Using the GEOquery Package

Sean Davis

September 21, 2014

Contents

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. See the GEO home page for more information.

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.

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.

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).

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 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.

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. This loads the GEOquery library.

```r
library(GEOquery)
```

Now, we are free to access any GEO accession. *Note that in the following, I use a file packaged with the GEOquery package. In general, you will use only the GEO accession, as noted in the code comments.*

```r
# If you have network access, the more typical way to do this
# would be to use this:
# gds <- getGEO("GDS507")
gds <- getGEO(filename=system.file("extdata/GDS507.soft.gz",package="GEOquery"))
```

Now, `gds` contains the R data structure (of class `GDS`) that represents the GDS507 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 `GSM11805`, a GEO sample.

```r
# If you have network access, the more typical way to do this
# would be to use this:
# gds <- getGEO("GSM11805")
gsm <- getGEO(filename=system.file("extdata/GSM11805.txt.gz",package="GEOquery"))
```

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.

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:

```r
# Look at gsm metadata:
head(Meta(gsm))
```

```
## $channel_count
## [1] "1"
##
## $comment
## [1] "Raw data provided as supplementary file"
##
## $contact_address
## [1] "715 Albany Street, E613B"
##
## $contact_city
## [1] "Boston"
##
## $contact_country
## [1] "USA"
##
## $contact_department
## [1] "Genetics and Genomics"
```

```
# Look at data associated with the GSM:
# but restrict to only first 5 rows, for brevity
Table(gsm)[1:5,]
```

```
##            ID_REF  VALUE ABS_CALL
## 1 AFFX-BioB-5_at  953.9        P
## 2 AFFX-BioB-M_at 2982.8        P
## 3 AFFX-BioB-3_at 1657.9        P
## 4 AFFX-BioC-5_at 2652.7        P
## 5 AFFX-BioC-3_at 2019.5        P
```

```
# Look at Column descriptions:
Columns(gsm)
```

```
##     Column
## 1   ID_REF
## 2    VALUE
## 3 ABS_CALL
##                                                                Description
## 1
## 2                          MAS 5.0 Statistical Algorithm (mean scaled to 500)
## 3 MAS 5.0 Absent, Marginal, Present call  with Alpha1 = 0.05, Alpha2 = 0.065
```

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

```
Columns(gds)[,1:3]
```

```
##      sample disease.state individual
## 1  GSM11815           RCC        035
## 2  GSM11832           RCC        023
## 3  GSM12069           RCC        001
## 4  GSM12083           RCC        005
## 5  GSM12101           RCC        011
## 6  GSM12106           RCC        032
## 7  GSM12274           RCC          2
## 8  GSM12299           RCC          3
## 9  GSM12412           RCC          4
## 10 GSM11810        normal        035
## 11 GSM11827        normal        023
## 12 GSM12078        normal        001
## 13 GSM12099        normal        005
## 14 GSM12269        normal          1
## 15 GSM12287        normal          2
## 16 GSM12301        normal          3
## 17 GSM12448        normal          4
```

