Using the GEOquery package

Sean Davis[‡*]

September 2, 2008

[‡]Genetics Branch
National Cancer Institute
National Institutes of Health

Contents

[*]sdavis2@mail.nih.gov

1 Overview of GEO

The NCBI Gene Expression Omnibus (GEO) serves as a public repository for a wide range of high-throughput experimental data. These data include single and dual channel microarray-based experiments measuring mRNA, genomic DNA, and protein abundance, as well as non-array techniques such as serial analysis of gene expression (SAGE), mass spectrometry proteomic data, and high-throughput sequencing data.

At the most basic level of organization of GEO, there are four basic entity types. The first three (Sample, Platform, and Series) are supplied by users; the fourth, the dataset, is compiled and curated by GEO staff from the user-submitted data.[1]

1.1 Platforms

A Platform record describes the list of elements on the array (e.g., cDNAs, oligonucleotide probesets, ORFs, antibodies) or the list of elements that may be detected and quantified in that experiment (e.g., SAGE tags, peptides). Each Platform record is assigned a unique and stable GEO accession number (GPLxxx). A Platform may reference many Samples that have been submitted by multiple submitters.

1.2 Samples

A Sample record describes the conditions under which an individual Sample was handled, the manipulations it underwent, and the abundance measurement of each element derived from it. Each Sample record is assigned a unique and stable GEO accession number (GSMxxx). A Sample entity must reference only one Platform and may be included in multiple Series.

1.3 Series

A Series record defines a set of related Samples considered to be part of a group, how the Samples are related, and if and how they are ordered. A Series provides a focal point and description of the experiment as a whole. Series records may also contain tables describing extracted data, summary conclusions, or analyses. Each Series record is assigned a unique and stable GEO accession number (GSExxx). Series records are available in a couple of formats which are handled by GEOquery independently. The smaller and new GSEMatrix files are quite fast to parse; a simple flag is used by GEOquery to choose to use GSEMatrix files (see below).

1.4 Datasets

GEO DataSets (GDSxxx) are curated sets of GEO Sample data. A GDS record represents a collection of biologically and statistically comparable GEO Samples and forms the basis

[1] See `http://www.ncbi.nih.gov/geo` for more information

of GEO's suite of data display and analysis tools. Samples within a GDS refer to the same Platform, that is, they share a common set of probe elements. Value measurements for each Sample within a GDS are assumed to be calculated in an equivalent manner, that is, considerations such as background processing and normalization are consistent across the dataset. Information reflecting experimental design is provided through GDS subsets.

2 Getting Started using GEOquery

Getting data from GEO is really quite easy. There is only one command that is needed, `getGEO`. This one function interprets its input to determine how to get the data from GEO and then parse the data into useful R data structures. Usage is quite simple:

```
> library(GEOquery)
```

This loads the GEOquery library.

```
> gds <- getGEO("GDS12")

File stored at:
/tmp/Rtmpebe3Oh/GDS12.soft
```

Now, `gds` contains the R data structure (of class *GDS*) that represents the GDS1 entry from GEO. You'll note that the filename used to store the download was output to the screen (but not saved anywhere) for later use to a call to getGEO(filename=...).

We can do the same with any other GEO accession, such as GSM3, a GEO sample.

```
> gsm <- getGEO("GSM3")

File stored at:
/tmp/Rtmpebe3Oh/GSM3.soft
```

3 GEOquery Data Structures

The GEOquery data structures really come in two forms. The first, comprising *GDS*, *GPL*, and *GSM* all behave similarly and accessors have similar effects on each. The fourth GEOquery data structure, *GSE* is a composite data type made up of a combination of *GSM* and *GPL* objects. I will explain the first three together first.

