Introduction to the xps Package: Classes structure

Christian Stratowa

February, 2009

Contents

1 Introduction

This document provides a tutorial on the class structures used in the `xps` package. The `xps` package provides S4 class definitions and associated methods for raw intensity data, preprocessed intensity data, expression measures, and detection call data for batches of Affymetrix expression or exon arrays. Detailed information on functions, classes and methods can be obtained in the help files.

Since the main purpose of the S4 classes is to provide a link to and allow access to `ROOT` files and the `ROOT` trees contained therein, it is important to understand the meaning of `ROOT` files and the `ROOT` trees first.

2 XPS and ROOT

Package `xps` is based on two powerful frameworks, namely `R` and `ROOT`. `ROOT`(`http://root.cern.ch`) is an object-oriented C++ framework that has been developed at CERN for distributed data warehousing and data mining of particle data in the petabyte range. Data are stored as sets of objects in machine-independent files, and a specialized class, called a `ROOT` tree, has been designed to store large quantities of data, see The ROOT team (2007).

2.1 ROOT File

A ROOT file, C++ class *TFile*, is like a UNIX file directory. It can contain directories and objects organized in unlimited number of levels. It is also stored in machine independent format. Thus it can be considered to be the `ROOT` equivalent of an `R environment`. Just like an `R environment` a `ROOT` file can be copied to any system directory and used from within any OS.

A `ROOT` file can store any C++ object like `ROOT` trees, but also any other object such as a text file. For our purpose it is only important to know that `ROOT` trees can be stored in `ROOT` files.

2.2 ROOT Tree

`ROOT` trees, C++ class *TTree*, have been especially designed to store large quantities of same-class objects. As simplest case, a `ROOT` tree can store data imported from tables, and thus can be considered to be the `ROOT` equivalent of an `R data.frame`. A `ROOT` tree consists of branches and leaves, with each leaf containing the data from one column of the imported data table.

The *TTree* class is optimized to reduce disk space and enhance access speed. When using a TTree, you fill its branch buffers with leaf data and the buffers are written to disk when it is full. Branches, buffers, and leaves are explained in: The ROOT team (2007); here it is important to realize that each object is not written individually, but rather collected and written a bunch at a time. This is where *TTree* takes advantage of compression and will produce a much smaller file than if the objects were written individually.

3 XPS S4 classes

Taking advantage of the features described above, package `xps` stores all data as `ROOT` trees in portable `ROOT` files. The four main classes for Affymetrix oligonucleotide array data are:

`SchemeTreeSet`: Data describing microarray layout, probe information and metadata for genes are stored as `ROOT` trees in `ROOT` *scheme* files, accessible from `R` as S4 class *SchemeTreeSet* objects.

`DataTreeSet`: This class provides the link to `ROOT` data files and the `ROOT` trees contained therein. Data files contain the content of the raw CEL-files as `ROOT` trees. Additional data files can contain the background-corrected intensities as well as the computed background intensities.

`ExprTreeSet`: Preprocessing of raw data, i.e. summarization and normalization, results in conversion of probe level data to expression values. These data are stored as `ROOT` trees in `ROOT` expression files. Class *ExprTreeSet* provides the link to the `ROOT` expression file and the `ROOT` trees contained therein.

`CallTreeSet`: Results of MAS5 detection calls or DABG calls, respectively, and the corresponding p-values are stored as `ROOT` trees in `ROOT` call files, accessible via class *CallTreeSet*.

All of these S4 classes are derived from the same virtual base class *TreeSet*.

The following classes are relevant for filtering and analysis of expression values:

`PreFilter`: This class allows the initialization of different non–specific filters and stores the relevant parameters of each filter.

`UniFilter`: This class allows the initialization of different univariate filters and stores the relevant parameters of each filter..

`FilterTreeSet`: This class provides a link between class *ExprTreeSet*, the filter class applied to this *ExprTreeSet* and the resulting `ROOT` filter file and the `ROOT` mask tree contained therein.

`AnalysisTreeSet`: Univariate analysis allows the selection of differentially expressed genes. This class provides a link between class *FilterTreeSet* containing the relevant filtering information, and the resulting `ROOT` filter file and the `ROOT` trees contained therein.

The S4 classes *PreFilter* and *UniFilter* are derived from base class *Filter* while S4 classes *FilterTreeSet* and *AnalysisTreeSet* are derived from the virtual base class *TreeSet*.

