Using the GEOquery package

Sean Davis* and Jack Zhu

July 29, 2011

Genetics Branch
National Cancer Institute
National Institutes of Health

Contents

1 Overview of GEO 2
1.1 Platforms ......................................................... 3
1.2 Samples .......................................................... 3
1.3 Series ............................................................ 3
1.4 Datasets .......................................................... 3

2 Using GEOquery to Access NCBI GEO 3
2.1 GEOquery Data Structures ................................. 4
   2.1.1 The GDS, GSM, and GPL classes .................... 4
   2.1.2 The GSE class ............................................. 8
2.2 Converting to BioConductor ExpressionSets and limma MALists ................. 10
   2.2.1 Getting GSE Series Matrix files as an ExpressionSet .... 10
   2.2.2 Converting GDS to an ExpressionSet ................. 11
   2.2.3 Converting GDS to an MAList ......................... 13
   2.2.4 Converting GSE to an ExpressionSet ................. 18
2.3 Accessing Raw Data from GEO ............................. 24
2.4 Use Cases ....................................................... 25
   2.4.1 Getting all Series Records for a Given Platform .... 25
   2.4.2 Building a Selective NCBI GEO mirror ............ 26
2.5 GEOquery Summary ............................................ 26

*sdavis2@mail.nih.gov
# 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.\footnote{See \url{http://www.ncbi.nih.gov/geo} for more information}

1

---

**3 The GEOmetadb Package**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Introduction</td>
<td>26</td>
</tr>
<tr>
<td>3.1.1 What is GEOmetadb?</td>
<td>27</td>
</tr>
<tr>
<td>3.1.2 Conversion capabilities</td>
<td>28</td>
</tr>
<tr>
<td>3.1.3 What GEOmetadb is not</td>
<td>28</td>
</tr>
<tr>
<td>3.2 Getting Started</td>
<td>28</td>
</tr>
<tr>
<td>3.2.1 Getting the GEOmetadb database</td>
<td>28</td>
</tr>
<tr>
<td>3.2.2 A word about SQL</td>
<td>29</td>
</tr>
<tr>
<td>3.3 Examples</td>
<td>30</td>
</tr>
<tr>
<td>3.3.1 Interacting with the database</td>
<td>30</td>
</tr>
<tr>
<td>3.3.2 Writing SQL queries and getting results</td>
<td>31</td>
</tr>
<tr>
<td>3.3.3 Conversion of GEO entity types</td>
<td>34</td>
</tr>
<tr>
<td>3.3.4 Mappings between GPL and Bioconductor microarray annotation packages</td>
<td>36</td>
</tr>
<tr>
<td>3.3.5 More advanced queries</td>
<td>36</td>
</tr>
</tbody>
</table>

**4 Introduction to SRA and the SRAdb Package**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Preliminaries</td>
<td>37</td>
</tr>
<tr>
<td>4.2 Using the SRAdb package</td>
<td>38</td>
</tr>
<tr>
<td>4.2.1 Interacting with the database</td>
<td>38</td>
</tr>
<tr>
<td>4.2.2 Writing SQL queries and getting results</td>
<td>40</td>
</tr>
<tr>
<td>4.2.3 Finding Relationships Between SRA Entities</td>
<td>41</td>
</tr>
<tr>
<td>4.2.4 Full text search</td>
<td>42</td>
</tr>
<tr>
<td>4.2.5 Get SRA or SRA-lite Data File Information</td>
<td>43</td>
</tr>
<tr>
<td>4.3 Interactive Views of Sequence Data</td>
<td>44</td>
</tr>
<tr>
<td>4.4 Graphical View of SRA Entity Relationships</td>
<td>45</td>
</tr>
</tbody>
</table>

**5 sessionInfo**

46
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 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 Using GEOquery to Access NCBI GEO

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. See the Bioconductor website for how to install GEOquery. Assuming that the installation was successful, usage is quite simple:
> library(GEOquery)

This loads the GEOquery library.

> # 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. If you like, you can visit the url [http://www.ncbi.nlm.nih.gov/sites/GDSbrowser?acc=GDS507](http://www.ncbi.nlm.nih.gov/sites/GDSbrowser?acc=GDS507) to see the webpage for this GDS entry. 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.

> # 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"))

2.1 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.

2.1.1 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:

> # Look at gsm metadata:
> 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"

$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 Fuhrman Grade = 3; Lymph Node Invasion = true; Renal Vein Invasion = true; Scaling Target = 500; Scaling Factor = 7.09; Raw Q = 2.39; Noise = 2.60; Background = 55.24."

[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

$title
[1] "N035 Normal Human Kidney U133A"

$type
[1] "RNA"
There is a lot of useful information in the Metadata section of a GSM, GDS, or GPL object. The Meta method returns a list, so one can pull out relevant information as needed. Note that the GEOmetadb that we will discuss next has parsed all of these sections into a SQLite database, so searching based on metadata becomes straightforward.

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

<table>
<thead>
<tr>
<th>ID_REF</th>
<th>VALUE</th>
<th>ABS_CALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFFX-BioB-5_at</td>
<td>953.9</td>
<td>P</td>
</tr>
<tr>
<td>AFFX-BioB-M_at</td>
<td>2982.8</td>
<td>P</td>
</tr>
<tr>
<td>AFFX-BioB-3_at</td>
<td>1657.9</td>
<td>P</td>
</tr>
<tr>
<td>AFFX-BioC-5_at</td>
<td>2652.7</td>
<td>P</td>
</tr>
<tr>
<td>AFFX-BioC-3_at</td>
<td>2019.5</td>
<td>P</td>
</tr>
</tbody>
</table>

The Table method returns a data.frame, typically. It contains the data values for the GEO entity.

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

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ID_REF</td>
<td>MAS 5.0 Statistical Algorithm (mean scaled to 500)</td>
</tr>
<tr>
<td>2 VALUE</td>
<td>MAS 5.0 Absent, Marginal, Present call with Alpha1 = 0.05, Alpha2 = 0.065</td>
</tr>
</tbody>
</table>

The columns present in the GEOdataTable class object are described in some detail. The GPL behaves exactly as the GSM class. However, the GDS has a bit more information associated with the Columns method:

> Columns(gds)