3.1 The GDS, GSM, and GPL classes

Each of these classes is comprised of a metadata header (taken nearly verbatim from the SOFT format header) and a GEODataTable. The GEODataTable has two simple parts, a Columns part which describes the column headers on the Table part. There is also a *show* method for each class. For example, using the gsm from above:

```
> Meta(gsm)

$channel_count
[1] "1"

$contact_address
[1] "6 Center Drive"

$contact_city
[1] "Bethesda"

$contact_country
[1] "USA"

$contact_department
[1] "LCDB"

$contact_email
[1] "oliver@helix.nih.gov"

$contact_fax
[1] "301-496-5239"

$contact_institute
[1] "NIDDK, NIH"

$contact_name
[1] "Brian,,Oliver"

$contact_phone
[1] "301-496-5495"

$contact_state
[1] "MD"

$contact_web_link
[1] "http://www.niddk.nih.gov/intram/people/boliver.htm"
```

```
$`contact_zip/postal_code`
[1] "20892"

$data_row_count
[1] "3456"

$description
[1] "Testis dissected from adult (12-24 hours post-eclosion) Drosophila melanogaster of
[2] "Keywords = gonad, male, sex"

$geo_accession
[1] "GSM3"

$last_update_date
[1] "May 27 2005"

$molecule_ch1
[1] "total RNA"

$organism_ch1
[1] "Drosophila melanogaster"

$platform_id
[1] "GPL5"

$series_id
[1] "GSE462"

$source_name_ch1
[1] "y w[67c1]/Y testis"

$status
[1] "Public on Oct 18 2000"

$submission_date
[1] "Oct 18 2000"

$supplementary_file
[1] "NONE"

$title
```

```
[1] "testis a"

$type
[1] "RNA"
```

```
> Table(gsm)[1:5, ]
```

```
  ID_REF SIGNAL_RAW BKD_FORM NORM_FORM     BKD_RAW  NORM_VALUE CONST
1      1  138392.65       no        no 101113.7775 395070.1312 39542
2      2  100973.49       no        no 101113.7775 395070.1312 39542
3      3  118994.03       no        no 101113.7775 395070.1312 39542
4      4  108126.05      yes        no 101113.7775 395070.1312 39542
5      5  293362.11       no        no 101113.7775 395070.1312 39542
        VALUE
1 76820.87249
2  39401.7125
3 57422.25249
4  46554.2725
5 231790.3324
```

```
> Columns(gsm)
```

```
      Column                                                Description
1     ID_REF
2 SIGNAL_RAW                                                 raw signal
3   BKD_FORM
4  NORM_FORM
5    BKD_RAW raw background as taken in four quarters of microarray
6 NORM_VALUE                                      normalization value
7      CONST                                           constant value
8      VALUE
```

The *GPL* behaves exactly as the *GSM* class. However, the GDS has a bit more information associated with the *Columns* method:

```
> Columns(gds)
```

```
  sample cell.line
1 GSM834      HS-5
2 GSM835      HS-5
3 GSM836      HS-5
4 GSM837      HS-5
5 GSM838    HS-27a
6 GSM839    HS-27a
```

```
7 GSM840    HS-27a
8 GSM841    HS-27a
                                                                                  description
1         Value for GSM834: HS-5_10; src: HS-5; src: Universal human reference RNA
2     Value for GSM835: HS-5_06; src: HS-5 RNA; src: Universal human reference RNA
3     Value for GSM836: HS-5_05; src: HS-5 RNA; src: Universal human reference RNA
4     Value for GSM837: HS-5_04; src: HS-5 RNA; src: Universal human reference RNA
5 Value for GSM838: HS-27a_31; src: HS-27a RNA; src: Universal human reference RNA
6 Value for GSM839: HS-27a_27; src: HS-27a RNA; src: Universal human reference RNA
7 Value for GSM840: HS-27a_09; src: HS-27a RNA; src: Universal human reference RNA
8 Value for GSM841: HS-27a_05; src: HS-27a RNA; src: Universal human reference RNA
```

