

An Introduction to Bioconductor's *ExpressionSet* Class

Seth Falcon, Martin Morgan, and Robert Gentleman

6 October, 2006; revised 9 February, 2007

1 Introduction

Biobase is part of the Bioconductor project, and is used by many other packages. Biobase contains standardized data structures to represent genomic data. The *ExpressionSet* class is designed to combine several different sources of information into a single convenient structure. An *ExpressionSet* can be manipulated (e.g., subsetted, copied) conveniently, and is the input or output from many Bioconductor functions.

The data in an *ExpressionSet* is complicated, consisting of expression data from microarray experiments (`assayData`; `assayData` is used to hint at the methods used to access different data components, as we will see below), 'meta-data' describing samples in the experiment (`phenoData`), annotations and meta-data about the features on the chip or technology used for the experiment (`featureData`, `annotation`), and a flexible structure to describe the experiment (`experimentData`). The *ExpressionSet* class coordinates all of this data, so that you do not usually have to worry about the details. However, an *ExpressionSet* needs to be created in the first place, and creation can be complicated.

In this introduction we learn how to create and manipulate *ExpressionSet* objects, and practice some basic R skills.

2 Preliminaries

2.1 Installing Packages

If you are reading this document and have not yet installed any software on your computer, visit <http://bioconductor.org> and follow the instructions for installing R and Bioconductor. Once you have installed R and Bioconductor, you are ready to go with this document. In the future, you might find that you need to install one or more additional packages. The best way to do this is to start an R session and evaluate commands like

```
> source("http://bioconductor.org/biocLite.R")
> biocLite(c("Biobase"))
```

2.2 Loading Packages

The definition of the *ExpressionSet* class along with many methods for manipulating *ExpressionSet* objects are defined in the *Biobase* package. In general, you need to load class and method definitions before you use them. When using Bioconductor, this means loading R packages using `library` or `require`.

```
> library("Biobase")
```

Exercise 1

What happens when you try to load a package that is not installed?

When using `library`, you get an error message. With `require`, the return value is `FALSE` and a warning is printed.

3 Building an ExpressionSet From .CEL and other files

Many users have access to .CEL or other files produced by microarray chip manufacturer hardware. Usually the strategy is to use a Bioconductor package such as `affyPLM`, `affy`, `oligo`, `limma`, or `arrayMagic` to read these files. These Bioconductor packages have functions (e.g., `ReadAffy`, `expresso`, or `justRMA` in `affy`) to read CEL files and perform preliminary preprocessing, and to represent the resulting data as an *ExpressionSet* or other type of object. Suppose the result from reading and preprocessing CEL or other files is named `object`, and `object` is different from *ExpressionSet*; a good bet is to try, e.g.,

```
> library(convert)
> as(object, "ExpressionSet")
```

It might be the case that no converter is available. The path then is to extract relevant data from `object` and use this to create an *ExpressionSet* using the instructions below.

4 Building an ExpressionSet From Scratch

As mentioned in the introduction, the data from many high-throughput genomic experiments, such as microarray experiments, usually consist of several conceptually distinct parts: assay data, phenotypic meta-data, feature annotations and meta-data, and a description of the experiment. We'll construct each of these components, and then assemble them into an *ExpressionSet*.

4.1 Assay data

One important part of the experiment is a matrix of ‘expression’ values. The values are usually derived from microarrays of one sort or another, perhaps after initial processing by manufacturer software or Bioconductor packages. The matrix has F rows and S columns, where F is the number of features on the chip and S is the number of samples.

A likely scenario is that your assay data is in a ‘tab-delimited’ text file (as exported from a spreadsheet, for instance) with rows corresponding to features and columns to samples. The strategy is to read this file into R using the `read.table` command, converting the result to a *matrix*. A typical command to read a tab-delimited file that includes column ‘headers’ is

```
> dataDirectory <- system.file("extdata", package = "Biobase")
> exprsFile <- file.path(dataDirectory, "exprsData.txt")
> exprs <- as.matrix(read.table(exprsFile, header = TRUE,
+   sep = "\t", row.names = 1, as.is = TRUE))
```

The first two lines create a file path pointing to where the assay data is stored; replace these with a character string pointing to your own file, e.g,

```
> exprsFile <- "c:/path/to/exprsData.txt"
```

(Windows users: note the use of / rather than \; this is because R treats the \ character as an ‘escape’ sequence to change the meaning of the subsequent character). See the help pages for `read.table` for more detail. A common variant is that the character separating columns is a comma (“comma-separated values”, or “csv” files), in which case the `sep` argument might be `sep=","`.