<table>
<thead>
<tr>
<th>sample</th>
<th>disease.state</th>
<th>individual</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSM11815</td>
<td>RCC 035</td>
<td></td>
</tr>
<tr>
<td>GSM11832</td>
<td>RCC 023</td>
<td></td>
</tr>
<tr>
<td>GSM12069</td>
<td>RCC 001</td>
<td></td>
</tr>
<tr>
<td>GSM12083</td>
<td>RCC 005</td>
<td></td>
</tr>
<tr>
<td>GSM12101</td>
<td>RCC 011</td>
<td></td>
</tr>
<tr>
<td>GSM12106</td>
<td>RCC 032</td>
<td></td>
</tr>
<tr>
<td>GSM</td>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>GSM12274</td>
<td>RCC</td>
<td>2</td>
</tr>
<tr>
<td>GSM12299</td>
<td>RCC</td>
<td>3</td>
</tr>
<tr>
<td>GSM12412</td>
<td>RCC</td>
<td>4</td>
</tr>
<tr>
<td>GSM11810</td>
<td>normal</td>
<td>035</td>
</tr>
<tr>
<td>GSM11827</td>
<td>normal</td>
<td>023</td>
</tr>
<tr>
<td>GSM12078</td>
<td>normal</td>
<td>001</td>
</tr>
<tr>
<td>GSM12099</td>
<td>normal</td>
<td>005</td>
</tr>
<tr>
<td>GSM12269</td>
<td>normal</td>
<td>1</td>
</tr>
<tr>
<td>GSM12287</td>
<td>normal</td>
<td>2</td>
</tr>
<tr>
<td>GSM12301</td>
<td>normal</td>
<td>3</td>
</tr>
<tr>
<td>GSM12448</td>
<td>normal</td>
<td>4</td>
</tr>
</tbody>
</table>

**2.1.2 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:

```r
> # 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
> class(GSMList(gse)[[1]])

[1] "GSM"
attr("package")
[1] "GEOquery"

> head(Meta(GSMList(gse)[[1]]))

$channel_count
[1] "1"

$comment
[1] "Raw data provided as supplementary file"
2.2 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 convert GDS data structures to limma or Biobase data structures.

2.2.1 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.

>` # 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)]])

<table>
<thead>
<tr>
<th>title</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSM48681 Patient sample ST18,</td>
<td>Dermatofibrosarcoma RNA</td>
</tr>
<tr>
<td>GSM48682 Patient sample ST410,</td>
<td>Ewing Sarcoma RNA</td>
</tr>
<tr>
<td>GSM48683 Patient sample ST130,</td>
<td>Sarcoma, NOS RNA</td>
</tr>
<tr>
<td>GSM48684 Patient sample ST293,</td>
<td>Malignant Peripheral Nerve</td>
</tr>
<tr>
<td>GSM48685 Patient sample ST367,</td>
<td>Nerve Sheath Tumor RNA</td>
</tr>
<tr>
<td>source_name_ch1</td>
<td></td>
</tr>
<tr>
<td>GSM48681 Dermatofibrosarcoma</td>
<td></td>
</tr>
<tr>
<td>GSM48682 Ewing Sarcoma</td>
<td></td>
</tr>
<tr>
<td>GSM48683 Sarcoma, NOS</td>
<td></td>
</tr>
<tr>
<td>GSM48684 Malignant Peripheral</td>
<td></td>
</tr>
<tr>
<td>GSM48685 Liposarcoma</td>
<td></td>
</tr>
</tbody>
</table>

2.2.2 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)
> pData(eset)

<table>
<thead>
<tr>
<th>sample</th>
<th>disease.state</th>
<th>individual</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSM11815</td>
<td>GSM11815</td>
<td>RCC</td>
<td>035</td>
</tr>
<tr>
<td>GSM11832</td>
<td>GSM11832</td>
<td>RCC</td>
<td>023</td>
</tr>
<tr>
<td>GSM12069</td>
<td>GSM12069</td>
<td>RCC</td>
<td>001</td>
</tr>
<tr>
<td>GSM12083</td>
<td>GSM12083</td>
<td>RCC</td>
<td>005</td>
</tr>
<tr>
<td>GSM12101</td>
<td>GSM12101</td>
<td>RCC</td>
<td>011</td>
</tr>
<tr>
<td>GSM12106</td>
<td>GSM12106</td>
<td>RCC</td>
<td>032</td>
</tr>
<tr>
<td>GSM12274</td>
<td>GSM12274</td>
<td>RCC</td>
<td>2</td>
</tr>
<tr>
<td>GSM12299</td>
<td>GSM12299</td>
<td>RCC</td>
<td>3</td>
</tr>
<tr>
<td>GSM12412</td>
<td>GSM12412</td>
<td>RCC</td>
<td>4</td>
</tr>
<tr>
<td>GSM11810</td>
<td>GSM11810</td>
<td>normal</td>
<td>035</td>
</tr>
<tr>
<td>GSM11827</td>
<td>GSM11827</td>
<td>normal</td>
<td>023</td>
</tr>
<tr>
<td>GSM12078</td>
<td>GSM12078</td>
<td>normal</td>
<td>001</td>
</tr>
<tr>
<td>GSM12099</td>
<td>GSM12099</td>
<td>normal</td>
<td>005</td>
</tr>
<tr>
<td>GSM12269</td>
<td>GSM12269</td>
<td>normal</td>
<td>1</td>
</tr>
<tr>
<td>GSM12287</td>
<td>GSM12287</td>
<td>normal</td>
<td>2</td>
</tr>
<tr>
<td>GSM12301</td>
<td>GSM12301</td>
<td>normal</td>
<td>3</td>
</tr>
<tr>
<td>GSM12448</td>
<td>GSM12448</td>
<td>normal</td>
<td>4</td>
</tr>
</tbody>
</table>

GSM11815 Value for GSM11815: C035 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
GSM11832 Value for GSM11832: C023 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
GSM12069 Value for GSM12069: C001 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
GSM12083 Value for GSM12083: C005 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
GSM12101 Value for GSM12101: C011 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
GSM12106 Value for GSM12106: C032 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
GSM12274 Value for GSM12274: C2 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
GSM12299 Value for GSM12299: C3 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
GSM12412 Value for GSM12412: C4 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
GSM11810 Value for GSM11810: N035 Normal Human Kidney U133B; src: Trizol isolation of total RNA from normal tissue adjacent to Renal Cell Carcinoma
GSM11827 Value for GSM11827: N023 Normal Human Kidney U133B; src: Trizol isolation of total RNA from normal tissue adjacent to Renal Cell Carcinoma
GSM12078 Value for GSM12078: N001 Normal Human Kidney U133B; src: Trizol isolation of total RNA from normal tissue adjacent to Renal Cell Carcinoma
GSM12099 Value for GSM12099: N005 Normal Human Kidney U133B; src: Trizol isolation
GSM12269 Value for GSM12269: N1 Normal Human Kidney U133B; src: Trizol isolation
GSM12287 Value for GSM12287: N2 Renal Clear Cell Carcinoma U133B; src: Trizol isolation
GSM12301 Value for GSM12301: N3 Renal Clear Cell Carcinoma U133B; src: Trizol isolation
GSM12448 Value for GSM12448: N4 Renal Clear Cell Carcinoma U133B; src: Trizol isolation

2.2.3 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.

> #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 GPL97 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

GSM11815 GSM11832 GSM12069 GSM12083 GSM12101 GSM12106 GSM12274 GSM12299
[1,]  4254.0  5298.2  4026.5  3498.4  3566.4  4903.1  6372.6  4829.1
[2,] 17996.2 12010.7 10283.5  2534.7 11048.4  13354.0  8563.8 17247.6
[3,] 41678.8 39116.9 32847.7  39633.9  43511.2  46856.7  47032.4
[4,] 65390.9 34806.2 31257.2  28308.5  67447.5  56989.9  57972.5  57570.5
[5,] 19030.1 15813.6 16355.7  9579.7 14273.5  17217.0  19116.9 17487.6

GSM12412 GSM11810 GSM11827 GSM12078 GSM12099 GSM12269 GSM12287 GSM12301
[1,]  5205.8  2756.8  3932.0  3729.9  3223.4  3640.5  4886.3  4070.2
[2,] 16018.5  6077.0 15703.8 10138.5 11614.4  8460.5 10282.6 11844.3
[3,] 22152.2 26660.7 26373.6 23809.6 24749.3  21936.8 31462.8 22733.7
[4,] 29062.2 35140.9 23629.3 22100.5 21651.0 18550.7 23496.5 21315.4
[5,] 14671.6 17733.1 18022.4 17957.4 15958.0 15799.8 16685.8 18817.3

GSM12448
[1,]  3482.1
<table>
<thead>
<tr>
<th>ID</th>
<th>Gene.title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Homo sapiens PRP8 pre-mRNA processing factor 8 homolog (S. cerevisiae)</td>
</tr>
<tr>
<td>2</td>
<td>Homo sapiens calpain, small subunit 1 (CAPNS1), transcript variant 1</td>
</tr>
<tr>
<td>3</td>
<td>Homo sapiens ribosomal protein L35</td>
</tr>
<tr>
<td>4</td>
<td>Homo sapiens ribosomal protein L28</td>
</tr>
<tr>
<td>5</td>
<td>Homo sapiens ribosomal protein L28</td>
</tr>
</tbody>
</table>