3.2 The GSE class

The *GSE* is the most confusing of the GEO entities. A GSE entry can represent an arbitrary number of samples run on an arbitrary number of platforms. The *GSE* has a metadata section, just like the other classes. However, it doesn't have a GEODataTable. Instead, it contains two lists, accessible using *GPLList* and GSMList, that are each lists of *GPL* and *GSM* objects. To show an example:

```
> gse <- getGEO("GSE462", GSEMatrix = FALSE)

File stored at:
/tmp/Rtmpebe3Oh/GSE462.soft
Parsing....
^PLATFORM = GPL5
^SAMPLE = GSM3
^SAMPLE = GSM4
^SAMPLE = GSM5
^SAMPLE = GSM6
^SAMPLE = GSM7
^SAMPLE = GSM8
^SAMPLE = GSM9

> Meta(gse)

$contact_address
[1] "6 Center Drive"

$contact_city
[1] "Bethesda"

$contact_country
[1] "USA"
```

```
$contact_department
[1] "LCDB"

$contact_email
[1] "oliver@helix.nih.gov"

$contact_fax
[1] "301-496-5239"

$contact_institute
[1] "NIDDK, NIH"

$contact_name
[1] "Brian,,Oliver"

$contact_phone
[1] "301-496-5495"

$contact_state
[1] "MD"

$contact_web_link
[1] "http://www.niddk.nih.gov/intram/people/boliver.htm"

$`contact_zip/postal_code`
[1] "20892"

$contributor
[1] "Justen,,Andrews"   "Gerard,G,Bouffard" "Chris,,Cheadle"
[4] "Jining,,LÃij"       "Kevin,G,Becker"    "Brian,,Oliver"

$geo_accession
[1] "GSE462"

$last_update_date
[1] "Oct 28 2005"

$platform_id
[1] "GPL5"

$pubmed_id
```

```
[1] "11116097"

$sample_id
[1] "GSM10" "GSM3"  "GSM4"  "GSM5"  "GSM6"  "GSM7"  "GSM8"  "GSM9"

$status
[1] "Public on Jul 16 2003"

$submission_date
[1] "Jun 25 2003"

$summary
[1] "Identification and annotation of all the genes in the sequenced Drosophila genome i

$title
[1] "Analysis of transcription in the Drosophila melanogaster testis"

$type
[1] "other"
```

```
> names(GSMList(gse))
```

```
[1] "GSM10" "GSM3"  "GSM4"  "GSM5"  "GSM6"  "GSM7"  "GSM8"  "GSM9"
```

```
> GSMList(gse)[[1]]
```

```
An object of class "GSM"
channel_count
[1] "1"
contact_address
[1] "6 Center Drive"
contact_city
[1] "Bethesda"
contact_country
[1] "USA"
contact_department
[1] "LCDB"
contact_email
[1] "oliver@helix.nih.gov"
contact_fax
[1] "301-496-5239"
contact_institute
[1] "NIDDK, NIH"
```

```
contact_name
[1] "Brian,,Oliver"
contact_phone
[1] "301-496-5495"
contact_state
[1] "MD"
contact_web_link
[1] "http://www.niddk.nih.gov/intram/people/boliver.htm"
contact_zip/postal_code
[1] "20892"
data_row_count
[1] "3456"
description
[1] "Whole adult male minus (12-24 hours post-eclosion) Drosophila melanogaster of the g
geo_accession
[1] "GSM10"
last_update_date
[1] "Mar 09 2006"
molecule_ch1
[1] "total RNA"
organism_ch1
[1] "Drosophila melanogaster"
platform_id
[1] "GPL5"
series_id
[1] "GSE462"
source_name_ch1
[1] "y w[67c1] female"
status
[1] "Public on Oct 18 2000"
submission_date
[1] "Oct 18 2000"
title
[1] "female b"
type
[1] "RNA"
An object of class "GEODataTable"
****** Column Descriptions ******
      Column         Description
1     ID_REF
2 SIGNAL_RAW          raw signal
3   BKD_FORM
```

```
4  NORM_FORM
5    BKD_RAW      raw background
6 NORM_VALUE normalization value
7      CONST      constant value
8      VALUE
****** Data Table ******
  ID_REF SIGNAL_RAW BKD_FORM NORM_FORM  BKD_RAW NORM_VALUE CONST    VALUE
1      1    4486.49        0         0 3379.579   23337.54 39542 55845.45
2      2    3482.51        0         0 3379.579   23337.54 39542 41058.05
3      3    3812.39        0         0 3379.579   23337.54 39542 45916.78
4      4    3257.56        1         0 3379.579   23337.54 39542 37744.81
5      5    5436.91        0         0 3379.579   23337.54 39542 69843.97
3450 more rows ...
```

```
> names(GPLList(gse))
```

```
[1] "GPL5"
```

See below for an additional, preferred method of obtaining GSE information.