The GSE class

The `GSE` entity 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` class has a metadata section, just like the other classes. However, it doesn't have a GEODataTable. Instead, it contains two lists, accessible using the `GPLList` and `GSMList` methods, that are each lists of `GPL` and `GSM` objects. To show an example:

```
# Again, with good network access, one would do:
# gse <- getGEO("GSE781",GSEMatrix=FALSE)
gse <- getGEO(filename=system.file("extdata/GSE781_family.soft.gz",package="GEOquery"))
```

```
## Parsing....
```

```
head(Meta(gse))
```

```
## $contact_address
## [1] "715 Albany Street, E613B"
##
## $contact_city
## [1] "Boston"
##
## $contact_country
## [1] "USA"
##
## $contact_department
## [1] "Genetics and Genomics"
##
## $contact_email
## [1] "mlenburg@bu.edu"
##
## $contact_fax
## [1] "617-414-1646"
```

```
# names of all the GSM objects contained in the GSE
names(GSMList(gse))
```

```
##  [1] "GSM11805" "GSM11810" "GSM11814" "GSM11815" "GSM11823" "GSM11827"
##  [7] "GSM11830" "GSM11832" "GSM12067" "GSM12069" "GSM12075" "GSM12078"
## [13] "GSM12079" "GSM12083" "GSM12098" "GSM12099" "GSM12100" "GSM12101"
## [19] "GSM12105" "GSM12106" "GSM12268" "GSM12269" "GSM12270" "GSM12274"
## [25] "GSM12283" "GSM12287" "GSM12298" "GSM12299" "GSM12300" "GSM12301"
## [31] "GSM12399" "GSM12412" "GSM12444" "GSM12448"
```

```
# and get the first GSM object on the list
GSMList(gse)[[1]]
```

```
## An object of class "GSM"
## channel_count
## [1] "1"
## comment
## [1] "Raw data provided as supplementary file"
## contact_address
## [1] "715 Albany Street, E613B"
## contact_city
## [1] "Boston"
## contact_country
## [1] "USA"
## contact_department
## [1] "Genetics and Genomics"
## contact_email
## [1] "mlenburg@bu.edu"
## contact_fax
## [1] "617-414-1646"
## contact_institute
## [1] "Boston University School of Medicine"
## contact_name
## [1] "Marc,E.,Lenburg"
## contact_phone
## [1] "617-414-1375"
## contact_state
## [1] "MA"
## contact_web_link
## [1] "http://gg.bu.edu"
## contact_zip/postal_code
## [1] "02130"
## data_row_count
## [1] "22283"
## description
## [1] "Age = 70; Gender = Female; Right Kidney; Adjacent Tumor Type = clear cell; Adjacent Tumor Fuhrm
## [2] "Keywords = kidney"
## [3] "Keywords = renal"
## [4] "Keywords = RCC"
## [5] "Keywords = carcinoma"
## [6] "Keywords = cancer"
## [7] "Lot batch = 2004638"
## geo_accession
## [1] "GSM11805"
## last_update_date
## [1] "May 28 2005"
## molecule_ch1
## [1] "total RNA"
## organism_ch1
## [1] "Homo sapiens"
## platform_id
## [1] "GPL96"
## series_id
## [1] "GSE781"
## source_name_ch1
## [1] "Trizol isolation of total RNA from normal tissue adjacent to Renal Cell Carcinoma"
## status
## [1] "Public on Nov 25 2003"
## submission_date
## [1] "Oct 20 2003"
## supplementary_file
## [1] "ftp://ftp.ncbi.nih.gov/pub/geo/DATA/supplementary/samples/GSM11nnn/GSM11805/GSM11805.CEL.gz"
## title
## [1] "N035 Normal Human Kidney U133A"
## type
## [1] "RNA"
## An object of class "GEODataTable"
## ****** Column Descriptions ******
##     Column
## 1   ID_REF
## 2    VALUE
## 3 ABS_CALL
##                                                                   Description
## 1
## 2                         MAS 5.0 Statistical Algorithm (mean scaled to 500)
## 3 MAS 5.0 Absent, Marginal, Present call  with Alpha1 = 0.05, Alpha2 = 0.065
## ****** Data Table ******
##           ID_REF  VALUE ABS_CALL
## 1 AFFX-BioB-5_at  953.9        P
## 2 AFFX-BioB-M_at 2982.8        P
## 3 AFFX-BioB-3_at 1657.9        P
## 4 AFFX-BioC-5_at 2652.7        P
## 5 AFFX-BioC-3_at 2019.5        P
## 22278 more rows ...
```

```
# and the names of the GPLs represented
names(GPLList(gse))
```

```
## [1] "GPL96" "GPL97"
```

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

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.

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.

```
# Note that GSEMatrix=TRUE is the default
gse2553 <- getGEO('GSE2553',GSEMatrix=TRUE)
show(gse2553)
```

```
## $GSE2553_series_matrix.txt.gz
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 12600 features, 181 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: GSM48681 GSM48682 ... GSM48861 (181 total)
##   varLabels: title geo_accession ... data_row_count (30 total)
##   varMetadata: labelDescription
## featureData
##   featureNames: 1 2 ... 12600 (12600 total)
##   fvarLabels: ID PenAt ... Chimeric_Cluster_IDs (13 total)
##   fvarMetadata: Column Description labelDescription
## experimentData: use 'experimentData(object)'
## Annotation: GPL1977
```

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

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

Converting GDS to an ExpressionSet

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

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

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: 22645 features, 17 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: GSM11815 GSM11832 ... GSM12448 (17 total)
##   varLabels: sample disease.state individual description
##   varMetadata: labelDescription
## featureData
##   featureNames: 200000_s_at 200001_at ... AFFX-TrpnX-M_at (22645
##     total)
##   fvarLabels: ID Gene title ... GO:Component ID (21 total)
##   fvarMetadata: Column labelDescription
## experimentData: use 'experimentData(object)'
##   pubMedIds: 14641932
## Annotation:
```

```
pData(eset)[,1:3]
```

```
##           sample disease.state individual
## GSM11815 GSM11815           RCC        035
## GSM11832 GSM11832           RCC        023
## GSM12069 GSM12069           RCC        001
## GSM12083 GSM12083           RCC        005
## GSM12101 GSM12101           RCC        011
## GSM12106 GSM12106           RCC        032
## GSM12274 GSM12274           RCC          2
## GSM12299 GSM12299           RCC          3
## GSM12412 GSM12412           RCC          4
## GSM11810 GSM11810        normal        035
## GSM11827 GSM11827        normal        023
## GSM12078 GSM12078        normal        001
## GSM12099 GSM12099        normal        005
## GSM12269 GSM12269        normal          1
## GSM12287 GSM12287        normal          2
## GSM12301 GSM12301        normal          3
## GSM12448 GSM12448        normal          4
```

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.