```sql
$A
NULL

$targets
description
table
1 Value for GSM11815: C035 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
2 Value for GSM11832: C023 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
3 Value for GSM12069: C001 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
4 Value for GSM12083: C005 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
5 Value for GSM12101: C011 Renal Clear Cell Carcinoma U133B; src: Trizol isolation of total RNA from Renal Clear Cell Carcinoma tissue
```
Homo sapiens eukaryotic translation initiation factor 4 gamma, 2 (EIF4G2), transcript variant 1, mRNA

<table>
<thead>
<tr>
<th>GI</th>
<th>GenBank.Accession Platform</th>
<th>CLONEID</th>
<th>Platform.ORF Platform</th>
<th>SPOTID</th>
<th>Chromosome.location</th>
<th>Chromosome.annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>91208425</td>
<td>NM_006445</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>17p13.3</td>
<td>Chromosome 17, NC_000017.9 (1500673..1534926, complement)</td>
</tr>
<tr>
<td>2</td>
<td>51599152</td>
<td>NM_001749</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>19q13.12</td>
<td>Chromosome 19, NC_000019.8 (41322758..41333095)</td>
</tr>
<tr>
<td>3</td>
<td>78190471</td>
<td>NM_007209</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>9q34.1</td>
<td>Chromosome 9, NC_000009.10 (126659979..126664061, complement)</td>
</tr>
<tr>
<td>4</td>
<td>34486095</td>
<td>NM_000991</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>19q13.4</td>
<td>Chromosome 19, NC_000019.8 (60589112..60595265)</td>
</tr>
<tr>
<td>5</td>
<td>111494227</td>
<td>NM_001418</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>11p15</td>
<td>Chromosome 11, NC_000011.8 (10775169..10787158, complement)</td>
</tr>
</tbody>
</table>

RNA binding///RNA splicing factor activity, transesterification

- mRNA binding///protein binding///structural constituent of ribosome

- RNA binding///protein binding///structural constituent of ribosome///structural constituent of ribosome

- protein binding///protein binding///translation initiation factor activity///translation initiation factor activity

RNA metabolic process///cell cycle arrest///

GO.Component

- nuclear speck///nucleus///snRNP U5///spliceosome

- cytoplasm///plasma membrane

- cytosol///cytosolic large ribosomal subunit///intracellular///nucleolus///ribosome

- cytosol///cytosolic large ribosomal subunit///intracellular///ribosome

- eukaryotic translation initiation factor 4F complex

GO.Function.1

| GO:0003723///GO:00031202///GO:0005515  |
| GO:0005515///GO:0005515///GO:0003735  |
| GO:0003729///GO:0005515///GO:0003735  |
| GO:0003723///GO:0005515///GO:0003735///GO:0003735  |
5 GO:0005515///GO:0005515///GO:0003743///GO:0003743
GO.Process.1
1 GO:0008380///GO:0000398///GO:0000398///GO:0050896///GO:0007601
2 GO:0008284
3 GO:0006414
4 GO:0006412///GO:0006414
5 GO:0016070///GO:0007050///GO:0008219///GO:0006446
GO.Component.1
1 GO:0016607///GO:0005634///GO:0005682///GO:0005681
2 GO:0005737///GO:0005886
3 GO:0005829///GO:0022625///GO:0005622///GO:0005730///GO:0005840
4 GO:0005829///GO:0022625///GO:0005622///GO:0005840
5 GO:0016281

22640 more rows ...

$notes
$channel_count
[1] "1"

$dataset_id
[1] "GDS507" "GDS507" "GDS507" "GDS507" "GDS507" "GDS507" "GDS507"
[9] "GDS507" "GDS507" "GDS507" "GDS507"

$description
[1] "Investigation into mechanisms of renal clear cell carcinogenesis (RCC). Comparison..."
[2] "RCC"
[3] "normal"
[4] "035"
[5] "023"
[6] "001"
[7] "005"
[8] "011"
[9] "032"
[10] "1"
[11] "2"
[12] "3"
[13] "4"

$email
[1] "geo@ncbi.nlm.nih.gov"

$feature_count
[1] "22645"

$institute
[1] "NCBI NLM NIH"

$name
[1] "Gene Expression Omnibus (GEO)"

$order
[1] "none"

$platform
[1] "GPL97"

$platform_organism
[1] "Homo sapiens"

$platform_technology_type
[1] "in situ oligonucleotide"

$pubmed_id
[1] "14641932"

$ref
[1] "Nucleic Acids Res. 2005 Jan 1;33 Database Issue:D562-6"

$reference_series
[1] "GSE781"

$sample_count
[1] "17"

$sample_id
[1] "GSM11815,GSM11832,GSM12069,GSM12083,GSM12101,GSM12106,GSM12274,GSM12299,GSM12412"
[2] "GSM11810,GSM11827,GSM12078,GSM12099,GSM12269,GSM12287,GSM12301,GSM12448"
[3] "GSM11810,GSM11815"
[4] "GSM11827,GSM11832"
[5] "GSM12069,GSM12078"
[6] "GSM12083,GSM12099"
[7] "GSM12101"
[8] "GSM12106"
[9] "GSM12269"
Now, MA is of class MAList and contains not only the data, but the sample information and gene information associated with GDS507.

2.2.4 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
First, we need to make sure that all of the GSMs are from the same platform:

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

```r
$GSM11805
 [1] "GPL96"

$GSM11810
 [1] "GPL97"