It is always important to verify that the data you have read matches your expectations. At a minimum, check the class and dimensions of `geneData` and take a peak at the first several rows

```
> class(exprs)

[1] "matrix"

> dim(exprs)

[1] 500 26

> colnames(exprs)

[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"
[16] "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

> head(exprs)
```

	A	B	C	D	E	
AFFX-MurIL2_at	192.7420	85.75330	176.7570	135.5750	64.49390	
AFFX-MurIL10_at	97.1370	126.19600	77.9216	93.3713	24.39860	
AFFX-MurIL4_at	45.8192	8.83135	33.0632	28.7072	5.94492	
AFFX-MurFAS_at	22.5445	3.60093	14.6883	12.3397	36.86630	
AFFX-BioB-5_at	96.7875	30.43800	46.1271	70.9319	56.17440	
AFFX-BioB-M_at	89.0730	25.84610	57.2033	69.9766	49.58220	
	F	G	H	I	J	
AFFX-MurIL2_at	76.3569	160.5050	65.9631	56.9039	135.60800	
AFFX-MurIL10_at	85.5088	98.9086	81.6932	97.8015	90.48380	
AFFX-MurIL4_at	28.2925	30.9694	14.7923	14.2399	34.48740	
AFFX-MurFAS_at	11.2568	23.0034	16.2134	12.0375	4.54978	
AFFX-BioB-5_at	42.6756	86.5156	30.7927	19.7183	46.35200	
AFFX-BioB-M_at	26.1262	75.0083	42.3352	41.1207	91.53070	
	K	L	M	N	O	P
AFFX-MurIL2_at	63.44320	78.2126	83.0943	89.3372	91.0615	95.9377
AFFX-MurIL10_at	70.57330	94.5418	75.3455	68.5827	87.4050	84.4581
AFFX-MurIL4_at	20.35210	14.1554	20.6251	15.9231	20.1579	27.8139
AFFX-MurFAS_at	8.51782	27.2852	10.1616	20.2488	15.7849	14.3276
AFFX-BioB-5_at	39.13260	41.7698	80.2197	36.4903	36.4021	35.3054
AFFX-BioB-M_at	39.91360	49.8397	63.4794	24.7007	47.4641	47.3578
	Q	R	S	T	U	
AFFX-MurIL2_at	179.8450	152.4670	180.83400	85.4146	157.98900	
AFFX-MurIL10_at	87.6806	108.0320	134.26300	91.4031	-8.68811	
AFFX-MurIL4_at	32.7911	33.5292	19.81720	20.4190	26.87200	
AFFX-MurFAS_at	15.9488	14.6753	-7.91911	12.8875	11.91860	
AFFX-BioB-5_at	58.6239	114.0620	93.44020	22.5168	48.64620	
AFFX-BioB-M_at	58.1331	104.1220	115.83100	58.1224	73.42210	
	V	W	X	Y	Z	
AFFX-MurIL2_at	146.8000	93.8829	103.85500	64.4340	175.61500	
AFFX-MurIL10_at	85.0212	79.2998	71.65520	64.2369	78.70680	
AFFX-MurIL4_at	31.1488	22.3420	19.01350	12.1686	17.37800	
AFFX-MurFAS_at	12.8324	11.1390	7.55564	19.9849	8.96849	
AFFX-BioB-5_at	90.2215	42.0053	57.57380	44.8216	61.70440	
AFFX-BioB-M_at	64.6066	40.3068	41.82090	46.1087	49.41220	

4.2 Phenotypic data

Phenotypic data summarizes information about the samples (e.g., sex, age, and treatment status; referred to as ‘covariates’). The information describing the samples can be represented as a table with S rows and V columns, where V is the number of covariates. An example of phenotypic data can be input with

```

> pDataFile <- file.path(dataDirectory, "pData.txt")
> pData <- read.table(pDataFile, row.names = 1, header = TRUE,
+   sep = "\t")
> dim(pData)

[1] 26  3

> rownames(pData)

[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"
[16] "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

> summary(pData)

      gender      type      score
Female:11  Case    :15  Min.    :0.1000
Male   :15  Control:11  1st Qu.:0.3275
                        Median :0.4150
                        Mean    :0.5369
                        3rd Qu.:0.7650
                        Max.    :0.9800

```

There are three columns of data, and 26 rows. Note that the number of rows of phenotypic data match the number of columns of expression data, and indeed that the row and column names are identically ordered:

```

> all(rownames(pData) == colnames(exprs))

```

```

[1] TRUE

```

This is an essential feature of the relationship between the assay and phenotype data; *ExpressionSet* will complain if these names do not match.

Phenotypic data can take on a number of different forms. For instance, some covariates might reasonably be represented as numeric values. Other covariates (e.g., gender, tissue type, or cancer status) might better be represented as **factor** objects (see the help page for **factor** for more information). It is especially important that the phenotypic data are encoded correctly; the `colClasses` argument to `read.table` can be helpful in correctly inputting (and ignoring, if desired) columns from the file.

Exercise 2

What class does `read.table` return?