```
#get the platform from the GDS metadata
Meta(gds)$platform
```

```
## [1] "GPL97"
```

```
#So use this information in a call to getGEO
gpl <- getGEO(filename=system.file("extdata/GPL97.annot.gz",package="GEOquery"))
```

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)
class(MA)
```

```
## [1] "MAList"
## attr(,"package")
## [1] "limma"
```

Now, `MA` is of class `MAList` and contains not only the data, but the sample information and gene information associated with GDS507.

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})
head(gsmplatforms)
```

```
## $GSM11805
## [1] "GPL96"
##
## $GSM11810
## [1] "GPL97"
##
## $GSM11814
## [1] "GPL96"
##
## $GSM11815
## [1] "GPL97"
##
## $GSM11823
## [1] "GPL96"
##
## $GSM11827
## [1] "GPL97"
```

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  VALUE ABS_CALL
## 1 AFFX-BioB-5_at  953.9        P
## 2 AFFX-BioB-M_at 2982.8        P
## 3 AFFX-BioB-3_at 1657.9        P
## 4 AFFX-BioC-5_at 2652.7        P
## 5 AFFX-BioC-3_at 2019.5        P
```

```
# and get the column descriptions
Columns(GSMList(gse)[[1]])[1:5,]
```

```
##        Column
## 1      ID_REF
## 2       VALUE
## 3    ABS_CALL
## NA       <NA>
## NA.1     <NA>
##                                                                   Description
## 1
## 2                            MAS 5.0 Statistical Algorithm (mean scaled to 500)
## 3    MAS 5.0 Absent, Marginal, Present call  with Alpha1 = 0.05, Alpha2 = 0.065
## NA                                                                       <NA>
## NA.1                                                                     <NA>
```

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

```
# get the probeset ordering
probesets <- Table(GPLList(gse)[[1]])$ID
# make the data matrix from the VALUE columns from each GSM
# being careful to match the order of the probesets in the platform
# with those in the GSMs
data.matrix <- do.call('cbind',lapply(GSMList(gse),function(x)
                                      {tab <- Table(x)
                                       mymatch <- match(probesets,tab$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,]
```

```
##       GSM11805 GSM11810  GSM11814 GSM11815  GSM11823 GSM11827  GSM11830
## [1,] 10.926963       NA 11.105254       NA 11.275019       NA 11.438636
## [2,]  5.749534       NA  7.908092       NA  7.093814       NA  7.514122
## [3,]  7.066089       NA  7.750205       NA  7.244126       NA  7.962896
## [4,] 12.660353       NA 12.479755       NA 12.215897       NA 11.458355
## [5,]  6.195741       NA  6.061776       NA  6.565293       NA  6.583459
##      GSM11832  GSM12067 GSM12069  GSM12075 GSM12078  GSM12079 GSM12083
## [1,]       NA 11.424376       NA 11.222795       NA 11.469845       NA
## [2,]       NA  7.901470       NA  6.407693       NA  5.165912       NA
## [3,]       NA  7.337176       NA  6.569856       NA  7.477354       NA
## [4,]       NA 11.397568       NA 12.529870       NA 12.240046       NA
## [5,]       NA  6.877744       NA  6.652486       NA  3.981853       NA
##       GSM12098 GSM12099  GSM12100 GSM12101  GSM12105 GSM12106  GSM12268
## [1,] 10.823367       NA 10.835971       NA 10.810893       NA 11.062653
## [2,]  6.556123       NA  8.207014       NA  6.816344       NA  6.563768
## [3,]  7.708739       NA  7.428779       NA  7.754888       NA  7.126188
## [4,] 12.336534       NA 11.762839       NA 11.237509       NA 12.412490
## [5,]  5.501439       NA  6.247928       NA  6.017922       NA  6.525129
##      GSM12269  GSM12270 GSM12274  GSM12283 GSM12287  GSM12298 GSM12299
## [1,]       NA 10.323055       NA 11.181028       NA 11.566387       NA
## [2,]       NA  7.353147       NA  5.770829       NA  6.912889       NA
## [3,]       NA  8.742815       NA  7.339850       NA  7.602142       NA
## [4,]       NA 11.213408       NA 12.678380       NA 12.232901       NA
## [5,]       NA  6.683696       NA  5.918863       NA  5.837943       NA
##       GSM12300 GSM12301  GSM12399 GSM12412  GSM12444 GSM12448
## [1,] 11.078151        NA 11.535178        NA 11.105450        NA
## [2,]  4.812498        NA  7.471675        NA  7.488644        NA
## [3,]  7.383704        NA  7.432959        NA  7.381110        NA
## [4,] 12.090939        NA 11.421802        NA 12.172834        NA
## [5,]  6.281698        NA  5.419539        NA  5.469235        NA
```

