

HowTo: get pretty HTML output for my gene list

James W. MacDonald

October 22, 2008

1 Overview

The intent of this vignette is to show how to make reasonably nice looking HTML tables for presenting the results of a microarray analysis. These tables are a very nice format because you can insert clickable links to various public annotation databases, which facilitates the downstream analysis. In addition, the format is quite compact, can be posted on the web, and can be viewed using any number of free web browsers. One caveat; an HTML table is probably not the best format for presenting the results for *all* of the genes on a chip. For even a small (5000 gene) chip, the file could be 10 Mb or more, which would take an inordinate amount of time to open and view. Also note that the Bioconductor project supplies annotation packages for many of the more popular Affymetrix chips, as well as for many commercial spotted cDNA chips. For chips that have annotation packages, the *annaffy* package is the preferred method for making HTML tables.

To make an annotated HTML table, the only requirement is that we have some sort of annotation data for the microarray that we are using. Most manufacturers supply data in various formats that can be read into *R*. For instance, Affymetrix supplies CSV files that can be read into *R* using the `read.csv()` function <http://www.affymetrix.com/support/technical/byproduct.affx?cat=arrays>.

Another alternative is to annotate using functionality in the *biomaRt* package. This allows one to get the most current annotations interactively. In addition, the output can be used directly with functions in *annotate* to make HTML tables. We will use these functions in this vignette.

2 Data Analysis

I will assume that the reader is familiar with the analysis of microarray data, and has a set of genes that she would like to use. In addition, I will assume that the reader is familiar enough with *R* that she can subset the data based on a list of genes, and reorder based on a particular statistic. For any questions about subsetting or ordering data, please see “An Introduction to *R*”. For questions regarding microarray analysis, please consult the vignettes for, say *limma*, *multtest*, or *marray*.

3 Getting Started

We first load the *annotate* package, as well as some data. These data will be from the Affymetrix HG-U95Av2 chip (for which we would normally use *annaffy*). To keep the HTML table small, we will take a subset of fifteen genes as an example.

```
> library("annotate")
> data(sample.ExpressionSet)
> igenes <- featureNames(sample.ExpressionSet)[246:260]
```

We also have to load the *biomaRt* package and connect to a Biomart database, using the `useMart` function. Note that there are two interfaces that *biomaRt* can use to connect to a Biomart database, using either the *RCurl* package to connect via http protocols, or the *RMySQL* package to connect via database protocols. The default is to use *RCurl* because it can be difficult to get *RMySQL* set up on Windows computers. However, I find the database connectivity to be much faster, so for those who want to annotate a large number of genes, I would recommend using the *RMySQL* interface. See the help file for `useMart` for more information.

```
> library("biomaRt")
> mart <- useMart("ensembl", "hsapiens_gene_ensembl")
```

4 Annotation Data

The `htmlpage` function is designed to take two sets of input; data that will be converted to clickable links to various online databases, and data that will simply be put into the HTML table as is. For the clickable links we need an `list` of character vectors for each database. For the data, we need a `list`

of either `vectors`, `data.frames` or a mixture of the two. We will explore this topic more later. First, we will see how to get data using *biomaRt* functionality.

We first need to see exactly what sort of data we can get from Ensembl's Biomart. Note that some of the information can be a bit cryptic, so we can parse out a more reasonable description. We also need to see what things we can use as identifiers in our query of the Biomart server.

First, a bit of terminology. An 'attribute' is a data type that can be returned from a query of a Biomart server. A 'filter' is the identifier that we use to query the server. For instance, we can get GO terms and Entrez Gene IDs (the attributes) by querying on the Affy Probe ID (the filter).