4 Converting to BioConductor ExpressionSets and limma MALists

GEO datasets are (unlike some of the other GEO entities), quite similar to the *limma* data structure *MAList* and to the *Biobase* data structure *ExpressionSet*. Therefore, there are two functions, `GDS2MA` and `GDS2eSet` that accomplish that task.

4.1 Getting GSE Series Matrix files as an ExpressionSet

GEO Series are collections of related experiments. In addition to being available as SOFT format files, which are quite large, NCBI GEO has prepared a simpler format file based on tab-delimited text. The `getGEO` function can handle this format and will parse very large GSEs quite quickly. The data structure returned from this parsing is a list of ExpressionSets. As an example, we download and parse GSE2553.

```
> gse2553 <- getGEO("GSE2553", GSEMatrix = TRUE)
```

```
Found 1 file(s)
GSE2553_series_matrix.txt.gz
File stored at:
/tmp/Rtmpebe3Oh/GPL1977.soft
```

```
> show(gse2553)
```

```
$GSE2553_series_matrix.txt.gz
ExpressionSet (storageMode: lockedEnvironment)
assayData: 12600 features, 181 samples
  element names: exprs
phenoData
  sampleNames: GSM48681, GSM48682, ..., GSM48861  (181 total)
  varLabels and varMetadata description:
    title: NA
    geo_accession: NA
    ...: ...
    data_row_count: NA
    (27 total)
featureData
  featureNames: 1, 2, ..., 12600  (12600 total)
  fvarLabels and fvarMetadata description:
    ID: NA
    PenAt: NA
    ...: ...
    Chimeric_Cluster_IDs: NA
    (13 total)
  additional fvarMetadata: Column, Description
experimentData: use 'experimentData(object)'
Annotation: GPL1977
```

```
> show(pData(phenoData(gse2553[[1]]))[1:5, c(1, 6, 8)])
```

```
                                                                 title type
GSM48681                      Patient sample ST18, Dermatofibrosarcoma  RNA
GSM48682                           Patient sample ST410, Ewing Sarcoma  RNA
GSM48683                            Patient sample ST130, Sarcoma, NOS  RNA
GSM48684 Patient sample ST293, Malignant Peripheral Nerve Sheath Tumor  RNA
GSM48685                              Patient sample ST367, Liposarcoma  RNA
                                source_name_ch1
GSM48681                      Dermatofibrosarcoma
GSM48682                            Ewing Sarcoma
GSM48683                             Sarcoma, NOS
GSM48684 Malignant Peripheral Nerve Sheath Tumor
GSM48685                              Liposarcoma
```

4.2 Converting GDS to an ExpressionSet

Taking our gds object from above, we can simply do:

```
> eset <- GDS2eSet(gds, do.log2 = TRUE)
```

```
File stored at:
/tmp/Rtmpebe3Oh/GPL44.annot
```

