

# eSet structures, August 2005

VJ Carey <stvjc@channing.harvard.edu>

October 12, 2005

## Contents

<b>1 Preliminaries</b>	<b>1</b>
<b>2 Introduction</b>	<b>2</b>
<b>3 Annotated dataset class</b>	<b>4</b>
<b>4 Example</b>	<b>4</b>
<b>5 Validity conditions</b>	<b>6</b>
<b>6 Environments for assay results</b>	<b>8</b>
<b>7 phenoData metadata</b>	<b>9</b>
<b>8 A suggestion for a basic history mechanism</b>	<b>10</b>
8.1 Open question . . . . .	11
8.1.1 Seth's comments . . . . .	11
<b>9 Session Information</b>	<b>12</b>

## 1 Preliminaries

There is a need to expand the capabilities of *exprSet*, the primary multiassay container class in Biobase:

- we should have the ability to have multiple matrices of assay results, where rows correspond to reporters, columns correspond to samples
- we should have the ability to have fairly rich metadata content for both samples and reporters

- we should have the ability to make use of pass-by-reference performance with large assay data structures

We are open to other concerns about container design and would like developers to consider the following questions:

- what sorts of reporter data structures and functionalities would be of interest? Use cases would be helpful
- are there compelling concepts of auditability of these structures that you would like to address? I added a character slot ‘history’ in which one can place a vector of strings, capturing for example results of `match.call()`
- how should we handle `geneNames`, `reporterNames`, `sampleNames` data? Currently I have independent slots for `reporterNames` and `sampleNames`, but this information could be stored as `rownames/colnames` of the `reporterInfo` and `phenoData` structures.

Please discuss openly on the bioc-devel list.

## 2 Introduction

The high-throughput information that we wish to work with has the following general structure.

- There are  $N$  samples that have been assayed. Sample-level metadata are collected roughly uniformly for all samples, and take the form of  $q$ -vectors of mixed continuous and categorical variables for each sample.
- There are  $G$  reporters on which assay values are obtained.  $G$  can be as large as  $10^7$ . Reporter names come from a vocabulary that can be translated into information about biological systems.
- There are various forms of experiment-level metadata that are collected.

In Bioconductor 1.6, the *exprSet* class managed information of the type described above, with some restrictions. In an *exprSet* instance, the `exprs` slot is intended to hold a  $G \times N$  matrix of numerical assay results. An `se.exprs` slot has the same dimensions and structure. The column names and row names of the `exprs` slot provide sample identification and reporter identification. Experiment-level metadata can be contained in slots called `description`, `annotation` and `notes`. Sample level data are collected in a structure of class `phenoData`. This has a slot called `pData` containing an  $N \times q$  data frame, a list of `varLabels` that map attribute names to longer descriptive labels, and a data frame called `varMetadata` which can capture variable-level information such as units of measurement.

The *eSet* class is a provisional implementation of an extended container concept. The current definition is:

```
> library(Biobase)
```

```
> getClass("eSet")
```

Slots:

Name:	assayData	sampleNames	reporterNames
Class:	listOrEnv	character	character
Name:	description	notes	annotation
Class:	MIAME	character	character
Name:	history	reporterInfo	phenoData
Class:	character	data.frameOrNULL	phenoData

Extends: "annotatedDataset"

```
> package.version("Biobase")
```

```
[1] "1.8.0"
```

The *listOrEnv* virtual class confers flexibility on the assay data container component.

```
> getClass("listOrEnv")
```

Extended class definition ( "ClassUnionRepresentation" )  
Virtual Class

No Slots, prototype of class "NULL"

Known Subclasses: "list", "environment"

The *assayData* slot can be occupied by either a list or an environment. This allows accommodation of multiple  $G \times N$  matrices representing assay data. Environments can be used to obtain pass-by-reference semantics. Both list and environment representations for assay data relax structural restrictions that exist with an *exprSet* instance.