An optional S4 class, called *ProjectInfo*, can be used to describe the relevant phenotypic and MIAME-like project information.

3.1 `TreeSet` class

This is the virtual base class for all other S4 classes and provides the link to the `ROOT` file and the `ROOT` trees contained therein for the corresponding subclass.

Function `getClassDef` can be used to get information on the slots and the inheritance information:

```
> library(xps)

> getClassDef("TreeSet")

Virtual Class âĂIJTreeSetâĂ■

Slots:

Name:    setname   settype  rootfile   filedir  numtrees treenames
Class: character character character character   numeric      list

Known Subclasses:
Class "SchemeTreeSet", directly
Class "ProcesSet", directly
Class "DataTreeSet", by class "ProcesSet", distance 2
Class "ExprTreeSet", by class "ProcesSet", distance 2
Class "CallTreeSet", by class "ProcesSet", distance 2
Class "FilterTreeSet", by class "ProcesSet", distance 2
Class "AnalysisTreeSet", by class "ProcesSet", distance 2
```

The meaning of the slots is as follows:

`setname`: Object of class "character", representing the name to the `ROOT` file subdirectory where the `ROOT` trees are stored, usually one of *DataTreeSet*, *PreprocesSet*, *CallTreeSet*.

`settype`: Object of class "character", describing the type of treeset stored in `setname`, usually one of `scheme`, `rawdata`, `preprocess`.

`rootfile`: Object of class "character", representing the name of the `ROOT` file, including the full path.

`filedir`: Object of class "character", describing the full path to the system directory where `rootfile` is stored.

`numtrees`: Object of class "numeric", representing the number of `ROOT` trees stored in subdirectoy `setname`.

`treenames`: Object of class "list", representing the names of the `ROOT` trees stored in subdirectoy `setname`.

3.2 SchemeTreeSet class

This class extends class *TreeSet* and allows access to the ROOT scheme trees, probe trees and annotation trees. Furthermore, it contains certain array information (slot probeinfo) and allows access to the array mask (slot mask), describing the probe type.

```
> getClassDef("SchemeTreeSet")

Class âĂIJSchemeTreeSetâĂ■

Slots:

Name:    chipname   chiptype  probeinfo       mask    setname    settype
Class:  character  character       list data.frame  character  character

Name:    rootfile    filedir   numtrees  treenames
Class:  character  character    numeric       list

Extends: "TreeSet"
```

In addition to the slots of *TreeSet* it contains the following slots:

chipname: Object of class "character", representing the Affymetrix chip name.

chiptype: Object of class "character", representing the chip tpye, either "GeneChip" or "ExonChip".

probeinfo: Object of class "list", representing chip information, including nrows, ncols, number of probes, etc.

mask: Object of class "data.frame". The data.frame can contain the mask used to identify the probes as e.g. PM, MM, or control probes.

As an example, scheme.test3, described in Vignette "xps.pdf" contains the information for the Affymetrix 'Test3' expression array:

```
> scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root",
+     sep = "/"))
> str(scheme.test3)

Formal class 'SchemeTreeSet' [package "xps"] with 10 slots
  ..@ chipname : chr "Test3"
  ..@ chiptype : chr "GeneChip"
  ..@ probeinfo:List of 8
  .. ..$ nrows     : int 126
  .. ..$ ncols     : int 126
  .. ..$ nprobes   : int 14552
  .. ..$ ncontrols : int 13
  .. ..$ ngenes    : int 345
  .. ..$ nunits    : int 358
  .. ..$ nprobesets: int 358
  .. ..$ naffx     : int 0
  ..@ mask     :'data.frame':        0 obs. of  0 variables
  ..@ setname  : chr "Test3"
  ..@ settype  : chr "scheme"
```

```
  ..@ rootfile : chr "/tmp/Rinst53431486/xps/schemes/SchemeTest3.root"
  ..@ filedir  : chr "/tmp/Rinst53431486/xps/schemes"
  ..@ numtrees : int 4
  ..@ treenames:List of 4
  .. ..$ : chr "Test3.idx"
  .. ..$ : chr "Test3.scm"
  .. ..$ : chr "Test3.prb"
  .. ..$ : chr "Test3.ann"
```

As this example demonstrates, the `ROOT` scheme file contains 4 `ROOT` trees, storing all necessary GeneChip information, including the annotation. All trees can be exported as tables, and optionally also imported as `data.frame`.