Now, `eset` is an *ExpressionSet* that contains the same information as in the GEO dataset, including the sample information, which we can see here:

```
> eset

ExpressionSet (storageMode: lockedEnvironment)
assayData: 17588 features, 8 samples
  element names: exprs
phenoData
  sampleNames: GSM834, GSM835, ..., GSM841  (8 total)
  varLabels and varMetadata description:
    sample: NA
    cell.line: NA
    description: NA
featureData
  featureNames: 1, 2, ..., 17588  (17588 total)
  fvarLabels and fvarMetadata description:
    ID: ID from Platform data table
    Gene.title: Entrez Gene name
    ...: ...
    GO.Component.1: Gene Ontology Component identifier
    (21 total)
  additional fvarMetadata: Column
experimentData: use 'experimentData(object)'
  pubMedIds: 12184274
Annotation:

> pData(eset)

       sample cell.line
GSM834 GSM834      HS-5
GSM835 GSM835      HS-5
GSM836 GSM836      HS-5
GSM837 GSM837      HS-5
GSM838 GSM838    HS-27a
GSM839 GSM839    HS-27a
GSM840 GSM840    HS-27a
GSM841 GSM841    HS-27a
                                                                        description
GSM834         Value for GSM834: HS-5_10; src: HS-5; src: Universal human reference RNA
GSM835     Value for GSM835: HS-5_06; src: HS-5 RNA; src: Universal human reference RNA
```

```
GSM836     Value for GSM836: HS-5_05; src: HS-5 RNA; src: Universal human reference RNA
GSM837     Value for GSM837: HS-5_04; src: HS-5 RNA; src: Universal human reference RNA
GSM838 Value for GSM838: HS-27a_31; src: HS-27a RNA; src: Universal human reference RNA
GSM839 Value for GSM839: HS-27a_27; src: HS-27a RNA; src: Universal human reference RNA
GSM840 Value for GSM840: HS-27a_09; src: HS-27a RNA; src: Universal human reference RNA
GSM841 Value for GSM841: HS-27a_05; src: HS-27a RNA; src: Universal human reference RNA
```

4.3 Converting GDS to an MAList

No annotation information (called platform information by GEO) was retrieved from because *ExpressionSet* does not contain slots for gene information, typically. However, it is easy to obtain this information. First, we need to know what platform this GDS used. Then, another call to `getGEO` will get us what we need.

```
> Meta(gds)$platform

[1] "GPL44"

> gpl <- getGEO("GPL5")

File stored at:
/tmp/Rtmpebe3Oh/GPL5.soft
```

So, `gpl` now contains the information for GPL5 from GEO. Unlike *ExpressionSet*, the limma *MAList* does store gene annotation information, so we can use our newly created `gpl` of class *GPL* in a call to `GDS2MA` like so:

```
> MA <- GDS2MA(gds, GPL = gpl)
> MA

An object of class "MAList"
$M
      GSM834 GSM835 GSM836 GSM837 GSM838 GSM839 GSM840 GSM841
[1,] -0.844 -0.805 -0.751 -0.887 -1.242 -1.145 -1.004 -0.686
[2,]  1.203  0.493  0.665  0.825  0.995  0.988  0.625  0.897
[3,]  0.299  0.536  0.183  0.433 -0.304 -0.090  0.181 -0.104
[4,] -0.886 -0.437 -0.471 -0.861 -0.481 -0.204 -0.457 -0.022
[5,] -0.934  0.082 -0.350 -0.634  0.363  0.511 -0.006  0.257
17583 more rows ...

$A
NULL

$targets
```

```
  sample cell.line
1 GSM834      HS-5
2 GSM835      HS-5
3 GSM836      HS-5
4 GSM837      HS-5
5 GSM838    HS-27a
6 GSM839    HS-27a
7 GSM840    HS-27a
8 GSM841    HS-27a
                                                                        description
1         Value for GSM834: HS-5_10; src: HS-5; src: Universal human reference RNA
2     Value for GSM835: HS-5_06; src: HS-5 RNA; src: Universal human reference RNA
3     Value for GSM836: HS-5_05; src: HS-5 RNA; src: Universal human reference RNA
4     Value for GSM837: HS-5_04; src: HS-5 RNA; src: Universal human reference RNA
5 Value for GSM838: HS-27a_31; src: HS-27a RNA; src: Universal human reference RNA
6 Value for GSM839: HS-27a_27; src: HS-27a RNA; src: Universal human reference RNA
7 Value for GSM840: HS-27a_09; src: HS-27a RNA; src: Universal human reference RNA
8 Value for GSM841: HS-27a_05; src: HS-27a RNA; src: Universal human reference RNA

$genes
  ID   GB_ACC BSCC_ID    CLONE_ID SUB.ARRAY DUPLICATE ROW COLUMN PCR_QC SPOT_ID
1  1 AI944549 bs03g07 FBgn0033989         1         a   1      1 passed
2  2 AI944695 bs04c11 FBgn0032821         1         a   1      2 passed
3  3 AI944741 bs04h01 FBgn0034374         1         a   1      3 passed
4  4 AI944801 bs05f04 FBgn0039421         1         a   1      4 failed
5  5 AI945043 bs08c11 FBgn0045370         1         a   1      5 passed

1                                                                                g
2
3
4 gi|4505995|ref|NP_002697.1|PPPM1B| protein phosphatase 1B (formerly 2C), magnesium-dep
5
  E_VAL SPOT_QC
1 2e-08   44364
2  <NA>   16957
3  <NA>   17896
4 1e-25   16363
5  <NA>   83502
17583 more rows ...

$notes
[[1]]
```

```
[1] "able_begin"

$channel_count
[1] "2"

$description
[1] "Examination of two functionally distinct human bone marrow stromal cell lines, HS-2

$feature_count
[1] "17588"

$order
[1] "none"

$platform
[1] "GPL44"

$platform_organism
[1] "Homo sapiens"

$platform_technology_type
[1] "spotted DNA/cDNA"

$pubmed_id
[1] "12184274"

$reference_series
[1] "GSE463"

$sample_count
[1] "8"

$sample_organism
[1] "Homo sapiens"

$sample_type
[1] "RNA"

$title
[1] "Bone marrow stromal cell lines"

$type
```

```
[1] "gene expression array-based"

$update_date
[1] "Jun 25 2003"

$value_type
[1] "log ratio"
```