$GSM11814
 [1] "GPL96"

$GSM11815
 [1] "GPL97"

$GSM11823
 [1] "GPL96"

$GSM11827
 [1] "GPL97"

$GSM11830
 [1] "GPL96"

$GSM11832
 [1] "GPL97"

$GSM12067
 [1] "GPL96"

$GSM12069
 [1] "GPL97"

$GSM12075
 [1] "GPL96"

$GSM12078
 [1] "GPL97"

$GSM12079
```
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).

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

<table>
<thead>
<tr>
<th>ID_REF</th>
<th>VALUE</th>
<th>ABS_CALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFFX-BioB-5_at</td>
<td>953.9</td>
<td>P</td>
</tr>
<tr>
<td>AFFX-BioB-M_at</td>
<td>2982.8</td>
<td>P</td>
</tr>
<tr>
<td>AFFX-BioB-3_at</td>
<td>1657.9</td>
<td>P</td>
</tr>
<tr>
<td>AFFX-BioC-5_at</td>
<td>2652.7</td>
<td>P</td>
</tr>
<tr>
<td>AFFX-BioC-3_at</td>
<td>2019.5</td>
<td>P</td>
</tr>
</tbody>
</table>
```

> # and get the column descriptions
> Columns(GSMList(gse)[[1]])[1:5,]
<table>
<thead>
<tr>
<th>Column</th>
<th>ID_REF</th>
<th>VALUE</th>
<th>ABS_CALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NA</td>
<td>&lt;NA&gt;</td>
<td>NA</td>
<td>&lt;NA&gt;</td>
</tr>
</tbody>
</table>

Description

1. MAS 5.0 Statistical Algorithm (mean scaled to 500)
2. MAS 5.0 Absent, Marginal, Present call with Alpha1 = 0.05, Alpha2 = 0.065

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

```r
# 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])
}))
# 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

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
```
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.

### 2.3 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.

As a simple example, download the supplemental file for the GEO sample, GSM3922.

```r
> df = getGEOSuppFiles('GSM3922')

```

The metadata information for the file is stored in the returned data.frame, `df`. In this case, there is only one row, but there could be more than one row, so the returned data frame can be useful.

```r
> df

/Volumes/mstore/home/sdavis/Documents/git/publicDataTutorial/inst/doc/GSM3922/GSM3922.CEL.gz
/Volumes/mstore/home/sdavis/Documents/git/publicDataTutorial/inst/doc/GSM3922/GSM3922.CEL.gz
/Volumes/mstore/home/sdavis/Documents/git/publicDataTutorial/inst/doc/GSM3922/GSM3922.CEL.gz
/Volumes/mstore/home/sdavis/Documents/git/publicDataTutorial/inst/doc/GSM3922/GSM3922.CEL.gz
/Volumes/mstore/home/sdavis/Documents/git/publicDataTutorial/inst/doc/GSM3922/GSM3922.CEL.gz
/Volumes/mstore/home/sdavis/Documents/git/publicDataTutorial/inst/doc/GSM3922/GSM3922.CEL.gz
/Volumes/mstore/home/sdavis/Documents/git/publicDataTutorial/inst/doc/GSM3922/GSM3922.CEL.gz
/Volumes/mstore/home/sdavis/Documents/git/publicDataTutorial/inst/doc/GSM3922/GSM3922.CEL.gz
/Volumes/mstore/home/sdavis/Documents/git/publicDataTutorial/inst/doc/GSM3922/GSM3922.CEL.gz
/Volumes/mstore/home/sdavis/Documents/git/publicDataTutorial/inst/doc/GSM3922/GSM3922.CEL.gz
```

24
2.4 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.

2.4.1 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)$title
> head(Meta(gpl97)$series_id)
[1] "GSE362" "GSE473" "GSE620" "GSE674" "GSE781" "GSE907"
> length(Meta(gpl97)$series_id)
[1] 124
> head(Meta(gpl97)$sample_id)
[1] "GSM3922" "GSM3924" "GSM3926" "GSM3928" "GSM3930" "GSM3932"
> length(Meta(gpl97)$sample_id)
[1] 5164
```

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 more than 120 series associated with this platform and more than 5100 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
> # Feel free to run the next two lines, but I leave them out
> # here to cut down on processing time
> # gsmlist <- sapply(gsmids[1:5],getGEO)
> # names(gsmlist)
```

25
2.4.2 Building a Selective NCBI GEO mirror

GEOquery has the ability to use a "local repository" of NCBI GEO. Enabling this functionality is very simple. Simply supply a destination directory to `getGEO` and any files that GEOquery would normally get from NCBI via download will be taken from the destination directory if available. In other words, GEOquery will use a simple caching system. If the destination directory is used consistently, the result will be a local GEOquery mirror populated with all previously-downloaded GEO records. An example is probably most useful here:

```
> destdir = tempdir()
> # this will be downloaded
> x = getGEO('GDS507', destdir=destdir)
```

Since the GDS507 file is now on disk, why redownload?

```
> # this will NOT be downloaded
> # local copy will be used instead
> y = getGEO('GDS507', destdir=destdir)
```

2.5 GEOquery Summary

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.

3 The GEOmetadb Package

One difficulty in dealing with GEO is finding the microarray data that is of interest. As part of the NCBI Entrez search system, GEO can be searched online via web pages or using NCBI Eutils. However, the web search is not as full-featured as it could be, particularly for programmatic access and data mining. NCBI Eutils offers another option for finding data within the vast stores of GEO, but it is cumbersome to use, often requiring multiple complicated Eutils calls to get at the relevant information. We have found it absolutely critical to have ready access not just to the microarray data, but to the metadata describing the microarray experiments. To this end we have created GEOmetadb.

3.1 Introduction

In this section, we present a high-level overview of GEOmetadb.
3.1.1 What is GEOmetadb?

The GEOmetadb is an attempt to make querying the metadata describing microarray experiments, platforms, and datasets both easier and more powerful. At the heart of GEOmetadb is a SQLite database that stores nearly all the metadata associated with all GEO data types including GEO samples (GSM), GEO platforms (GPL), GEO data series (GSE), and curated GEO datasets (GDS), as well as the relationships between these data types. This database is generated by our server by parsing all the records in GEO and needs to be downloaded via a simple helper function to the user’s local machine before GEOmetadb is useful. Once this is done, the entire GEO database is accessible with simple SQL-based queries. With the GEOmetadb database, queries that are simply not possible using NCBI tools or web pages are often quite simple. The relationships between the tables in the GEOmetadb SQLite database can be seen in figure 1.
3.1.2 Conversion capabilities

A very typical problem for large-scale consumers of GEO data is to determine the relationships between various GEO accession types. As examples, consider the following questions:

- What samples are associated with GEO platform “GPL96”, which represents the Affymetrix hgu133a array?
- What GEO Series were performed using “GPL96”?
- What samples are in my favorite three GEO Series records?
- How many samples are associated with the ten most popular GEO platforms?

Because these types of questions are common, GEOmetadb contains the function geoConvert that addresses these questions directly and efficiently.