Exercise 3

Determine the column names of `pData`. Hint: `apropos("name")`.

```
> names(pData)

[1] "gender" "type"  "score"
```

Exercise 4

Use `sapply` to determine the classes of each column of `pData`. Hint: read the help page for `sapply`.

```
> sapply(pData, class)

gender      type      score
"factor"   "factor" "numeric"
```

Exercise 5

What is the sex and Case/Control status of the 15th and 20th samples? And for the sample(s) with `score` greater than 0.8.

```
> pData[c(15, 20), c("gender", "type")]

gender type
0 Female Case
T Female Case

> pData[pData$score > 0.8, ]

gender      type      score
E Female    Case    0.93
G  Male     Case    0.96
X  Male     Control  0.98
Y Female    Case    0.94
```

Investigators often find that the meaning of simple column names does not provide enough information about the covariate – What is the cryptic name supposed to represent? What units are the covariates measured in? We can create a data frame containing such meta-data (or read the information from a file using `read.table`) with

```
> metadata <- data.frame(labelDescription = c("Patient gender",
+      "Case/control status", "Tumor progress on XYZ scale"),
+      row.names = c("gender", "type", "score"))
```

This creates a *data.frame* object with a single column called `labelDescription`, and with row names identical to the column names of the *data.frame* containing the phenotypic data. The column `labelDescription` *must* be present; other columns are optional.

Bioconductor's Biobase package provides a class called *AnnotatedDataFrame* that conveniently stores and manipulates the phenotypic data and its metadata in a coordinated fashion. Create and view an *AnnotatedDataFrame* instance with:

```
> phenoData <- new("AnnotatedDataFrame", data = pData,
+   varMetadata = metadata)
> phenoData
```

```
An object of class "AnnotatedDataFrame"
rowNames: A, B, ..., Z (26 total)
varLabels and varMetadata description:
  gender: Patient gender
  type: Case/control status
  score: Tumor progress on XYZ scale
```

Some useful operations on an *AnnotatedDataFrame* include `sampleNames`, `pData` (to extract the original `pData data.frame`), and `varMetadata`. In addition, *AnnotatedDataFrame* objects can be subset much like a *data.frame*:

```
> head(pData(phenoData))
```

```
  gender    type score
A Female Control  0.75
B  Male    Case   0.40
C  Male Control  0.73
D  Male    Case   0.42
E Female    Case   0.93
F  Male Control  0.22
```

```
> phenoData[c("A", "Z"), "gender"]
```

```
An object of class "AnnotatedDataFrame"
rowNames: A, Z
varLabels and varMetadata description:
  gender: Patient gender
```

```
> pData(phenoData[phenoData$score > 0.8, ])
```

```
  gender    type score
E Female    Case   0.93
G  Male    Case   0.96
X  Male Control  0.98
Y Female    Case   0.94
```

4.3 Annotations and feature data

Meta-data on features is as important as meta-data on samples, and can be very large and diverse. A single chip design (i.e., collection of features) is likely to be used in many different experiments, and it would be inefficient to repeatedly collect and coordinate the same meta-data for each *ExpressionSet* instance. Instead, the idea is to construct specialized meta-data packages for each type of chip or instrument. Many of these packages are available from the Bioconductor web site. These packages contain information such as the gene name, symbol and chromosomal location. There are other meta-data packages that contain the information that is provided by other initiatives such as GO and KEGG. The *annotate* package provides basic data manipulation tools for the meta-data packages.

The appropriate way to create annotation data for features is very straight-forward: we provide a character string identifying the type of chip used in the experiment. For instance, the data we are using is from the Affymetrix hgu95av2 chip:

```
> annotation <- "hgu95av2"
```

It is also possible to record information about features that are unique to the experiment (e.g., flagging particularly relevant features). This is done by creating or modifying an *AnnotatedDataFrame* like that for *phenoData* but with row names of the *AnnotatedDataFrame* matching rows of the assay data.

4.4 Experiment description

Basic description about the experiment (e.g., the investigator or lab where the experiment was done, an overall title, and other notes) can be recorded by creating a *MIAME* object. One way to create a *MIAME* object is to use the `new` function:

```
> experimentData <- new("MIAME", name = "Pierre Fermat",
+   lab = "Francis Galton Lab", contact = "pfermat@lab.not.exist",
+   title = "Smoking-Cancer Experiment", abstract = "An example ExpressionSet",
+   url = "www.lab.not.exist", other = list(notes = "Created from text files"))
```

Usually, `new` takes as arguments the class name and pairs of names and values corresponding to different slots in the class; consult the help page for *MIAME* for details of available slots.

4.5 Assembling an *ExpressionSet*

An *ExpressionSet* object is created by assembling its component parts, and after all this work the final assembly is disappointingly easy:

```
> exampleSet <- new("ExpressionSet", exprs = exprs,
+   phenoData = phenoData, experimentData = experimentData,
+   annotation = "hgu95av2")
```

Note that the names on the right of each equal sign can refer to any object of appropriate class for the argument. See the help page for *ExpressionSet* for more information.