There are too many attributes to list here, so we can parse out things that may be interesting to us. Let's say we want to get Entrez Gene and SwissProt IDs for our set of genes.

```
> attrbuts <- listAttributes(mart)
> attrbuts[grep("swiss", attrbuts[,1]),]

              name              description
103      uniprot_swissprot      UniProt/SwissProt ID
104 uniprot_swissprot_accession UniProt/SwissProt Accession

> attrbuts[grep("entrez", attrbuts[,1]),]

      name      description
40 entrezgene EntrezGene ID
```

So the attributes we want are `uniprot_swissprot_accession` and `entrezgene`.

The same basic idea can be used to figure out which filter to use. Since we are using the HG-U95av2 chip, we need to figure out what that filter is called.

```
> fltr <- listFilters(mart)
> fltr[grep("affy", fltr[,1]),]

      name
1      affy_hc_g110
2      affy_hg_focus
3      affy_hg_u133_plus_2
4      affy_hg_u133a
```

5	affy_hg_u133a_2	
6	affy_hg_u133b	
7	affy_hg_u95a	
8	affy_hg_u95av2	
9	affy_hg_u95b	
10	affy_hg_u95c	
11	affy_hg_u95d	
12	affy_hg_u95e	
13	affy_hugene1	
14	affy_u133_x3p	
96	with_affy_hc_g110	
97	with_affy_hg_focus	
98	with_affy_hg_u133_plus_2	
99	with_affy_hg_u133a	
100	with_affy_hg_u133a_2	
101	with_affy_hg_u133b	
102	with_affy_hg_u95a	
103	with_affy_hg_u95av2	
104	with_affy_hg_u95b	
105	with_affy_hg_u95c	
106	with_affy_hg_u95d	
107	with_affy_hg_u95e	
108	with_affy_huex_1_0_st_v2	
109	with_affy_hugene_1_0_st_v1	
110	with_affy_hugene1	
111	with_affy_u133_x3p	
		description
1	Affy hc g110	ID(s)
2	Affy hg focus	ID(s)
3	Affy hg u133 plus 2	ID(s)
4	Affy hg u133a	ID(s)
5	Affy hg u133a 2	ID(s)
6	Affy hg u133b	ID(s)
7	Affy hg u95a	ID(s)
8	Affy hg u95av2	ID(s)
9	Affy hg u95b	ID(s)
10	Affy hg u95c	ID(s)
11	Affy hg u95d	ID(s)
12	Affy hg u95e	ID(s)
13	Affy hugene1	ID(s)

```

14                                Affy u133 x3p ID(s)
96          with Affymetrix Microarray hc g110 ID(s)
97          with Affymetrix Microarray hg Focus ID(s)
98    with Affymetrix Microarray hg u133 plus 2 ID(s)
99          with Affymetrix Microarray hg u133a ID(s)
100         with Affymetrix Microarray hg u133a 2 ID(s)
101         with Affymetrix Microarray hg u133b ID(s)
102         with Affymetrix Microarray hg u95a ID(s)
103         with Affymetrix Microarray hg u95av2 ID(s)
104         with Affymetrix Microarray hg u95b ID(s)
105         with Affymetrix Microarray hg u95c ID(s)
106         with Affymetrix Microarray hg u95d ID(s)
107         with Affymetrix Microarray hg u95e ID(s)
108    with Affymetrix Microarray huex 1 0 st v2 ID(s)
109 with Affymetrix Microarray hugene 1 0 st v1 ID(s)
110         with Affymetrix Microarray HuGeneFL ID(s)
111         with Affymetrix Microarray u133 x3p ID(s)

```

This one is pretty obvious - we want the `affy_hg_u95av2` filter.

Now to get back to the task at hand; we have 15 Affy Probeset IDs that we want to use to create our HTML table. Let's say we want to create an HTML table in which we map the Affy IDs to Entrez Gene, SwissProt, UniGene, and RefSeq IDs (all clickable links) and in addition we want to include the gene description and symbol, the Gene Ontology terms, and chromosome location, as well as the *t*-statistic, *p*-value, fold change and expression values. That would be a nice compact format for presenting the data to someone.