3.1.3 What GEOmetadb is not

We have faithfully parsed and maintained in GEO when creating GEOmetadb. This means that limitations inherent to GEO are also inherent in GEOmetadb. We have made no attempt to curate, semantically recode, or otherwise “clean up” GEO; to do so would require significant resources, which we do not have.

GEOmetadb does not contain any microarray data. For access to microarray data from within R/Bioconductor, please look at the GEOquery package. In fact, we would expect that many users will find that the combination of GEOmetadb and GEOquery is quite powerful.

3.2 Getting Started

Once GEOmetadb is installed (see the Bioconductor website for full installation instructions), we are ready to begin.

3.2.1 Getting the GEOmetadb database

This package does not come with a pre-installed version of the database. This has the advantage that the user will get the most up-to-date version of the database to start; the database can be re-downloaded using the same command as often as desired. First, load the library.

```r
> library(GEOmetadb)
```

The download and uncompress steps are done automatically with a single command, getSQLiteFile.

```r
> if(!file.exists('GEOmetadb.sqlite')) {
+   getSQLiteFile()
+ }
```
The default storage location is in the current working directory and the default filename is “GEOmetadb.sqlite”; it is best to leave the name unchanged unless there is a pressing reason to change it.

Since this SQLite file is of key importance in GEOmetadb, it is perhaps of some interest to know some details about the file itself.

> file.info('GEOmetadb.sqlite')

        size isdir mode
GEOmetadb.sqlite 1587592192 FALSE 644

        mtime                 ctime
GEOmetadb.sqlite 2011-07-29 00:58:41 2011-07-29 00:58:41

        atime           uid   gid   uname
GEOmetadb.sqlite 2011-07-29 01:23:42 10005 513   sdavis

GEOmetadb.sqlite Domain Users

Now, the SQLite file is available for connection. The standard DBI functionality as implemented in RSQLite function dbConnect makes the connection to the database. The dbDisconnect function disconnects the connection.

> con <- dbConnect(SQLite(), 'GEOmetadb.sqlite')
> dbDisconnect(con)

[1] TRUE

The variable con is an RSQLite connection object.

3.2.2 A word about SQL

The Structured Query Language, or SQL, is a very powerful and standard way of working with relational data. GEO is composed of several data types, all of which are related to each other; in fact, NCBI uses a relational SQL database for metadata storage and querying. SQL databases and SQL itself are designed specifically to work efficiently with just such data. While the goal of many programming projects and programmers is to hide the details of SQL from the user, we are of the opinion that such efforts may be counterproductive, particularly with complex data and the need for ad hoc queries, both of which are characteristics with GEO metadata. We have taken the view that exposing the power of SQL will enable users to maximally utilize the vast data repository that is GEO. We understand that many users are not accustomed to working with SQL and, therefore, have devoted a large section of the vignette to working examples. Our goal is not to teach SQL, so a quick tutorial of SQL is likely to be beneficial to those who have not used it before. Many such tutorials are available online and can be completed in 30 minutes or less.
3.3 Examples

3.3.1 Interacting with the database

The functionality covered in this section is covered in much more detail in the DBI and RSQLite package documentation. We cover enough here only to be useful.

Again, we connect to the database.

```r
> con <- dbConnect(SQLite(), 'GEOmetadb.sqlite')
```

The `dbListTables` function lists all the tables in the SQLite database handled by the connection object `con`.

```r
> geo_tables <- dbListTables(con)
> geo_tables

[1] "gds"                "gds_subset"
[3] "geoConvert"         "geodb_column_desc"
[5] "gpl"                "gse"
[7] "gse_gpl"            "gse_gsm"
[9] "gsm"                "metaInfo"
```

There is also the `dbListFields` function that can list database fields associated with a table.

```r
> dbListFields(con, 'gse')

[1] "ID"                "title"
[3] "gse"               "status"
[5] "submission_date"   "last_update_date"
[7] "pubmed_id"         "summary"
[9] "type"              "contributor"
[11] "web_link"          "overall_design"
[13] "repeats"           "repeats_sample_list"
[15] "variable"          "variable_description"
[17] "contact"           "supplementary_file"
```

Sometimes it is useful to get the actual SQL schema associated with a table. As an example of doing this and using an RSQLite shortcut function, `sqliteQuickSQL`, we can get the table schema for the `gpl` table.

```r
> sqliteQuickSQL(con, 'PRAGMA TABLE_INFO(gpl)')
```
### 3.3.2 Writing SQL queries and getting results

Select 5 records from the `gse` table and show the first 7 columns.

```r
> rs <- dbGetQuery(con, 'select * from gse limit 5')
> rs[,1:7]

<table>
<thead>
<tr>
<th>ID</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NHGRI_Melanoma_class</td>
</tr>
<tr>
<td>2</td>
<td>Cerebellar development</td>
</tr>
<tr>
<td>3</td>
<td>Renal Cell Carcinoma Differential Expression</td>
</tr>
<tr>
<td>4</td>
<td>Diurnal and Circadian-Regulated Genes in Arabidopsis</td>
</tr>
<tr>
<td>5</td>
<td>Global profile of germline gene expression in C. elegans</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>gse status submission_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSE1 Public on Jan 22 2001 2001-01-22</td>
</tr>
<tr>
<td>GSE2 Public on Apr 26 2001 2001-04-19</td>
</tr>
</tbody>
</table>
```
Get the GEO series accession and title from GEO series that were submitted by “Sean Davis”. The

```r
> rs <- dbGetQuery(con,paste("select gse,title from gse where",
+ "contributor like '%Sean%Davis%',sep=''"))
> rs

<table>
<thead>
<tr>
<th>gse</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSE2553</td>
<td>NHGRI_Sarcoma_Baird</td>
</tr>
<tr>
<td>GSE4406</td>
<td>Gene expression profiling of CD4+ T-cells and GM6990</td>
</tr>
<tr>
<td>GSE5357</td>
<td>lymphoblastoid cell lines</td>
</tr>
<tr>
<td>GSE7376</td>
<td>NHGRI Menin ChIP-Chip</td>
</tr>
<tr>
<td>GSE7882</td>
<td>Detection of novel amplification units in prostate cancer</td>
</tr>
<tr>
<td>GSE8486</td>
<td>Gene Expression and Comparative Genomic Hybridization</td>
</tr>
<tr>
<td>GSE9328</td>
<td>of Ductal Carcinoma In Situ of the Breast</td>
</tr>
<tr>
<td>GSE14543</td>
<td>Whole genome DNAse hypersensitivity in human CD4+ T-cells</td>
</tr>
<tr>
<td>GSE15621</td>
<td></td>
</tr>
<tr>
<td>GSE16087</td>
<td></td>
</tr>
<tr>
<td>GSE16088</td>
<td></td>
</tr>
<tr>
<td>GSE16091</td>
<td></td>
</tr>
<tr>
<td>GSE16102</td>
<td></td>
</tr>
<tr>
<td>GSE18544</td>
<td></td>
</tr>
<tr>
<td>GSE19063</td>
<td></td>
</tr>
<tr>
<td>GSE20016</td>
<td></td>
</tr>
<tr>
<td>GSE25164</td>
<td></td>
</tr>
<tr>
<td>GSE22520</td>
<td></td>
</tr>
<tr>
<td>GSE25127</td>
<td></td>
</tr>
<tr>
<td>GSE25127</td>
<td></td>
</tr>
</tbody>
</table>
```

