

# HowTo: get pretty HTML output for my gene list

James W. MacDonald

July 25, 2006

## 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.exprSet)
> igenes <- geneNames(sample.exprSet)[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")
```

```
Checking attributes and filters ... ok
```

## 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] "adf_swissprot"          "uniprot_swissprot"
[3] "uniprot_swissprot_accession"

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

[1] "adf_entrezgene" "entrezgene"
```

Something like `adf_entrezgene` is a database table name, and it isn't all that informative, so we can try to get more information by extracting the description for these database table names. We can also get the location of the attribute in the attributes list so we don't have to write them all out.

```
> cbind(sapply(mget(attrbuts[grep("swiss", attrbuts)]),
+   mart@attributes), function(x) x$description), grep("swiss",
+   attrbuts))
```

|                             | [,1]                           | [,2]  |
|-----------------------------|--------------------------------|-------|
| adf_swissprot               | "swissprot"                    | "7"   |
| uniprot_swissprot           | "UniProt/Swiss-Prot ID"        | "348" |
| uniprot_swissprot_accession | "UniProt/Swiss-Prot Accession" | "349" |

```
> cbind(sapply(mget(attrbuts[grep("entrez", attrbuts)],
+   mart@attributes), function(x) x$description), grep("entrez",
+   attrbuts))
```

```
      [,1]      [,2]
adf_entrezgene "entrezgene" "2"
entrezgene     "EntrezGene ID" "155"
```

So the attributes we want are uniprot\_swissprot\_accession and entrez-gene.

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] "affy_hc_g110"      "affy_hg_focus"
[3] "affy_hg_u133_plus_2" "affy_hg_u133a"
[5] "affy_hg_u133a_2"   "affy_hg_u133b"
[7] "affy_hg_u95av2"    "affy_hg_u95b"
[9] "affy_hg_u95c"      "affy_hg_u95d"
[11] "affy_hg_u95e"      "affy_hugenefl"
[13] "affy_u133_x3p"     "with_affy_hc_g110"
[15] "with_affy_hg_focus" "with_affy_hg_u133_plus_2"
[17] "with_affy_hg_u133a" "with_affy_hg_u133a_2"
[19] "with_affy_hg_u133b" "with_affy_hg_u95av2"
[21] "with_affy_hg_u95b"  "with_affy_hg_u95c"
[23] "with_affy_hg_u95d"  "with_affy_hg_u95e"
[25] "with_affy_hugenefl" "with_affy_u133_x3p"
```

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_accession", "refseq_dna"), filter = "affy_hg_u95av2",
+   values = igenes, mart = mart, output = "list", na.value = "&nbsp;")
> genelist[[1]] <- igenes
> 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] 649365 6331

$entrezgene$`31491_s_at`
[1] 841

$entrezgene$`31492_at`
[1] 27335

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

$entrezgene$`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"

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

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

$refseq_dna$`31490_at`
[1] "NM_000335" "NM_198056" ""

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

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

$refseq_dna$`31493_s_at`
[1] "NM_022644" "NM_022640" "NM_022557" "NM_001317" "NM_020991"
[6] "NM_022645" "NM_022641" ""

$refseq_dna$`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 = "&nbsp;"`, 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.

```
> attrbuts[grepl("description", attrbuts)]

[1] "description"
[2] "family_description"
[3] "go_description"
[4] "hsapiens_gene_ensembl_structure.description"
[5] "interpro_description"
[6] "interpro_short_description"

> attrbuts[grepl("symbol", attrbuts)]

[1] "hgnc_symbol"

> attrbuts[grepl("go", attrbuts)]

[1] "adf_go"          "go"              "go_description"

> attrbuts[grepl("^chrom", attrbuts)]

[1] "chromosome_location" "chromosome_name"
```

We want "description", "hgnc\_symbol", "go\_description" and "band"

```
> annotlist <- getBM(attributes = c("description", "hgnc_symbol",
+   "go_description", "band"), 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 alpha subunit (Sodium channel"
[2] "protein type V alpha subunit) (Voltage-gated sodium channel"
[3] "alpha subunit Nav1.5) (Sodium channel protein, cardiac muscle"
[4] "alpha-subunit) (HH1). [Source:Uniprot/SWISSPROT;Acc:Q14524]"

$description$`31491_s_at`
[1] "Caspase-8 precursor (EC 3.4.22.-) (CASP-8) (ICE-like apoptotic"
[2] "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 12 (eIF-3"
[2] "p25) (eIF-3 p28) (eIF3k) (Muscle-specific gene M9 protein)"
[3] "(ARG134) (PLAC- 24). [Source:Uniprot/SWISSPROT;Acc:Q9UBQ5]"

$description$`31493_s_at`
[1] "Chorionic somatomammotropin hormone precursor"
[2] "(Choriomammotropin) (Lactogen). "
[3] "[Source:Uniprot/SWISSPROT;Acc:P01243]"

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

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

$hgnc_symbol$`31490_at`
[1] "SCN5A"

```



\$hgnc\_symbol\$`31491\_s\_at`  
[1] "CASP8"

\$hgnc\_symbol\$`31492\_at`  
[1] "EIF3S12"

\$hgnc\_symbol\$`31493\_s\_at`  
[1] "CSH1" "CSH2"

\$hgnc\_symbol\$`31494\_at`  
[1] "&nbsp;";

\$go\_description  
\$go\_description\$`31489\_at`  
[1] "&nbsp;";

\$go\_description\$`31490\_at`  
[1] "voltage-gated sodium channel activity"  
[2] "sodium ion binding"  
[3] "cation transport"  
[4] "sodium ion transport"  
[5] "muscle contraction"  
[6] "regulation of heart contraction"  
[7] "voltage-gated sodium channel complex"  
[8] "membrane fraction"  
[9] "membrane"  
[10] "cation channel activity"  
[11] "integral to membrane"

\$go\_description\$`31491\_s\_at`  
[1] "signal transducer activity"  
[2] "cysteine-type peptidase activity"  
[3] "caspase activity"  
[4] "identical protein binding"  
[5] "proteolysis"  
[6] "apoptotic program"  
[7] "regulation of apoptosis"  
[8] "positive regulation of I-kappaB kinase/NF-kappaB cascade"  
[9] "mitochondrion"

```

[10] "cytoskeleton"
[11] "cytoplasm"
[12] "nucleus"
[13] "protein binding"
[14] "apoptosis"
[15] "Noc1p-Noc2p complex"
[16] "macrophage differentiation"

$go_description$`31492_at`
[1] "translation initiation factor activity"
[2] "protein biosynthesis"
[3] "translational initiation"
[4] "nucleus"

$go_description$`31493_s_at`
[1] "hormone activity"
[2] "extracellular region"
[3] "growth hormone receptor binding"
[4] "signal transduction"
[5] "cell-cell signaling"
[6] "pregnancy"
[7] "extracellular space"

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

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

```
Version 2.3.1 (2006-06-01)
i386-pc-mingw32
```

```
attached base packages:
```

```
[1] "tools"      "methods"    "stats"      "graphics"   "grDevices"
[6] "utils"      "datasets"   "base"
```

```
other attached packages:
```

```
biomaRt      RCurl        XML annotate  Biobase
"1.6.3"      "0.6-2"      "0.99-7"     "1.10.0"     "1.10.1"
```