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:

```r
require(Biobase)
# go through the necessary steps to make a compliant ExpressionSet
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: 22283 features, 34 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: GSM11805 GSM11810 ... GSM12448 (34 total)
##   varLabels: samples
##   varMetadata: labelDescription
## featureData: 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.

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.

Use Cases

GEOquery can be quite powerful for gathering a lot of data quickly. A few examples can be useful to show how this might be done for data mining purposes.

Getting all Series Records for a Given Platform

For data mining purposes, it is sometimes useful to be able to pull all the GSE records for a given platform. GEOquery makes this very easy, but a little bit of knowledge of the GPL record is necessary to get started. The GPL record contains both the GSE and GSM accessions that reference it. Some code is useful to illustrate the point:

```r
gpl97 <- getGEO('GPL97')
Meta(gpl97)$title
```

```
## [1] "[HG-U133B] Affymetrix Human Genome U133B Array"
```

```r
head(Meta(gpl97)$series_id)
```

```
## [1] "GSE362" "GSE473" "GSE620" "GSE674" "GSE781" "GSE907"
```

```r
length(Meta(gpl97)$series_id)
```

```
## [1] 149
```

```r
head(Meta(gpl97)$sample_id)
```

```
## [1] "GSM3922" "GSM3924" "GSM3926" "GSM3928" "GSM3930" "GSM3932"
```

```r
length(Meta(gpl97)$sample_id)
```

```
## [1] 6156
```

The code above loads the GPL97 record into R. The Meta method extracts a list of header information from the GPL record. The `title` gives the human name of the platform. The `series_id` gives a vector of series ids. Note that there are 149 series associated with this platform and 6156 samples. Code like the following could be used to download all the samples or series. I show only the first 5 samples as an example:

```r
gsmids <- Meta(gpl97)$sample_id
gsmlist <- sapply(gsmids[1:5],getGEO)
names(gsmlist)
```

```
## [1] "GSM3922" "GSM3924" "GSM3926" "GSM3928" "GSM3930"
```

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.

Citing GEOquery

Please consider citing GEOquery if used in support of your own research:

```
citation("GEOquery")
```

```
##
## Please cite the following if utilizing the GEOquery software:
##
##   Davis, S. and Meltzer, P. S. GEOquery: a bridge between the Gene
##   Expression Omnibus (GEO) and BioConductor. Bioinformatics, 2007,
##   14, 1846-1847
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     author = {Sean Davis and Paul Meltzer},
##     title = {GEOquery: a bridge between the Gene Expression Omnibus (GEO) and BioConductor},
##     journal = {Bioinformatics},
##     year = {2007},
##     volume = {14},
##     pages = {1846--1847},
##   }
```

Reporting problems or bugs

If you run into problems using GEOquery, the Bioconductor Support site is a good first place to ask for help. If you are convinced that there is a bug in GEOquery (this is pretty unusual, but not unheard of), feel free to submit an issue on the GEOquery github site or file a bug report directly from R (will open a new github issue):

```
bug.report(package='GEOquery')
```

Session info

The following package and versions were used in the production of this vignette.

```
## R version 3.1.1 Patched (2014-09-25 r66681)
## Platform: x86_64-unknown-linux-gnu (64-bit)
##
## 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
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] limma_3.22.0        GEOquery_2.32.0     Biobase_2.26.0
## [4] BiocGenerics_0.12.0 knitr_1.7
##
## loaded via a namespace (and not attached):
##  [1] RCurl_1.95-4.3  XML_3.98-1.1    codetools_0.2-9 digest_0.6.4
##  [5] evaluate_0.5.5  formatR_1.0     htmltools_0.2.6 rmarkdown_0.3.3
##  [9] stringr_0.6.2   tools_3.1.1     yaml_2.1.13
```