We need to collect all this information in two lists; one that will be used to make the hyperlinks, and one that will just be static information. First, we do the hyperlinks. By using the ideas presented above, I figured out which attributes correspond to Entrez Gene, SwissProt, UniGene, and RefSeq IDs, so I will use them here. Instead of writing them all out, I will simply select the correct terms using the `listAttributes` function.

```

> genelist <- getBM(attributes = c("affy_hg_u95av2","entrezgene","uniprot_swissprot_a
+                               filter = "affy_hg_u95av2", values = igenes, mart = mart,
+                               output = "list", na.value = "&nbsp;";")
> genelist[[1]] <- igenes
> ## let's look at genelist values
> lapply(genelist, function(x) x[5:10])

```

```
$affy_hg_u95av2
[1] "31489_at" "31490_at" "31491_s_at" "31492_at" "31493_s_at"
[6] "31494_at"
```

```
$entrezgene
$entrezgene$`31489_at`
[1] "&nbsp;"
```

```
$entrezgene$`31490_at`
[1] 6331 731231
```

```
$entrezgene$`31491_s_at`
[1] 841
```

```
$entrezgene$`31492_at`
[1] 27335
```

```
$entrezgene$`31493_s_at`
[1] 1443 1442 1444
```

```
$entrezgene$`31494_at`
[1] "&nbsp;"
```

```
$uniprot_swissprot_accession
$uniprot_swissprot_accession$`31489_at`
[1] "&nbsp;"
```

```
$uniprot_swissprot_accession$`31490_at`
[1] "Q14524"
```

```
$uniprot_swissprot_accession$`31491_s_at`
[1] "Q14790"
```

```
$uniprot_swissprot_accession$`31492_at`
[1] "Q9UBQ5"
```

```
$uniprot_swissprot_accession$`31493_s_at`
[1] "P01243" "Q14406"
```

```

$uniprot_swissprot_accession$`31494_at`
[1] "&nbsp;"

$refseq_dna
$refseq_dna$`31489_at`
[1] "&nbsp;"

$refseq_dna$`31490_at`
[1] "&nbsp;"

$refseq_dna$`31491_s_at`
[1] "NM_001080125" "NM_001228"      "NM_033356"      "NM_001080124"
[5] "NM_033355"      "NM_033358"

$refseq_dna$`31492_at`
[1] "NM_013234"

$refseq_dna$`31493_s_at`
[1] "NM_020991" "NM_022644" "NM_001317" "NM_022641" "NM_022640"
[6] "NM_022579" "NM_001318"

$refseq_dna$`31494_at`
[1] "&nbsp;"

```

Here we can see that `genelist` is a `list` of `lists`, with each sub-list being made up of a character vector. I substituted the `'igenes'` vector back into the first position of the list because the Biomart server we are using doesn't have data for all Affy IDs (including the Affy ID itself). Since we want links for all of the Affy IDs, I simply substituted the original vector into the `genelist`.

Two important things to note about the call to `getBM`. First, we have to use the argument `output = "list"`. Second, we have to use `na.value = " "`, which will create an empty table entry for any missing data. This is much nicer than leaving the NAs, which will tend to clutter up the table without adding any information.

Making the second `list` is a bit more complicated. We need to get some annotation from the Biomart database, and append that to some data we have from our experiment. The first step is to get the annotation data. As noted above, we want the gene name and symbol, as well as the GO terms

and chromosome location for these probesets. We can figure out which attribute terms to use, following the ideas presented above.

```
> attrbutts[grep("description", attrbutts[,1]),]
```

	name	description
32	description	Description
61	interpro_description	Interpro Description
62	interpro_short_description	Interpro Short Description
65	mim_gene_description	MIM Gene Description
67	mim_morbid_description	MIM Morbid Description
302	homologs_description	<NA>
543	sequence_description	<NA>
582	snp_description	<NA>
610	structure_description	<NA>

```
> attrbutts[grep("symbol", attrbutts[,1]),]
```