The 'Unit' tree with extension *idx* contains information about the processing units, i.e. the Affymetrix probesets, such as an internal 'UNIT_ID', probeset names and number of cells:

```
> idx <- export(scheme.test3, treetype = "idx", outfile = "Test3_idx.txt",
+     as.dataframe = TRUE, verbose = FALSE)
> idx[10:16, ]
```

```
   UNIT_ID         UnitName NumCells NumAtoms UnitType
10      -4             QC10      161        1       -5
11      -3             QC11      128        1      -12
12      -2             QC12      128        1      -11
13      -1             QC13        9        1      -13
14       0 Pae_16SrRNA_s_at       32       16        3
15       1 Pae_23SrRNA_s_at       32       16        3
16       2   PA1178_oprH_at       24       12        3
```

The 'Scheme' tree with extension *scm* provides the chip layout for the different probesets, i.e. the (x,y)-coordinates for each probeset with a defined 'UNIT_ID':

```
> scm <- export(scheme.test3, treetype = "scm", outfile = "Test3_scm.txt",
+     as.dataframe = TRUE, verbose = FALSE)
> scm[1843:1848, ]
```

```
     UNIT_ID   X  Y ProbeLength Mask Atom ProbeBase TargetBase
1843      -1  62 63          25   -1    0         N          N
1844      -1  62 64          25   -1    0         N          N
1845       0 111 80          25    0    0         A          A
1846       0 111 79          25    1    0         T          A
1847       0  19 19          25    1    1         A          T
1848       0  19 20          25    0    1         T          T
```

As shown, this tree contains also a column (leaf) 'Mask', which assigns the probe type to each probe on the array. For expression arrays, probe types are PM with `msk=1`, MM with `msk=0`, and control probes with `msk=-1`. 'Mask' for exon arrays will be described elsewhere.

As default, slot `mask` is empty to save memory (especially when using exon arrays). It is easy to add the 'Mask' to slot `mask` (and remove it, too):

```
> scheme.test3 <- attachMask(scheme.test3)
> msk <- chipMask(scheme.test3)
> scheme.test3 <- removeMask(scheme.test3)
> msk[1843:1848, ]
```

```
     UNIT_ID   X  Y Mask
1843      -1  62 63   -1
1844      -1  62 64   -1
1845       0 111 80    0
1846       0 111 79    1
1847       0  19 19    1
1848       0  19 20    0
```

The 'Probe' tree with extension *prb* contains the nucleotide sequence for each PM oligonucleotide, as well as the GC-content and the calculated melting temperature Tm:

```
> prb <- export(scheme.test3, treetype = "prb", outfile = "Test3_prb.txt",
+     as.dataframe = TRUE, verbose = FALSE)
> prb[1843:1848, ]
```

```
     ProbeSetID ProbeX ProbeY ProbeInterrogationPosition
1843         -1     62     63                         -1
1844         -1     62     64                         -1
1845          0    111     80                         -1
1846          0    111     79                        128
1847          0     19     19                        174
1848          0     19     20                         -1
                 ProbeSequence ContentGC Tm ProbeType
1843                         N        -1 -1        -1
1844                         N        -1 -1        -1
1845                         N        14 73         1
1846 TGCCTAGGAATCTGCCTGGTAGTGG        14 73         1
1847 CGCTAATACCGCATACGTCCTGAGG        14 73         1
1848                         N        14 73         1
```

Finally, the transcript 'Annotation' tree with extension *ann* contains the probeset annotation, such as gene name and symbol:

```
> ann <- export(scheme.test3, treetype = "ann", outfile = "Test3_ann.txt",
+     as.dataframe = TRUE, verbose = FALSE)
> head(ann)
```

```
  UNIT_ID       ProbesetID
1       0 Pae_16SrRNA_s_at
2       1 Pae_23SrRNA_s_at
3       2   PA1178_oprH_at
4       3   PA1816_dnaQ_at
5       4    PA3183_zwf_at
6       5   PA3640_dnaE_at
                                                                GeneName GeneSymbol
1                                                                   <NA>       <NA>
2                                                                   <NA>       <NA>
3 PhoP/Q and low Mg2+ inducible outer membrane protein H1 precursor       oprH
4                                DNA polymerase III, epsilon chain       dnaQ
5                              glucose-6-phosphate 1-dehydrogenase        zwf
6                                 DNA polymerase III, alpha chain       dnaE
  GeneAccession EntrezID Chromosome Cytoband Start Stop Strand
```

```
1          <NA>       -1         NA     <NA>    -1   -1      ?
2          <NA>       -1         NA     <NA>    -1   -1      ?
3          <NA>       -1         NA     <NA>    -1   -1      ?
4          <NA>       -1         NA     <NA>    -1   -1      ?
5          <NA>       -1         NA     <NA>    -1   -1      ?
6          <NA>       -1         NA     <NA>    -1   -1      ?
```