Now, `MA` is of class *MAList* and contains not only the data, but the sample information and gene information associated with GDS1.

4.4 Converting GSE to an ExpressionSet

First, make sure that using the method described above in the section "Getting GSE Series Matrix files as an ExpressionSet" for using GSE Series Matrix files is not sufficient for the task, as it is much faster and simpler. If it is not (i.e., other columns from each GSM are needed), then this method will be needed.

Converting a *GSE* object to an *ExpressionSet* object currently takes a bit of R data manipulation due to the varied data that can be stored in a *GSE* and the underlying *GSM* and *GPL* objects. However, using a simple example will hopefully be illustrative of the technique.

First, we need to make sure that all of the *GSMs* are from the same platform:

```
> gsmplatforms <- lapply(GSMList(gse), function(x) {
+     Meta(x)$platform
+ })
> gsmplatforms

$GSM10
[1] "GPL5"

$GSM3
[1] "GPL5"

$GSM4
[1] "GPL5"

$GSM5
[1] "GPL5"

$GSM6
[1] "GPL5"
```

```
$GSM7
[1] "GPL5"

$GSM8
[1] "GPL5"

$GSM9
[1] "GPL5"
```

Indeed, they all used GPL5 as their platform (which we could have determined by looking at the GPLList for `gse`, which shows only one GPL for this particular GSE.). So, now we would like to know what column represents the data that we would like to extract. Looking at the first few rows of the Table of a single GSM will likely give us an idea (and by the way, GEO uses a convention that the column that contains the single "measurement" for each array is called the "VALUE" column, which we could use if we don't know what other column is most relevant).

```
> Table(GSMList(gse)[[1]])[1:5, ]

  ID_REF SIGNAL_RAW BKD_FORM NORM_FORM  BKD_RAW NORM_VALUE CONST    VALUE
1      1    4486.49        0         0 3379.579   23337.54 39542 55845.45
2      2    3482.51        0         0 3379.579   23337.54 39542 41058.05
3      3    3812.39        0         0 3379.579   23337.54 39542 45916.78
4      4    3257.56        1         0 3379.579   23337.54 39542 37744.81
5      5    5436.91        0         0 3379.579   23337.54 39542 69843.97

> Columns(GSMList(gse)[[1]])[1:5, ]

      Column    Description
1     ID_REF
2 SIGNAL_RAW     raw signal
3   BKD_FORM
4  NORM_FORM
5    BKD_RAW raw background
```