	name	description
55	hgnc_symbol	HGNC symbol

```
> attrbutts[grep("go", attrbutts[,1]),]
```

	name	description
44	go_biological_process_id	GO ID
45	go_biological_process_linkage_type	Linkage type
46	go_cellular_component_id	GO ID
47	go_cellular_component_linkage_type	Linkage type
48	go_molecular_function_id	GO ID
49	go_molecular_function_linkage_type	Linkage type

```
> attrbutts[grep("^chrom", attrbutts[,1]),]
```

	name	description
26	chromosome_name	Chromosome Name
573	chromosome_location	Chromosome Location (bp)

We want "description", "hgnc_symbol", "go_biological_process.id" and "band"

```
> annotlist <- getBM(attributes = c("description", "hgnc_symbol", "go_biological_proc
+                               filter = "affy_hg_u95av2", values = igenes, mart = mart,
+                               output = "list", na.value = "&nbsp;")
```



```
> lapply(annotlist, function(x) x[5:10])
```

```
$description
```

```
$description$`31489_at`
```

```
[1] "&nbsp;"
```

```
$description$`31490_at`
```

```
[1] "Sodium channel protein type 5 subunit alpha (Sodium channel"  
[2] "protein type V subunit alpha) (Voltage-gated sodium channel"  
[3] "subunit alpha Nav1.5) (Sodium channel protein cardiac muscle"  
[4] "subunit alpha) (HH1). [Source:Uniprot/SWISSPROT;Acc:Q14524]"
```

```
$description$`31491_s_at`
```

```
[1] "Caspase-8 precursor (EC 3.4.22.61) (CASP-8) (ICE-like"  
[2] "apoptotic protease 5) (MORT1-associated CED-3 homolog) (MACH)"  
[3] "(FADD-homologous ICE/CED-3-like protease) (FADD-like ICE)"  
[4] "(FLICE) (Apoptotic cysteine protease) (Apoptotic protease"  
[5] "Mch-5) (CAP4) [Contains [Source:Uniprot/SWISSPROT;Acc:Q14790]"
```

```
$description$`31492_at`
```

```
[1] "Eukaryotic translation initiation factor 3 subunit K"  
[2] "(Eukaryotic translation initiation factor 3 subunit 12) (eIF-3"  
[3] "p25) (eIF-3 p28) (eIF3k) (Muscle-specific gene M9 protein)"  
[4] "(PLAC-24). [Source:Uniprot/SWISSPROT;Acc:Q9UBQ5]"
```

```
$description$`31493_s_at`
```

```
[1] "chorionic somatomammotropin hormone 2 isoform 3"  
[2] "[Source:RefSeq_peptide;Acc:NP_072171]"  
[3] "Chorionic somatomammotropin hormone precursor"  
[4] "(Choriomammotropin) (Lactogen)."  
[5] "[Source:Uniprot/SWISSPROT;Acc:P01243]"  
[6] "Chorionic somatomammotropin hormone-like 1 precursor"  
[7] "(Chorionic somatomammotropin-like) (Lactogen-like)."  
[8] "[Source:Uniprot/SWISSPROT;Acc:Q14406]"
```

```
$description$`31494_at`
```

```
[1] "&nbsp;"
```

```
$hgnc_symbol
```

\$hgnc_symbol\$`31489_at`

[1] " "

\$hgnc_symbol\$`31490_at`

[1] "SCN5A"

\$hgnc_symbol\$`31491_s_at`

[1] "CASP8"

\$hgnc_symbol\$`31492_at`

[1] "EIF3K"

\$hgnc_symbol\$`31493_s_at`

[1] "CSH2" "CSH1" "CSHL1"

\$hgnc_symbol\$`31494_at`

[1] " "