The ROOT *scheme* files for exon arrays contain additional tree types, see the help file `?validTreeType`.

3.3 ProcesSet class

This is the common base class for subclasses *DataTreeSet*, *ExprTreeSet*, and *CallTreeSet*, respectively, and contains the following slots:

```
> getClassDef("ProcesSet")

Class âĂIJProcesSetâĂ■

Slots:

Name:         scheme           data         params        setname       settype
Class: SchemeTreeSet     data.frame           list      character     character

Name:       rootfile       filedir      numtrees     treenames
Class:     character     character       numeric          list

Extends: "TreeSet"

Known Subclasses: "DataTreeSet", "ExprTreeSet", "CallTreeSet", "FilterTreeSet", "AnalysisTreeSet"
```

The additional slots are:

`scheme`: Object of class "SchemeTreeSet", providing access to the ROOT scheme file.

`data`: Object of class "data.frame". The data.frame can contain the data stored in ROOT data trees.

`params`: Object of class "list" representing relevant parameters.

The content of slot `data` depends on the type of subclass:
For subclass *DataTreeSet* slot `data` is reserved for the probe data, however, it is empty by default to avoid potential memory problems with large datasets. For subclasses *ExprTreeSet* and *CallTreeSet* slot `data` contains expression levels and detection p-values, respectively.

3.4 DataTreeSet class

Class *DataTreeSet* is usually created when importing raw CEL-files as ROOT trees into a ROOT data file, or accessing an existing ROOT data file, using functions `import.data` or `root.data`, respectively. However, certain preprocessing functions such as `bgcorrect` can also result in the creation of class *DataTreeSet*. The slots are:

```
> getClassDef("DataTreeSet")
```

```
Class âĂIJDataTreeSetâĂ▪

Slots:

Name:    bgtreenames          bgrd   projectinfo        scheme          data
Class:          list    data.frame   ProjectInfo SchemeTreeSet    data.frame

Name:         params       setname       settype      rootfile       filedir
Class:          list     character     character     character     character

Name:       numtrees     treenames
Class:       numeric          list

Extends:
Class "ProcesSet", directly
Class "TreeSet", by class "ProcesSet", distance 2
```

Following additional slots are defined for class *DataTreeSet*:

`bgtreenames`: Object of class "list", representing the names of optional `ROOT` background trees.

`bgrd`: Object of class "data.frame". The data.frame can contain background intensities stored in `ROOT` background trees.

`projectinfo`: Object of class "ProjectInfo", containing information about the project.

Slots `bgtreenames` and `bgrd` are reserved for optional background tree names and data, respectively.

Slot `projectinfo` allows to save optional project information defined in class *ProjectInfo* described below. The project information will not only be stored in class *DataTreeSet*, but also in the `ROOT` data file containing the raw data from the CEL-files, when using functions `import.data` or `addData`, respectively.