### 3 Annotated dataset class

One superclass from which *eSet* inherits is *annotatedDataset*.

```
> getClass("annotatedDataset")
```

Virtual Class

Slots:

Name:	reporterInfo	phenoData
Class:	data.frameOrNULL	phenoData

Known Subclasses: "eSet", "exprSet"

A variety of operators such as `[[` and `$` are defined for objects of classes that extend *annotatedDataset*. See the help page on this class.

A key formal container conferred by *annotatedDataset* inheritance is *phenoData*.

```
> getClass("phenoData")
```

Slots:

Name:	pData	varLabels	varMetadata
Class:	data.frame	list	data.frame

The *phenoData* slots are **pData**, the  $N \times q$  data frame holding sample level non-assay information, **varLabels**, a named list of  $q$  variable labels, and **varMetadata**, a data frame of arbitrary dimensions that provides additional information on *phenoData* variables. See section 7 below for an example.

In addition to the *phenoData* container, we now allow a **reporterInfo** slot to hold a data.frame of  $G$  rows with metadata about reporters. The intention is that columns in **reporterInfo** will likely hold factors to facilitate splitting up the reporter set, into control and active reporters, for example.

### 4 Example

We include a coercion tool, and can thus use the Golub data to illustrate. We will add contrived information on standard errors and presence calls to illustrate the flexibility.

```
> data(golubMergeSub)
> gmes <- as(golubMergeSub, "eSet")
> assayData(gmes)[["se.exprs"]] <- runif(length(exprs(gmes)),
```

```

+      1.5, 2.5) * assayData(gmes)[["exprs"]]
> ratpred <- (assayData(gmes)[["exprs"]]/assayData(gmes)[["se.exprs"]]) >
+      2)
> assayData(gmes)[["presence"]] <- ratpred
> gmes

```

instance of eSet

assayData component is of class list

dimensions of the assayData components:

```

      exprs se.exprs presence
nrow 1000      1000      1000
ncol   72         72         72

```

phenoData object with 11 variables and 72 cases

varLabels

```

Samples: Sample index
ALL.AML: Factor, indicating ALL or AML
BM.PB: Factor, sample from marrow or peripheral blood
T.B.cell: Factor, T cell or B cell leuk.
FAB: Factor, FAB classification
Date: Date sample obtained
Gender: Factor, gender of patient
pctBlasts: pct of cells that are blasts
Treatment: response to treatment
PS: Prediction strength
Source: Source of sample

```

first reporterNames:

```

[1] "L37378_at" "L37792_at" "L37868_s_at" "L37882_at"
[5] "L37936_at"

```

first sampleNames:

```

[1] "39" "40" "42" "47" "48"

```

Note that the show method is slightly elaborated to cope with the added potential content of the assayData component. The `exprs` method still works, provided that an element of the assayData slot has name `exprs`.

```

> dim(exprs(gmes))

```

```

[1] 1000 72

```

The use of `exprs` as an accessor is permitted but does a bit more than the analogous accessor for *exprSet* objects. Assay data are, in this example, contained in a named list of matrices.

```

> dim(assayData(gmes)[["exprs"]])

```

```
[1] 1000 72
```

Note the interpretation of dim:

```
> dim(gmes)
```

```
      exprs se.exprs presence
nrow 1000      1000      1000
ncol  72        72        72
```

Subsetting operations perform as expected:

```
> gmes[1:10, 1:10]
```

```
instance of eSet
```

```
assayData component is of class list
```

```
dimensions of the assayData components:
```

```
      exprs se.exprs presence
nrow   10        10        10
ncol   10        10        10
```

```
phenoData object with 11 variables and 10 cases
```

```
varLabels
```

```
Samples: Sample index
```

```
ALL.AML: Factor, indicating ALL or AML
```

```
BM.PB: Factor, sample from marrow or peripheral blood
```

```
T.B.cell: Factor, T cell or B cell leuk.
```

```
FAB: Factor, FAB classification
```

```
Date: Date sample obtained
```

```
Gender: Factor, gender of patient
```

```
pctBlasts: pct of cells that are blasts
```

```
Treatment: response to treatment
```

```
PS: Prediction strength
```

```
Source: Source of sample
```

```
first reporterNames:
```

```
[1] "L37378_at" "L37792_at" "L37868_s_at" "L37882_at"
```

```
[5] "L37936_at"
```

```
first sampleNames:
```

```
[1] "39" "40" "42" "47" "48"
```

## 5 Validity conditions

```
> getValidity(getClass("eSet"))
```

```
function (object)
validEset(object)
<environment: namespace:Biobase>
```

I would like to have a constraint on reporterNames length but more discussion is needed.