\$go_biological_process_id

\$go_biological_process_id\$`31489_at`

[1] " "

\$go_biological_process_id\$`31490_at`

[1] "GO:0006811" "GO:0006814" "GO:0006936" "GO:0008015" "GO:0008016"

[6] "GO:0006816" "GO:0006813"

\$go_biological_process_id\$`31491_s_at`

[1] "GO:0006508" "GO:0008624" "GO:0008633" "GO:0042981" "GO:0043123"

[6] "GO:0006915" "GO:0001525" "GO:0007507" "GO:0001841" "GO:0030225"

\$go_biological_process_id\$`31492_at`

[1] "GO:0006446"

\$go_biological_process_id\$`31493_s_at`

[1] "GO:0007165" "GO:0007565" "GO:0006508" "GO:0008150"

\$go_biological_process_id\$`31494_at`

[1] " "

```
[[4]]
[[4]]$`31489_at`
[1] "&nbsp;"

[[4]]$`31490_at`
[1] "p22.2"

[[4]]$`31491_s_at`
[1] "q33.1"

[[4]]$`31492_at`
[1] "q13.2"

[[4]]$`31493_s_at`
[1] "q23.3"

[[4]]$`31494_at`
[1] "&nbsp;"
```

Now we have the annotation data, it is time to add in the experimental data. As an example, we will only use the first ten samples. We also use the `round` function to truncate the data to a reasonable number of decimal points.

```
> dat <- round(exprs(sample.ExpressionSet)[igenes,1:10], 3)
> FC <- round(rowMeans(dat[igenes,1:5]) - rowMeans(dat[igenes,6:10]), 2)
> pval <- round(esApply(sample.ExpressionSet[igenes,1:10], 1,
+                       function(x) t.test(x[1:5], x[6:10])$p.value), 3)
> tstat <- round(esApply(sample.ExpressionSet[igenes,1:10], 1,
+                       function(x) t.test(x[1:5], x[6:10])$statistic), 2)
```

We now need to put all this into one list.

```
> othernames <- vector("list", length = 8)
> othernames[1:4] <- annotlist
> othernames[5:8] <- list( tstat, pval, FC, dat)
```

5 Build the Table

Once we have all our data in `lists`, it is simple to build the HTML table.

```

> table.head <- c("Affy ID", "Entrez Gene", "SwissProt", "RefSeq",
+               "Name", "Symbol", "GO Term", "Band",
+               "t-statistic", "p-value", "Fold change",
+               sampleNames(sample.ExpressionSet)[1:10])
> repository <- list("affy", "en", "sp", "gb")
> htmlpage(genelist, "Annotated genes.html", "Annotated genes", othernames, table.head,
+          repository = repository)

```

The resulting HTML table should be in `R_HOME/library/annotate/doc`. If not, you can reproduce it using the `vExplorer` function in the *tkWidgets* package, which will allow you to step through the code in this vignette and examine all the objects that are made. Alternatively, one could use `getwd` to change the working directory to `R_HOME/library/annotate/doc`, then use `Stangle` on the vignette and then source the resulting R code (e.g., `Stangle("prettyOutput.Rnw")` followed by `source("prettyOutput.R")`).

6 Session Information

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

```

R version 2.8.0 (2008-10-20)
i386-pc-mingw32

```

```

locale:

```

```

LC_COLLATE=English_United States.1252;LC_CTYPE=English_United States.1252;LC_MONETARY

```

```

attached base packages:

```

```

[1] tools      stats      graphics  grDevices  utils      datasets
[7] methods    base

```

```

other attached packages:

```

```

[1] biomaRt_1.16.0      annotate_1.20.0      xtable_1.5-4
[4] AnnotationDbi_1.4.0 Biobase_2.2.0

```

```

loaded via a namespace (and not attached):

```

```

[1] DBI_0.2-4      RCurl_0.91-0  RSQLite_0.7-0 XML_1.98-1

```