As an example, `data.test3`, described in Vignette "xps.pdf" contains the information to access the CEL-files from the 'Test3' expression array, which are stored as `ROOT` trees in `ROOT` data file *DataTest3_cel.root*:

```
> data.test3 <- root.data(scheme.test3, paste(.path.package("xps"),
+     "rootdata/DataTest3_cel.root", sep = "/"))
> str(data.test3)

Formal class 'DataTreeSet' [package "xps"] with 12 slots
  ..@ bgtreenames: list()
  ..@ bgrd       :'data.frame':        0 obs. of  0 variables
  ..@ projectinfo:Formal class 'ProjectInfo' [package "xps"] with 15 slots
  .. .. ..@ submitter     : chr ""
  .. .. ..@ laboratory    : chr ""
  .. .. ..@ contact       : chr ""
  .. .. ..@ project       : list()
  .. .. ..@ author        : list()
  .. .. ..@ dataset       : list()
  .. .. ..@ source        : list()
  .. .. ..@ sample        : list()
  .. .. ..@ celline       : list()
  .. .. ..@ primarycell   : list()
```

```
.. .. ..@ tissue        : list()
.. .. ..@ biopsy        : list()
.. .. ..@ arraytype     : list()
.. .. ..@ hybridizations:'data.frame':        0 obs. of  0 variables
.. .. ..@ treatments    :'data.frame':        0 obs. of  0 variables
..@ scheme     :Formal class 'SchemeTreeSet' [package "xps"] with 10 slots
.. .. ..@ chipname : chr "Test3"
.. .. ..@ chiptype : chr "GeneChip"
.. .. ..@ probeinfo:List of 8
.. .. .. ..$ nrows     : int 126
.. .. .. ..$ ncols     : int 126
.. .. .. ..$ nprobes   : int 14552
.. .. .. ..$ ncontrols : int 13
.. .. .. ..$ ngenes    : int 345
.. .. .. ..$ nunits    : int 358
.. .. .. ..$ nprobesets: int 358
.. .. .. ..$ naffx     : int 0
.. .. ..@ mask     :'data.frame':        0 obs. of  0 variables
.. .. ..@ setname  : chr "Test3"
.. .. ..@ settype  : chr "scheme"
.. .. ..@ rootfile : chr "/tmp/Rinst53431486/xps/schemes/SchemeTest3.root"
.. .. ..@ filedir  : chr "/tmp/Rinst53431486/xps/schemes"
.. .. ..@ numtrees : int 4
.. .. ..@ treenames:List of 4
.. .. .. ..$ : chr "Test3.idx"
.. .. .. ..$ : chr "Test3.scm"
.. .. .. ..$ : chr "Test3.prb"
.. .. .. ..$ : chr "Test3.ann"
..@ data       :'data.frame':        0 obs. of  0 variables
..@ params     : list()
..@ setname    : chr "DataSet"
..@ settype    : chr "rawdata"
..@ rootfile   : chr "/tmp/Rinst53431486/xps/rootdata/DataTest3_cel.root"
..@ filedir    : chr "/tmp/Rinst53431486/xps/rootdata"
..@ numtrees   : int 4
..@ treenames  :List of 4
.. ..$ : chr "TestA1.cel"
.. ..$ : chr "TestA2.cel"
.. ..$ : chr "TestB1.cel"
.. ..$ : chr "TestB2.cel"
```

As mentioned, slot `data` is initially empty to avoid memory problems with large datasets, especially exon array datasets. Functions `attachInten` and `removeInten` allow to attach/remove these data, respectively, as described in Vignette "xps.pdf".

3.5 ExprTreeSet class

Class *ExprTreeSet* is obtained as result of preprocessing the raw data, i.e. summarization, or as result of normalizing preprocessed data, respectively. It contains the expression levels in slot `data`. It has the following slots:

```
> getClassDef("ExprTreeSet")
```

```
Class âĂIJExprTreeSetâĂ■

Slots:

Name:       exprtype      normtype        scheme          data        params
Class:     character     character SchemeTreeSet    data.frame          list

Name:        setname       settype      rootfile       filedir      numtrees
Class:     character     character     character     character       numeric

Name:      treenames
Class:          list

Extends:
Class "ProcesSet", directly
Class "TreeSet", by class "ProcesSet", distance 2
```

The additional slots are:

`exprtype`: Object of class "character", representing the exression type, i.e. `rma`, `mas5`, `mas4` or `custom`.

`normtype`: Object of class "character", representing the normalization type, i.e. `none`, `mean`, `median`, `lowess`,`supsmu`.

3.6 `CallTreeSet` class

Class *CallTreeSet* allows to access the Affymetrix detection calls and corresponding p-values, either the MAS5 detection calls or the detection above background (DABG) calls. It is obtained as result of functions `mas5.call` or `dabg.call`, respectively. It has the following slots:

```
> getClassDef("CallTreeSet")

Class âĂIJCallTreeSetâĂ■

Slots:

Name:       calltype       detcall        scheme          data        params
Class:     character    data.frame SchemeTreeSet    data.frame          list

Name:        setname       settype      rootfile       filedir      numtrees
Class:     character     character     character     character       numeric

Name:      treenames
Class:          list

Extends:
Class "ProcesSet", directly
Class "TreeSet", by class "ProcesSet", distance 2
```

The additional slots are:

`calltype`: Object of class "character", representing the call type, i.e. "mas5" or "dabg".