Let's illustrate the use of the reporterInfo construct. Suppose we wish to make it convenient to bind the gene symbols with the data.

```
> data(bbsym)
> gs <- unlist(mget(reporterNames(gmes), bbsym))
> gs <- data.frame(name = gs)
> rownames(gs) <- reporterNames(gmes)
> reporterInfo(gmes) <- gs
```

Now let's restrict our eSet to those reporters for which symbols contain the substring NFK.

```
> gmes[grep("NFK", as.character(reporterInfo(gmes)$name)),
+      ]
```

instance of eSet

assayData component is of class list

dimensions of the assayData components:

	exprs	se.exprs	presence
nrow	3	3	3
ncol	72	72	72

phenoData object with 11 variables and 72 cases

varLabels

```
Samples: Sample index
ALL.AML: Factor, indicating ALL or AML
BM.PB: Factor, sample from marrow or peripheral blood
T.B.cell: Factor, T cell or B cell leuk.
FAB: Factor, FAB classification
Date: Date sample obtained
Gender: Factor, gender of patient
pctBlasts: pct of cells that are blasts
Treatment: response to treatment
PS: Prediction strength
Source: Source of sample
```

first reporterNames:

```
[1] "L40407_at" "M58603_at" "M69043_at"
```

first sampleNames:

```
[1] "39" "40" "42"
```

## 6 Environments for assay results

It is permissible to employ an environment containing matrices as the assay data container. Here we construct the *eSet* instance explicitly, notice the direct containment of sample names and reporter names in slots.

```
> ee <- new.env()
> assign("exprs", exprs(golubMergeSub), ee)
> gme <- new("eSet", assayData = ee, phenoData = phenoData(golubMergeSub),
+   sampleNames = sampleNames(golubMergeSub),
+   reporterNames = geneNames(golubMergeSub))
> gme
```

instance of eSet

assayData component is of class environment

dimensions of the assayData components:

    exprs

nrow 1000

ncol 72

    phenoData object with 11 variables and 72 cases

    varLabels

        Samples: Sample index

        ALL.AML: Factor, indicating ALL or AML

        BM.PB: Factor, sample from marrow or peripheral blood

        T.B.cell: Factor, T cell or B cell leuk.

        FAB: Factor, FAB classification

        Date: Date sample obtained

        Gender: Factor, gender of patient

        pctBlasts: pct of cells that are blasts

        Treatment: response to treatment

        PS: Prediction strength

        Source: Source of sample

first reporterNames:

[1] "L37378\_at" "L37792\_at" "L37868\_s\_at" "L37882\_at"

[5] "L37936\_at"

first sampleNames:

[1] "39" "40" "42" "47" "48"

Subsetting works as expected, and the `exprs` or `assayData` extractors can be used.

```
> gme[1:3, 1:4]
```

instance of eSet

assayData component is of class environment



```

dimensions of the assayData components:
  exprs
nrow    3
ncol    4
  phenoData object with 11 variables and 4 cases
  varLabels
    Samples: Sample index
    ALL.AML: Factor, indicating ALL or AML
    BM.PB: Factor, sample from marrow or peripheral blood
    T.B.cell: Factor, T cell or B cell leuk.
    FAB: Factor, FAB classification
    Date: Date sample obtained
    Gender: Factor, gender of patient
    pctBlasts: pct of cells that are blasts
    Treatment: response to treatment
    PS: Prediction strength
    Source: Source of sample
first reporterNames:
[1] "L37378_at" "L37792_at" "L37868_s_at"
first sampleNames:
[1] "39" "40" "42"

> dim(exprs(gme))

[1] 1000  72

