` has 11450 mapped keys (of 12625 keys)
- `hgu95av2MAPCOUNTS` has 1040 mapped keys (of 1040 keys)
- `hgu95av2OMIM` has 11474 mapped keys (of 12625 keys)
- `hgu95av2PATH` has 9104 mapped keys (of 9104 keys)
- `hgu95av2PATH2PROBE` has 1797 mapped keys (of 1797 keys)
- `hgu95av2PFAM` has 39 mapped keys (of 39 keys)
- `hgu95av2PMID` has 11472 mapped keys (of 12625 keys)
- `hgu95av2PMID2PROBE` has 960 mapped keys (of 960 keys)
- `hgu95av2PROSITE` has 2097 mapped keys (of 2097 keys)
- `hgu95av2REFSEQ` has 2097 mapped keys (of 2097 keys)
- `hgu95av2SYMBOL` has 11474 mapped keys (of 12625 keys)
- `hgu95av2UNIGENE` has 11474 mapped keys (of 12625 keys)
- `hgu95av2UNIPROT` has 11474 mapped keys (of 12625 keys)
- `hgu95av2_dbconn` has 11474 mapped keys (of 12625 keys)
- `hgu95av2_dbfile` has 11474 mapped keys (of 12625 keys)
- `hgu95av2_dbschema` has 11474 mapped keys (of 12625 keys)
Now let's consider a specific object, say the hgu95av2ENTREZID object.

```r
> hgu95av2ENTREZID

ENTREZID map for chip hgu95av2 (object of class "ProbeAnnDbBimap")

If we type its name we see that it is an R environment; all this means is that it is a special data type that is efficient at storing and retrieving mappings between symbols (the Affymetrix identifiers) and associated values (the LocusLink IDs).

We can retrieve values from this environment in many different ways (and the reader is referred to the R help pages for more details on using some of the functions described briefly here). Suppose that we are interested in finding the LocusLink ID for the Affymetrix probe, 1000_at. Then we can do it in any one of the following ways:

```r
> get("1000_at", env=hgu95av2ENTREZID)

[1] "5595"
```

```r
> hgu95av2ENTREZID["1000_at"]

[1] "5595"
```

```r
> hgu95av2ENTREZID$"1000_at"

[1] "5595"
```
And in all cases we see that the LocusLink identifier is 5595.

If you want to get more than one object from an environment you also have a number of choices. You can extract them one at a time using either a for loop or apply or eapply. It will be more efficient to use mget since it does the retrieval using internal code and is optimized. You may also want to turn the environment into a named list, so that you can perform different list operations on it, this can be done using the function contents or as.list.

```r
> LLs = as.list(hgu95av2ENTREZID)
> length(LLs)
[1] 12625
> names(LLs)[1:10]
[1] "1000_at"  "1001_at"  "1002_f_at"  "1003_s_at"  "1004_at"  "1005_at"
[7] "1006_at"  "1007_s_at"  "1008_f_at"  "1009_at"
```

## 2 Developers Tips

Software developers that are building other tools that might make use of the infrastructure produced here should make use of the get* family of functions. Examples include getEG, getGO and so on. There are two reasons for using these functions.

First, they allow you to implement functionality that is independent of the meta-data packages. Since each of these functions takes two arguments, one a list of the manufacturers ids and the second the basename of the annotation package these functions will provide the correct results for all annotation packages.

A second reason to make use of these interface functions is that they are guaranteed not to change. The underlying structure of the meta-data packages may change and developers that access this directly will find that their code needs to be updated regularly. But developers that make use of these interface functions will find that their code needs much less maintenance.

## 3 Session Information

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

```
R version 3.5.0 (2018-04-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.4 LTS
```
Matrix products: default
BLAS: /home/biocbuild/bbs-3.7-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.7-bioc/R/lib/libRlapack.so

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

attached base packages:
[1] grid stats4 parallel stats graphics grDevices utils
[8] datasets methods base

other attached packages:
[1] GO.db_3.6.0 hgu95av2.db_3.2.3 org.Hs.eg.db_3.6.0
[4] Rgraphviz_2.24.0 graph_1.58.0 xtable_1.8-2
[7] annotate_1.58.0 XML_3.98-1.11 AnnotationDbi_1.42.0
[10] IRanges_2.14.0 S4Vectors_0.18.0 Biobase_2.40.0
[13] BiocGenerics_0.26.0

loaded via a namespace (and not attached):
[1] Rcpp_0.12.16 bit_1.1-12 blob_1.1.1 tools_3.5.0
[5] DBI_0.8 bit64_0.9-7 digest_0.6.15 bitops_1.0-6
[9] RCurl_1.95-4.10 memoise_1.1.0 RSQLite_2.1.0 compiler_3.5.0
[13] pkgconfig_2.0.1

References