`detcall`: Object of class "data.frame". The data.frame can contain the detection calls stored in `ROOT` call trees.

Slot `data` contains the detection call p-values while slot `detcall` contains the detection calls.

3.7 `ProjectInfo` class

This class is a stand-alone supporting S4 class and allows to store phenotypic and MIAME-like project information. It contains the following slots:

```
> getClassDef("ProjectInfo")

Class âĂIJProjectInfoâĂ▪

Slots:

Name:       submitter     laboratory        contact        project
Class:      character      character      character           list

Name:          author        dataset         source         sample
Class:           list           list           list           list

Name:         celline    primarycell         tissue         biopsy
Class:           list           list           list           list

Name:       arraytype hybridizations     treatments
Class:           list     data.frame     data.frame
```

`submitter`: Object of class "character", representing the name of the submitter.

`laboratory`: Object of class "character", representing the laboratory of the submitter.

`contact`: Object of class "character", representing the contact address of the submitter.

`project`: Object of class "list", representing the project information.

`author`: Object of class "list", representing the author information.

`dataset`: Object of class "list", representing the dataset information.

`source`: Object of class "list", representing the sample source information.

`sample`: Object of class "list", representing the sample information.

`celline`: Object of class "list", representing the sample information for cell lines.

`primarycell`: Object of class "list", representing the sample information for primary cells.

`tissue`: Object of class "list", representing the sample information for tissues.

`biopsy`: Object of class "list", representing the sample information for biopsies.

`arraytype`: Object of class "list", representing the array information.

`hybridizations`: Object of class "data.frame", representing the hybridization information for each hybridization.

`treatments`: Object of class "data.frame", representing the treatment information for each hybridization.

This class must be created explicitely first using the constructor function `ProjectInfo`, before it can be added to class *DataTreeSet*. Here is a simple example for `data.test3`:

```
> project <- ProjectInfo(submitter = "Christian", laboratory = "home",
+     contact = "email")
> projectInfo(project) <- c("TestProject", "20060106", "Project Type",
+     "use Test3 data for testing", "my comment")
> authorInfo(project) <- c("Stratowa", "Christian", "Project Leader",
+     "Company", "Dept", "cstrato.at.aon.at", "++43-1-1234", "my comment")
> datasetInfo(project) <- c("Test3Set", "MC", "Tissue", "Stratowa",
+     "20060106", "description", "my comment")
> sourceInfo(project) <- c("Unknown", "source type", "Homo sapiens",
+     "caucasian", "description", "my comment")
> primcellInfo(project) <- c("Mel31", "primary cell", 20071123,
+     "extracted from patient", "male", "my pheno", "my genotype",
+     "RNA extraction", TRUE, "NMRI", "female", 7, "months", "my comment")
> arrayInfo(project) <- c("Test3", "GeneChip", "description", "my comment")
> hybridizInfo(project) <- c(c("TestA1", "hyb type", "TestA1.CEL",
+     20071117, "my prep1", "standard protocol", "A1", 1, "my comment"),
+     c("TestA2", "hyb type", "TestA2.CEL", 20071117, "my prep2",
+         "standard protocol", "A2", 1, "my comment"), c("TestB1",
+         "hyb type", "TestB1.CEL", 20071117, "my prep1", "standard protocol",
+         "B1", 2, "my comment"), c("TestB2", "hyb type", "TestB2.CEL",
+         20071117, "my prep2", "standard protocol", "B2", 2, "my comment"))
> treatmentInfo(project) <- c(c("TestA1", "DMSO", 4.3, "mM", 1,
+     "hours", "intravenous", "my comment"), c("TestA2", "DMSO",
+     4.3, "mM", 8, "hours", "intravenous", "my comment"), c("TestB1",
+     "DrugA2", 4.3, "mM", 1, "hours", "intravenous", "my comment"),
+     c("TestB2", "DrugA2", 4.3, "mM", 8, "hours", "intravenous",
+         "my comment"))
> show(project)

Project information:
   Submitter:  Christian
   Laboratory: home
   Contact:    email
   Information is available on: project, author, dataset, source, sample, primarycell, arraytype, hybr
```

3.8 `Filter` class

This is the base class for the S4 classes *PreFilter* and *UniFilter*.

```
> getClassDef("Filter")

Class âĂIJFilterâĂ▮

Slots:

Name:  numfilters
```

```
Class:    numeric

Known Subclasses: "PreFilter", "UniFilter"
```

The meaning of the slots is as follows:

`numfilters`: Object of class "numeric" giving the number of filters applied.