1 Gene expression profiling
2 Detection of novel amplification units in prostate cancer
3 Gene Expression and Comparative Genomic Hybridization
4 Whole genome DNAse hypersensitivity in human CD4+ T-cells
5
6
As another example, GEOmetadb can find all samples on GPL96 (Affymetrix hgu133a) that have .CEL files available for download.

```r
> rs <- dbGetQuery(con,paste("select gsm,supplementary_file",
+ "from gsm where gpl='GPL96'",
+ "and supplementary_file like '%CEL.gz'"))
> dim(rs)
[1] 18910 2
```

But why limit to only GPL96? Why not look for all Affymetrix arrays that have .CEL files? And list those with their associated GPL information, as well as the Bioconductor annotation package name?

```r
> rs <- dbGetQuery(con,paste("select gpl.bioc_package,gsm.gpl,",
+ "gsm,gsm.supplementary_file",
+ "from gsm join gpl on gsm.gpl=gpl.gpl",
+ "where gpl.manufacturer='Affymetrix'",
+ "and gsm.supplementary_file like '%CEL.gz'"))
> rs[1:5,]
```

<table>
<thead>
<tr>
<th>bioc_package</th>
<th>gpl</th>
<th>gsm</th>
<th>supplementary_file</th>
</tr>
</thead>
<tbody>
<tr>
<td>hu6800</td>
<td>GPL80</td>
<td>GSM577</td>
<td></td>
</tr>
<tr>
<td>hu6800</td>
<td>GPL80</td>
<td>GSM578</td>
<td></td>
</tr>
<tr>
<td>hu6800</td>
<td>GPL80</td>
<td>GSM579</td>
<td></td>
</tr>
</tbody>
</table>
Of course, we can combine programming and data access. A simple `sapply` example shows how to query each of the tables for number of records.

```r
> getTableCounts <- function(tableName, conn) {
+   sql <- sprintf("select count(*) from %s", tableName)
+   return(dbGetQuery(conn, sql)[1,1])
+ }
> do.call(rbind, sapply(geo_tables, getTableCounts, conn, simplify=FALSE))

       [,1]        
gds  2721  
gds_subset  15275  
geoConvert 2722824  
geodb_column_desc 104  
gpl  9084  
gse  23882  
gse_gpl  30048  
gse_gsm 691134  
gsm 580708  
metaInfo  2  
sMatrix 27040  
```

3.3.3 Conversion of GEO entity types

Large-scale consumers of GEO data might want to convert GEO entity type from one to others, e.g. finding all GSM and GSE associated with 'GPL96'. Function `goeConvert` does the conversion with a very fast mapping between entity types.

Covert 'GPL96' to other possible types in the GEOmetadb.sqlite.

```r
> conversion <- goeConvert("GPL96")
```

Check what GEO types and how many entities in each type in the conversion.

```r
> lapply(conversion, dim)

$gse
[1] 859 2

$gsm
[1] 28220 2
```
<table>
<thead>
<tr>
<th>from_acc</th>
<th>to_acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPL96</td>
<td>GSE1000</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSE10024</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSE10043</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSE10072</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSE10089</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>from_acc</th>
<th>to_acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPL96</td>
<td>GSM100386</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSM100454</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSM100455</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSM100456</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSM100457</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>from_acc</th>
<th>to_acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPL96</td>
<td>GDS1023</td>
</tr>
<tr>
<td>GPL96</td>
<td>GDS1036</td>
</tr>
<tr>
<td>GPL96</td>
<td>GDS1050</td>
</tr>
<tr>
<td>GPL96</td>
<td>GDS1062</td>
</tr>
<tr>
<td>GPL96</td>
<td>GDS1063</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>from_acc</th>
<th>to_acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPL96</td>
<td>GSE1000_series_matrix.txt.gz</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSE10024_series_matrix.txt.gz</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSE10043_series_matrix.txt.gz</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSE10072_series_matrix.txt.gz</td>
</tr>
<tr>
<td>GPL96</td>
<td>GSE10089_series_matrix.txt.gz</td>
</tr>
</tbody>
</table>
3.3.4 Mappings between GPL and Bioconductor microarray annotation packages

The function getBiocPlatformMap is to get GPL information of a given list of Bioconductor microarray annotation packages. Note currently the GEOmetadb does not contain all the mappings, but we are trying to construct a relative complete list.

Get GPL information of ‘hgu133a’ and ‘hgu95av2’:

```r
> getBiocPlatformMap(con, bioc=c('hgu133a','hgu95av2'))
```

<table>
<thead>
<tr>
<th>title gpl</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 [HG-U133A] Affymetrix Human Genome U133A Array GPL96</td>
</tr>
<tr>
<td>2 [HG_U95A] Affymetrix Human Genome U95A Array GPL91</td>
</tr>
</tbody>
</table>

| bioc_package manufacturer organism data_row_count |
|--------------------------|-----------------|-----------------|
| hgu133a Affymetrix Homo sapiens | 22283 |
| hgu95av2 Affymetrix Homo sapiens | 12626 |