We created a rich data object to coordinate diverse sources of information. Less rich objects can be created by providing less information. A minimal expression set can be created with

```
> minimalSet <- new("ExpressionSet", exprs = exprs)
```

Of course this object has no information about phenotypic or feature data, or about the chip used for the assay.

5 *ExpressionSet* Basics

Now that you have an *ExpressionSet* instance, let's explore some of the basic operations. You can get an overview of the structure and available methods for *ExpressionSet* objects by reading the help page:

```
> help("ExpressionSet-class")
```

When you print an *ExpressionSet* object, a brief summary of the contents of the object is displayed (displaying the entire object would fill your screen with numbers):

```
> exampleSet
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 500 features, 26 samples
  element names: exprs
phenoData
  sampleNames: A, B, ..., Z (26 total)
  varLabels and varMetadata description:
    gender: Patient gender
    type: Case/control status
    score: Tumor progress on XYZ scale
featureData
  featureNames: AFFX-MurIL2_at, AFFX-MurIL10_at, ..., 31739_at (500 total)
  fvarLabels and fvarMetadata description: none
experimentData: use 'experimentData(object)'
Annotation: hgu95av2
```

5.1 Accessing Data Elements

A number of accessor functions are available to extract data from an *ExpressionSet* instance. You can access the columns of the phenotype data (an *AnnotatedDataFrame* instance) using `$`:

```
> exampleSet$gender[1:5]

[1] Female Male   Male   Male   Female
Levels: Female Male
```

```
> exampleSet$gender[1:5] == "Female"

[1] TRUE FALSE FALSE FALSE TRUE
```

You can retrieve the names of the features using `featureNames`. For many microarray datasets, the feature names are the probe set identifiers.

```
> featureNames(exampleSet)[1:5]

[1] "AFFX-MurIL2_at" "AFFX-MurIL10_at" "AFFX-MurIL4_at"
[4] "AFFX-MurFAS_at" "AFFX-BioB-5_at"
```

The unique identifiers of the samples in the data set are available via the `sampleNames` method. The `varLabels` method lists the column names of the phenotype data:

```
> sampleNames(exampleSet)[1:5]

[1] "A" "B" "C" "D" "E"

> varLabels(exampleSet)

[1] "gender" "type" "score"
```

Extract the expression matrix and the *AnnotatedDataFrame* of sample information using `exprs` and `phenoData`, respectively:

```
> mat <- exprs(exampleSet)
> dim(mat)

[1] 500 26

> adf <- phenoData(exampleSet)
> adf
```

```
An object of class "AnnotatedDataFrame"
 sampleNames: A, B, ..., Z (26 total)
 varLabels and varMetadata description:
  gender: Patient gender
  type: Case/control status
  score: Tumor progress on XYZ scale
```

5.1.1 Subsetting

Probably the most useful operation to perform on *ExpressionSet* objects is subsetting. Subsetting an *ExpressionSet* is very similar to subsetting the expression matrix that is contained within the *ExpressionSet*, the first argument subsets the features and the second argument subsets the samples. Here are some examples: Create a new *ExpressionSet* consisting of the 5 features and the first 3 samples:

```
> vv <- exampleSet[1:5, 1:3]
> dim(vv)

Features  Samples
         5         3

> featureNames(vv)

[1] "AFFX-MurIL2_at" "AFFX-MurIL10_at" "AFFX-MurIL4_at"
[4] "AFFX-MurFAS_at" "AFFX-BioB-5_at"

> sampleNames(vv)

[1] "A" "B" "C"
```

Create a subset consisting of only the male samples:

```
> males <- exampleSet[, exampleSet$gender == "Male"]
> males
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 500 features, 15 samples
  element names: exprs
phenoData
  sampleNames: B, C, ..., X (15 total)
  varLabels and varMetadata description:
    gender: Patient gender
    type: Case/control status
    score: Tumor progress on XYZ scale
featureData
  featureNames: AFFX-MurIL2_at, AFFX-MurIL10_at, ..., 31739_at (500 total)
  fvarLabels and fvarMetadata description: none
experimentData: use 'experimentData(object)'
Annotation: hgu95av2
```

6 What was used to create this document

The version number of R and the packages and their versions that were used to generate this document are listed below.

- R version 2.6.1 (2007-11-26), x86_64-unknown-linux-gnu
- Locale: LC_CTYPE=en_US;LC_NUMERIC=C;LC_TIME=en_US;LC_COLLATE=en_US;LC_MONETARY=en_US
- Base packages: base, datasets, graphics, grDevices, methods, stats, tools, utils
- Other packages: Biobase 1.16.3