3.9 `PreFilter` class

Class *PreFilter* allows to apply different filters to class `ExprTreeSet`, i.e. to the expression level data.frame `data`..

```
> getClassDef("PreFilter")

Class âĂIJPreFilterâĂ■

Slots:

Name:          mad          cv    variance  difference       ratio         gap
Class:        list        list        list        list        list        list

Name:  lothreshold hithreshold    quantile    prescall  numfilters
Class:        list        list        list        list     numeric

Extends: "Filter"
```

The meaning of the slots is as follows:

`mad`:Object of class "list" describing parameters for `madFilter`.

`cv`:Object of class "list" describing parameters for `cvFilter`.

`variance`:Object of class "list" describing parameters for `varFilter`.

`difference`:Object of class "list" describing parameters for `diffFilter`.

`ratio`:Object of class "list" describing parameters for `RratioFilter`.

`gap`:Object of class "list" describing parameters for `RgapFilter`.

`hithreshold`:Object of class "list" describing parameters for `highFilter`.

`lothreshold`:Object of class "list" describing parameters for `lowFilter`.

`quantile`:Object of class "list" describing parameters for `quantileFilter`.

`prescall`:Object of class "list" describing parameters for `callFilter`.

This class must be created explicitely using the constructor function `PreFilter`, and is added as parameter `filter` to function `prefilter`. Here is an example, where all filters are initialized for demonstration purposes only:

```
> prefltr <- PreFilter()
> madFilter(prefltr) <- c(0.5, 0.01)
> cvFilter(prefltr) <- c(0.3, 0, 0.01)
> varFilter(prefltr) <- c(0.6, 0.02, 0.01)
> diffFilter(prefltr) <- c(2.2, 0, 0.01)
> ratioFilter(prefltr) <- c(1.5)
> gapFilter(prefltr) <- c(0.3, 0.05, 0, 0.01)
> lowFilter(prefltr) <- c(4, 3, "samples")
> highFilter(prefltr) <- c(14.5, 75, "percent")
> quantileFilter(prefltr) <- c(3, 0.05, 0.95)
> callFilter(prefltr) <- c(0.02, 80, "percent")
> str(prefltr)

Formal class 'PreFilter' [package "xps"] with 11 slots
  ..@ mad        :List of 2
  .. ..$ cutoff : num 0.5
  .. ..$ epsilon: num 0.01
  ..@ cv         :List of 3
  .. ..$ cutoff : num 0.3
  .. ..$ trim   : num 0
  .. ..$ epsilon: num 0.01
  ..@ variance   :List of 3
  .. ..$ cutoff : num 0.6
  .. ..$ trim   : num 0.02
  .. ..$ epsilon: num 0.01
  ..@ difference :List of 3
  .. ..$ cutoff : num 2.2
  .. ..$ trim   : num 0
  .. ..$ epsilon: num 0.01
  ..@ ratio      :List of 1
  .. ..$ cutoff: num 1.5
  ..@ gap        :List of 4
  .. ..$ cutoff : num 0.3
  .. ..$ window : num 0.05
  .. ..$ trim   : num 0
  .. ..$ epsilon: num 0.01
  ..@ lothreshold:List of 3
  .. ..$ cutoff   : num 4
  .. ..$ parameter: num 3
  .. ..$ condition: chr "samples"
  ..@ hithreshold:List of 3
  .. ..$ cutoff   : num 14.5
  .. ..$ parameter: num 75
  .. ..$ condition: chr "percent"
  ..@ quantile   :List of 3
  .. ..$ cutoff    : num 3
  .. ..$ loquantile: num 0.05
  .. ..$ hiquantile: num 0.95
  ..@ prescall   :List of 3
  .. ..$ cutoff   : num 0.02
  .. ..$ samples  : num 80
```

```
  .. ..$ condition: chr "percent"
  ..@ numfilters : num 10
```

3.10 UniFilter class

Class *UniFilter* allows to apply different unitest filters to class `ExprTreeSet`, i.e. to the expression level data.frame `data`.

```
> getClassDef("UniFilter")

Class âĂIJUniFilterâĂ▪

Slots:

Name:  foldchange   prescall  unifilter    unitest numfilters
Class:       list       list       list       list    numeric

Extends: "Filter"
```

The meaning of the slots is as follows:

`foldchange`:Object of class "list" describing parameters for `fcFilter`.