3.3.5 More advanced queries

Now, for something a bit more complicated, we would like to find all the human breast cancer-related Affymetrix gene expression GEO series.

```r
> sql <- paste("SELECT DISTINCT gse.title,gse.gse",
+ " FROM",
+ " gsm JOIN gse_gsm ON gsm.gsm=gse_gsm.gsm",
+ " JOIN gse ON gse_gsm.gse=gse.gse",
+ " JOIN gse_gpl ON gse_gpl.gse=gse.gse",
+ " JOIN gpl ON gse_gpl.gpl=gpl.gpl",
+ "WHERE",
+ " gsm.molecule_ch1 like '%total RNA%' AND",
+ " gse.title LIKE '%breast cancer%' AND",
+ " gpl.organism LIKE '%Homo sapiens%'",sep=" ")
> rs <- dbGetQuery(con,sql)
> dim(rs)
[1] 285 2

> print(rs[1:5,],right=FALSE)

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 A Modular Analysis of Breast Cancer Reveals a Novel Low-Grade Molecular Signature in Estrogen Receptor-Positive Tumors</td>
</tr>
<tr>
<td>2 A Phase II Study of Neoadjuvant Gemcitabine Plus Doxorubicin Followed by Gemcitabine Plus Cisplatin in Breast Cancer</td>
</tr>
<tr>
<td>3 A Supervised Risk Predictor of Breast Cancer Based on Biological Subtypes</td>
</tr>
</tbody>
</table>

36
Finally, it is probably a good idea to close the connection, please see \textit{DBI} for detail.

\begin{verbatim}
> dbDisconnect(con)
\end{verbatim}

\begin{verbatim}
[1] TRUE
\end{verbatim}

If you want to remove old GEOmetadb.sqlite file before retrieve a new version from the server, execute the following codes:

\begin{verbatim}
> file.remove('GEOmetadb.sqlite')
\end{verbatim}

## 4 Introduction to SRA and the \textit{SRAdb} Package

High throughput sequencing technologies have very rapidly become standard tools in biology. The data that these machines generate are large, extremely rich. As such, the Sequence Read Archives (SRA) have been set up at NCBI in the United States, EMBL in Europe, and DDBJ in Japan to capture these data in public repositories in much the same spirit as MIAME-compliant microarray databases like NCBI GEO and EBI ArrayExpress.

Accessing data in SRA requires finding it first. This R package provides a convenient and powerful framework to do just that. In addition, \textit{SRAdb} features functionality to determine availability of sequence files and to download files of interest.

SRA does not currently store aligned reads or any other processed data that might rely on alignment to a reference genome. However, NCBI GEO does often contain aligned reads for sequencing experiments and the \textit{SRAdb} package can help to provide links to these data as well. In combination with the \textit{GEOmetadb} and \textit{GEOquery} packages, these data are also, then, accessible.

### 4.1 Preliminaries

Since SRA is a continuously growing repository, the SRAdb SQLite file is updated regularly. The first step, then, is to get the SRAdb SQLite file from the online location. The download and uncompress steps are done automatically with a single command, \texttt{getSRAdbFile}. 

37
> library(SRAdb)
> if(!file.exists('SRAdb.sqlite')) {
+   sqlfile <- getSRAdbFile()
+ }

The default storage location is in the current working directory and the default filename is “SRAmetadb.sqlite”; it is best to leave the name unchanged unless there is a pressing reason to change it. Since this SQLite file is of key importance in SRAdb, it is perhaps of some interest to know some details about the file itself.

> file.info('SRAmetadb.sqlite')

    size  isdir mode mtime  ctime  atime  uid
gid uname grname
SRAmetadb.sqlite NA NA <NA>  <NA>  <NA>  NA
SRAmetadb.sqlite NA <NA>  <NA>

Then, create a connection for later queries. The standard DBI functionality as implemented in RSQLite function dbConnect makes the connection to the database. The dbDisconnect function disconnects the connection.

> #open connection
> sra_con <- dbConnect(SQLite(), 'SRAdb.sqlite')
> sra_con

<SQLiteConnection: DBI CON (2241, 3)>

For further details, at this time see help('SRAdb-package').

4.2 Using the SRAdb package

4.2.1 Interacting with the database

The functionality covered in this section is covered in much more detail in the DBI and RSQLite package documentation. We cover enough here only to be useful. The dbListTables function lists all the tables in the SQLite database handled by the connection object sra_con created in the previous section.

> sra_tables <- dbListTables(sra_con)
> sra_tables

[1] "col_desc"     "data_block"
[3] "experiment"  "metaInfo"
[5] "run"          "sample"
[7] "sra"          "sra_ft"
[9] "sra_ft_content" "sra_ft_segdir"
[11] "sra_ft_segments" "study"
[13] "submission"
There is also the `dbListFields` function that can list database fields associated with a table.

```r
> dbListFields(sra_con, 'study')
```

```
[1] "study_ID"        "study_alias"
[3] "study_accession" "study_title"
[5] "study_type"      "study_abstract"
[7] "center_name"     "center_project_name"
[9] "project_id"      "study_description"
[11] "study_url_link"  "study_entrez_link"
[13] "study_attribute" "submission_accession"
[15] "sradb_updated"
```

Sometimes it is useful to get the actual SQL schema associated with a table. As an example of doing this and using an RSQLite shortcut function, `sqliteQuickSQL`, we can get the table schema for the `study` table.

```r
> sqliteQuickSQL(sra_con, 'PRAGMA TABLE_INFO(study)')
```

```
cid  name     type    notnull dflt_value
1    0 study_ID REAL    0 <NA>
2    1 study_alias TEXT   0 <NA>
3    2 study_accession TEXT 0 <NA>
4    3 study_title TEXT   0 <NA>
5    4 study_type TEXT   0 <NA>
6    5 study_abstract TEXT 0 <NA>
7    6 center_name TEXT   0 <NA>
8    7 center_project_name TEXT 0 <NA>
9    8 project_id INTEGER 0 <NA>
10   9 study_description TEXT 0 <NA>
11   10 study_url_link TEXT 0 <NA>
12   11 study_entrez_link TEXT 0 <NA>
13   12 study_attribute TEXT 0 <NA>
14   13 submission_accession TEXT 0 <NA>
15   14 sradb_updated TEXT 0 <NA>

pk
1 0
2 0
3 0
4 0
5 0
6 0
7 0
```

39
4.2.2 Writing SQL queries and getting results

Select 3 records from the `study` table and show the first 5 columns:

```r
> rs <- dbGetQuery(sra_con, 'select * from study limit 3')
> rs[,1:5]

  study_ID study_alias study_accession study_title
1      1         Natto BEST195       DRP000001 Whole genome sequencing of Bacillus subtilis subsp. natto BEST195
2      2       Resequence B. subtilis 168 DRP000002 Whole genome resequencing of Bacillus subtilis subsp. subtilis str. 168
3      3    KU_MeDIPseq_2009       DRP000030 Whole-genome DNA methylation analysis in human breast cancer cell lines using MeDIP-seq
```

Get the SRA study accessions and titles from SRA study that study_type contains “Transcriptome”. The “%” sign is used in combination with the “like” operator to do a “wildcard” search for the term “Transcriptome” with any number of characters after it.

```r
> rs <- dbGetQuery(sra_con, paste("select study_accession,study_title from study where study_description like 'Transcriptome%'",sep=" "))
> rs[1:3,]

  study_accession study_title
1      SRP000568 Highly integrated epigenome maps in Arabidopsis
2      SRP000714 A Global View of Gene Activity and Alternative Splicing by Deep Sequencing of the Human
3      SRP001122 A Global View of Gene Activity and Alternative Splicing by Deep Sequencing of the Human
```
Of course, we can combine programming and data access. A simple `sapply` example shows how to query each of the tables for number of records.

```r
> getTableCounts <- function(tableName, conn) {
+   sql <- sprintf("select count(*) from %s", tableName)
+   return(dbGetQuery(conn, sql)[1, 1])
+ }
> do.call(rbind, sapply(sra_tables, getTableCounts, sra_con, simplify=FALSE))

[,1]
col_desc      102
data_block     4745
experiment    31637
metaInfo       2
run           77607
sample        108296
sra           82998
sra_ft         82998
sra_ft_content 82998
sra_ft_semdir  21
sra_ft_segments 37677
study         3824
submission    25188

4.2.3 Finding Relationships Between SRA Entities

Large-scale consumers of SRA data might want to convert SRA entity type from one to others, e.g. finding all experiment accessions (SRX, ERX or DRX) and run accessions (SRR, ERR or DRR) associated with 'SRP001007'. Function `sraConvert` does the conversion with a very fast mapping between entity types.