```

## 7 phenoData metadata

The `varMetadata` slot of *phenoData* helps to encode units or other important features of *phenoData* variables.

```
> pes <- phenoData(gme)
```

It is necessary to initialize the metadata-enabled *phenoData* structure.

```
> pes <- convertVarLabels(pes)
```

Then metadata can be added field by field.

```

> pes <- addVarMetadataEntry(pes, "PS", "units",
+   "pct/100")
> phenoData(gme) <- pes
> getUnits(phenoData(gme), "PS")

```

```
[1] pct/100
Levels: pct/100
```

It would be possible to have a reporter metadata structure as well. This could be used for preserving information on, e.g., the version of the mapping between reporter IDs and gene symbols. Before adding this feature I want to hear reactions from the community.

## 8 A suggestion for a basic history mechanism

It is desirable that an *eSet* object contains some basic information about itself. Note that the goal of such a history mechanism is less than complete reproducibility: for this we have the vignettes (Sweave) technology.

Currently, preprocessing functions in the *affy* and *vs*n packages already do add some information on what they did to their returned *exprSet* objects. For example, see the last dozen lines in the function *vs*n in the file *vs*n/R/*vs*n.R of the package *vs*n. Here is a toy example for an *eSet* object:

```
> setGeneric("wonderfulPreproc", function(x, lambda) standardGeneric("wonderfulPreproc"))
> setMethod("wonderfulPreproc", c("eSet", "numeric"),
+   function(x, lambda) {
+     y = assayData(x)$exprs
+     gamma = mean(log(diag(t(y) %*% y)))
+     assayData(x)$exprs = 1/(1 + (y^lambda)/gamma)
+     cmt = list(comment = paste("Processed by wonderful processing method on",
+       date()), lambda = lambda, gamma = gamma)
+     x@description@preprocessing = append(x@description@preprocessing,
+       cmt)
+     return(x)
+   })
> Pgme = wonderfulPreproc(gme, pi/2)
```

The resulting description slot then looks as follows.

```
> description(Pgme)
```

Experimenter name:

Laboratory:

Contact information:

Title:

URL:

No abstract available.

Information is available on: preprocessing

```
> description(Pgme)@preprocessing

$comment
[1] "Processed by wonderful processing method on Wed Oct 12 20:32:16 2005"

$lambda
[1] 1.570796

$gamma
[1] 22.25384
```

## 8.1 Open question

The mechanism proposed above allows to put arbitrary objects into the list `preprocessing`. In particular, these can then be interrogated and used by subsequent processing functions. For an example, please see the first dozen lines in the function `meanSdPlot` in the file `vsn/R/meanSdPlot.R` of the package `vsn`.

But how does this relate to the `history` slot in the *eSet*, which is simply of class `character`?

### 8.1.1 Seth's comments

I think the `history` slot should be a list. I can see two possible ways to go with this.

Method I: Use automation. Modify every replacement function for *eSet* to optionally accept a `history` argument. The default is to grab the current or parent call (as a *call* object) and append this to the `history` list. The benefit is we get history tracking without any work. It remains to be seen whether this is useful.

Method II: Require intervention by implementors to record history. Here we could define an *CallLoggingObject* class that has a `history` slot (list). The *eSet* class would extend *CallLoggingObject* as would any class wanting to log history. Usage would look like:

```
## record history
logCall(object) <- aCall

## access history
callHistory(object)
```

The big disadvantage, is that users/developers have to call `logCall` or else history is not recorded.

In recent discussions, Robert prefers method I because it doesn't require changes to preprocessing or other code. It should be fairly easy to implement. We can see if the history log it produces is useful and iterate.

## 9 Session Information

The version number of R and packages loaded for generating the vignette were:

```
\begin{itemize}
  \item R version 2.2.0, 2005-10-10, \verb|i386-pc-mingw32|
  \item Base packages: base, datasets, graphics,
    grDevices, methods, stats, tools, utils
  \item Other packages: Biobase~1.8.0
\end{itemize}
```