`prescall`:Object of class "list" describing parameters for `callFilter`.

`unifilter`:Object of class "list" describing parameters for `unitestFilter`.

`unitest`:Object of class "list" describing parameters for `uniTest`.

This class must be created explicitely using the constructor function `UniFilter`, and is added as parameter `filter` to function `unifilter`. Here is an example, where all filters are initialized for demonstration purposes only:

```
> unifltr <- UniFilter()
> fcFilter(unifltr) <- c(1.5, "both")
> callFilter(unifltr) <- c(0.02, 80, "percent")
> unitestFilter(unifltr) <- c(0.01, "pval")
> uniTest(unifltr) <- c("t.test", "two.sided", "wy", 5000, 0, FALSE,
+     0.98, TRUE)
> str(unifltr)

Formal class 'UniFilter' [package "xps"] with 5 slots
  ..@ foldchange:List of 2
  .. ..$ cutoff   : num 1.5
  .. ..$ direction: chr "both"
  ..@ prescall  :List of 3
  .. ..$ cutoff   : num 0.02
  .. ..$ samples  : num 80
  .. ..$ condition: chr "percent"
  ..@ unifilter :List of 2
  .. ..$ cutoff  : num 0.01
  .. ..$ variable: chr "pval"
  ..@ unitest   :List of 8
  .. ..$ type       : chr "t.test"
```

```
  .. ..$ alternative: chr "two.sided"
  .. ..$ correction : chr "wy"
  .. ..$ numperm    : int 5000
  .. ..$ mu         : num 0
  .. ..$ paired     : logi FALSE
  .. ..$ conflevel  : num 0.98
  .. ..$ varequ     : logi TRUE
  ..@ numfilters: num 2
```

3.11 FilterTreeSet class

Class *FilterTreeSet* is obtained as result of filtering an instance of class *ExprTreeSet*. It contains the filter mask in slot `data`.

```
> getClassDef("FilterTreeSet")

Class âĂIJFilterTreeSetâĂ▪

Slots:

Name:         filter       exprset       callset        scheme          data
Class:        Filter   ExprTreeSet   CallTreeSet SchemeTreeSet    data.frame

Name:         params       setname       settype      rootfile       filedir
Class:          list     character     character     character     character

Name:       numtrees     treenames
Class:       numeric          list

Extends:
Class "ProcesSet", directly
Class "TreeSet", by class "ProcesSet", distance 2
```

In addition to the slots of classes *ProcesSet* and *TreeSet* it contains the following slots:

`filter`:Object of class `"Filter"` currently providing access to the `PreFilter` settings.

`exprset`:Object of class `"ExprTreeSet"` providing direct access to the `ExprTreeSet` used for filtering.

`callset`:Object of class `"CallTreeSet"` providing direct access to the optional `CallTreeSet` used for filtering.

3.12 AnalysisTreeSet class

Class *AnalysisTreeSet* is obtained as result of filtering an instance of class *ExprTreeSet* using function `unifilter`. It contains currently the results of the univariate analysis in slot `data`.

```
> getClassDef("AnalysisTreeSet")

Class âĂIJAnalysisTreeSetâĂ▪

Slots:
```

```
Name:        fltrset        scheme          data        params       setname
Class: FilterTreeSet SchemeTreeSet    data.frame          list     character

Name:        settype      rootfile       filedir      numtrees     treenames
Class:     character     character     character       numeric          list

Extends:
Class "ProcesSet", directly
Class "TreeSet", by class "ProcesSet", distance 2
```

In addition to the slots of classes *ProcesSet* and *TreeSet* it contains the following slots:

`fltrset`:Object of class `"FilterTreeSet"` providing indirect access to the `ExprTreeSet` used and the `UniFilter` settings.

References

The ROOT team. ROOT User Guide. Technical report, CERN, 2007. URL `http://root.cern.ch/root/doc/RootDoc.html`.