Covert 'SRP001007' to other possible types in the SRAmetadb.sqlite.

```r
> conversion <- sraConvert(c('SRP001007','SRP000931'), sra_con= sra_con)
> conversion[,1:3,]

    study submission sample experiment run
1  SRP000931  SRA009053 SRS003453  SRX006122 SRR018256
2  SRP000931  SRA009053 SRS003453  SRX006129 SRR018263
3  SRP000931  SRA009053 SRS003453  SRX006130 SRR018264

Check what SRA types and how many entities in each type in the conversion.

```r
> apply(conversion, 2, unique)

41
4.2.4 Full text search

Searching by regular table and field specific SQL commands can be very powerful and if you are familiar with SQL language and the table structure. If not, SQLite has a very handy module called Full text search (fts3), which allow users to do Google like search with terms and operators. The function `getSRA` does Full text search against all fields in a fts3 table with terms constructed with the Standard Query Syntax and Enhanced Query Syntax. Please see http://www.sqlite.org/fts3.html for detail.

Find all run and study combined records in which any given fields has 'breast' and 'cancer' words, including 'breast' and 'cancer' are not next to each other:

```r
> rs <- getSRA (search_terms = 'breast cancer', out_types=c('run','study'), sra_con=sra_con)
> dim(rs)
[1] 225 22
```

If you only wants records containing exact phrase of 'breast cancer', in which 'breast' and 'cancer' have other characters between other than a space:

```r
> rs <- getSRA (search_terms ="'breast cancer'", out_types=c('run','study'), sra_con=sra_con)
> dim(rs)
```
Find all sample records containing words of either 'MCF7' or 'MCF-7':

```r
rs <- getSRA (search_terms = 'MCF7 OR "MCF-7"', out_types=c('sample'), sra_con=sra_con)
> dim(rs)
[1] 107 10
```

Find all submissions by GEO:

```r
rs <- getSRA (search_terms = 'submission_center: GEO', out_types=c('submission'), sra_con=sra_con)
> dim(rs)
[1] 527 7
```

Find study records containing a word beginning with 'Carcino':

```r
rs <- getSRA (search_terms = 'Carcino*', out_types=c('study'), sra_con=sra_con)
> dim(rs)
[1] 24 12
```

### 4.2.5 Get SRA or SRA-lite Data File Information

List sra-lite data file names including ftp addresses associated with "SRX000122":

```r
listSRAfile ("SRX000122", sra_con=sra_con, sraType='litesra')
```

```
 experiment
  1 SRX000122
  2 SRX000122
  3 SRX000122
  4 SRX000122
  5 SRX000122
  6 SRX000122

```

The above function does not check file availability, size and date of the sra or sra-lite data files on the server, but the function getSRAinfo does this, which is good to know if you are preparing to download them:
> rs <- getSRAinfo (in_acc=c("SRX000122"), sra_con=sra_con)
> rs[1:3,


<table>
<thead>
<tr>
<th>experiment</th>
<th>size(KB)</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRX000122</td>
<td>104</td>
<td>Apr 7</td>
</tr>
<tr>
<td>SRX000122</td>
<td>50536</td>
<td>Apr 7</td>
</tr>
<tr>
<td>SRX000122</td>
<td>319</td>
<td>Apr 7</td>
</tr>
</tbody>
</table>

Next you might want to download sra or sra-lite data files from the ftp site. The
getSRAfile function will download all available sra or sra-lite data files associated with
"SRR000648" and "SRR000657" from NCBI SRA ftp site to a new folder in current directory:

> getSRAfile (in_acc=c("SRR000648","SRR000657"), sra_con=sra_con, destdir=getwd(), sraType=

4.3 Interactive Views of Sequence Data

This section assumes that the Integrated Genome Browser (IGV) from the Broad Institute
is installed and runs correctly.

Working with sequence data is often best done interactively in a genome browser, a task
not easily done from R itself. We have found the Integrative Genomics Viewer (IGV) a
high-performance visualization tool for interactive exploration of large, integrated datasets,
increasing usefully for visualizing sequence alignments. In SRAdb, functions startIGV,
load2IGV and load2newIGV provide convenient functionality for R to interact with IGV.
Note that for some OS, these functions might not work or work well.

Launch IGV with 2 GB maximum usable memory support:

> startIGV("mm")

IGV offers a remote control port that allows R to communicate with IGV. The current
command set is fairly limited, but it does allow for some IGV operations to be performed in
the R console. To utilize this functionality, be sure that IGV is set to allow communication via
the “enable port” option in IGV preferences. To load BAM files to IGV and then manipulate
the window:

> exampleBams = file.path(system.file('extdata',package='SRAdb'),
+ dir(system.file('extdata',package='SRAdb'),pattern='bam$'))
> sock <- IGVsocket()
> IGVgenome(sock, 'hg18')
> IGVload(sock, exampleBams)
> IGVgoto(sock, 'chr1:1-1000')
> IGVsnapshot(sock)
4.4 Graphical View of SRA Entity Relationships

Due to the nature of SRA data and its design, sometimes it is hard to get a whole picture of the relationship between a set of SRA entities. For example, how many lanes of a given sample were sequenced? In a large study, how is the sequencing of various samples related to several studies? The functions `entityGraph` and `sraGraph` in this package generate graphNEL objects with edgemode='directed' from input data.frame or directly from search terms, and then the `plot` function can easily draw a graph.

Create a graphNEL object from SRA accessions, which are full text search results of terms 'colon cancer'

```r
> library(Rgraphviz)
> acc <- getSRA (search_terms = 'colon cancer', out_types=c('sra'), sra_con=sra_con, acc_only=TRUE)
> g <- entityGraph(acc)
> attrs <- getDefaultAttrs(list(node=list(fillcolor='lightblue', shape='ellipse')))
> plot(g, attrs= attrs)
```

Create a graphNEL object directly from full text search results of terms 'colon cancer'
> g <- sraGraph('colon cancer', sra_con)
> attrs <- getDefaultAttrs(list(node=list(fillcolor='lightblue', shape='ellipse')))
> plot(g, attrs=attrs)

It’s considered good practice to explicitly disconnect from the database once we are done with it:

> dbDisconnect(sra_con)

[1] TRUE

5 sessionInfo

- R version 2.14.0 Under development (unstable) (2011-06-08 r56096), x86_64-apple-darwin9.8.0
- Locale: C
• Base packages: base, datasets, grDevices, graphics, grid, methods, stats, utils

• Other packages: Biobase 2.13.3, DBI 0.2-5, GEOmetadb 1.13.0, GEOquery 2.19.1, RCurl 1.6-5, RSQLite 0.9-4, Rgraphviz 1.31.1, SRAdb 1.7.0, bitops 1.0-4.1, graph 1.31.1, limma 3.9.5

• Loaded via a namespace (and not attached): XML 3.4-0, tools 2.14.0