We will indeed use the "VALUE" column. We then want to make a matrix of these values like so:

```
> probesets <- Table(GPLList(gse)[[1]])$ID
> data.matrix <- do.call("cbind", lapply(GSMList(gse), function(x) {
+     tab <- Table(x)
+     mymatch <- match(probesets, tab$ID_REF)
+     return(tab$VALUE[mymatch])
+ }))
```

```
> data.matrix <- apply(data.matrix, 2, function(x) {
+     as.numeric(as.character(x))
+ })
> data.matrix <- log2(data.matrix)
> data.matrix[1:5, ]
```

```
        GSM10     GSM3     GSM4     GSM5     GSM6     GSM7     GSM8     GSM9
[1,] 15.76915 16.22921 16.13000 15.65034 17.09214 15.45853 16.09474 15.23515
[2,] 15.32538 15.26597      NaN 15.20406 16.47596 14.85776 15.14885 14.89007
[3,] 15.48673 15.80932 14.16259 15.18048 16.21235 15.06094 15.38242 14.96986
[4,] 15.20399 15.50663 13.41582 15.05939 16.18593 14.79861 14.80460 15.01923
[5,] 16.09185 17.82246 18.38270 16.24570 16.60964 15.90011 16.00962 15.88859
```

Note that we do a "match" to make sure that the values and the platform information are in the same order. Finally, to make the *ExpressionSet* object:

```
> require(Biobase)
> rownames(data.matrix) <- probesets
> colnames(data.matrix) <- names(GSMList(gse))
> pdata <- data.frame(samples = names(GSMList(gse)))
> rownames(pdata) <- names(GSMList(gse))
> pheno <- as(pdata, "AnnotatedDataFrame")
> eset2 <- new("ExpressionSet", exprs = data.matrix, phenoData = pheno)
> eset2
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 3455 features, 8 samples
  element names: exprs
phenoData
  sampleNames: GSM10, GSM3, ..., GSM9  (8 total)
  varLabels and varMetadata description:
    samples: NA
featureData
  featureNames: 1, 2, ..., 3455  (3455 total)
  fvarLabels and fvarMetadata description: none
experimentData: use 'experimentData(object)'
Annotation:
```

So, using a combination of `lapply` on the GSMList, one can extract as many columns of interest as necessary to build the data structure of choice. Because the GSM data from the GEO website are fully downloaded and included in the *GSE* object, one can extract foreground and background as well as quality for two-channel arrays, for example. Getting array annotation is also a bit more complicated, but by replacing "platform" in the lapply call to get platform information for each array, one can get other information associated with each array. Future work with this package will likely focus on better tools for manipulating *GSE* data.

5 Accessing Raw Data from GEO

NCBI GEO accepts (but has not always required) raw data such as .CEL files, .CDF files, images, etc. Sometimes, it is useful to get quick access to such data. A single function, `getGEOSuppFiles`, can take as an argument a GEO accession and will download all the raw data associate with that accession. By default, the function will create a directory in the current working directory to store the raw data for the chosen GEO accession. Combining a simple `sapply` statement or other loop structure with `getGEOSuppFiles` makes for a very simple way to get gobs of raw data quickly and easily without needing to know the specifics of GEO raw data URLs.

6 Conclusion

The GEOquery package provides a bridge to the vast array resources contained in the NCBI GEO repositories. By maintaining the full richness of the GEO data rather than focusing on getting only the "numbers", it is possible to integrate GEO data into current Bioconductor data structures and to perform analyses on that data quite quickly and easily. These tools will hopefully open GEO data more fully to the array community at large.

7 sessionInfo

- R version 2.7.2 (2008-08-25), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US;LC_NUMERIC=C;LC_TIME=en_US;LC_COLLATE=en_US;LC_MONETARY=C;LC_M`
- Base packages: base, datasets, graphics, grDevices, methods, stats, tools, utils
- Other packages: Biobase 2.0.1, GEOquery 2.4.5, limma 2.14.6, RCurl 0.9-4