

# xps

November 11, 2009

## R topics documented:

addData-methods . . . . .	3
AnalysisTreeSet-class . . . . .	4
attachBgrd-methods . . . . .	5
attachCall-methods . . . . .	6
attachExpr-methods . . . . .	7
attachInten-methods . . . . .	9
attachMask-methods . . . . .	10
bgcorrect . . . . .	11
boxplot.dev . . . . .	13
boxplot-methods . . . . .	14
callFilter-methods . . . . .	15
callplot-methods . . . . .	16
CallTreeSet-class . . . . .	17
cvFilter-methods . . . . .	18
dabg.call . . . . .	19
DataTreeSet-class . . . . .	22
dfw . . . . .	24
diffFilter-methods . . . . .	26
existsROOTFile . . . . .	27
exonLevel . . . . .	28
export.filter . . . . .	30
export . . . . .	31
export.root . . . . .	33
express . . . . .	34
exprs-methods . . . . .	37
ExprTreeSet-class . . . . .	39
farms . . . . .	40
fcFilter-methods . . . . .	43
Filter-class . . . . .	44
FilterTreeSet-class . . . . .	44
gapFilter-methods . . . . .	46
getChipName . . . . .	47
getChipType . . . . .	47
getDatatype . . . . .	48
getNameType . . . . .	49
getNumberTrees . . . . .	50
getProbeInfo . . . . .	50

getTreeData-methods . . . . .	51
getTreeNames . . . . .	52
highFilter-methods . . . . .	53
hist-methods . . . . .	54
image.dev . . . . .	55
image-methods . . . . .	56
import.data . . . . .	57
import.exon.scheme . . . . .	59
import.expr.scheme . . . . .	61
import.genome.scheme . . . . .	63
ini.call . . . . .	65
initialize-methods . . . . .	68
intensity-methods . . . . .	68
isROOTFile . . . . .	69
lowFilter-methods . . . . .	70
madFilter-methods . . . . .	71
mas4 . . . . .	72
mas5.call . . . . .	74
mas5 . . . . .	76
mboxplot-methods . . . . .	79
metaProbesets . . . . .	80
mvaplot.dev . . . . .	81
mvaplot-methods . . . . .	82
normalize . . . . .	83
pm-methods . . . . .	84
pmplot-methods . . . . .	86
PreFilter-class . . . . .	87
PreFilter-constructor . . . . .	89
prefilter . . . . .	91
presCall-methods . . . . .	92
ProcesSet-class . . . . .	94
ProjectInfo-class . . . . .	95
ProjectInfo-constructor . . . . .	97
quantileFilter-methods . . . . .	99
ratioFilter-methods . . . . .	100
rawCELName-methods . . . . .	101
rma . . . . .	102
root.browser-methods . . . . .	105
root.call . . . . .	106
root.data . . . . .	107
root.density . . . . .	108
root.expr . . . . .	109
root.graph1D . . . . .	110
root.graph2D . . . . .	111
root.hist1D . . . . .	112
root.hist2D . . . . .	113
root.hist3D . . . . .	114
root.image . . . . .	115
root.merge.data . . . . .	117
root.mvaplot . . . . .	118
root.profile . . . . .	119
ROOT . . . . .	120

root.scheme . . . . .	121
SchemeTreeSet-class . . . . .	122
summarize . . . . .	124
TreeSet-class . . . . .	125
type2Exten . . . . .	127
UniFilter-class . . . . .	128
UniFilter-constructor . . . . .	129
unifilter . . . . .	130
unittestFilter-methods . . . . .	132
uniTest-methods . . . . .	133
validData-methods . . . . .	134
validTreetype . . . . .	135
varFilter-methods . . . . .	137
volcanoplot-methods . . . . .	137
xpsOptions . . . . .	138
xps-package . . . . .	139

## Index 140

---

addData-methods      *Import additional CEL files into a DataTreeSet*

---

### Description

Import additional CEL files into a DataTreeSet and update [ROOT](#) data file.

#### Usage

```
addData(object, celdir = NULL, celfiles = "", celnames = NULL, project
= NULL, verbose = TRUE)
```

### Arguments

object	object of class DataTreeSet.
celdir	system directory containing the CEL-files for corresponding scheme.
celfiles	optional vector of CEL-files to be imported.
celnames	optional vector of names which should replace the CEL-file names.
project	optional class <a href="#">ProjectInfo</a> .
verbose	logical, if TRUE print status information.

### Details

Import additional CEL-files and update [ROOT](#) data file `rootfile`.

To import CEL-files from different directories, vector `celfiles` must contain the full path for each CEL-file and `celdir` must be `celdir=NULL`.

### Value

A DataTreeSet object.

### Author(s)

Christian Stratowa

**See Also**

[import.data](#), [root.data](#)

**Examples**

```
## get scheme and import subset of CEL-files from package
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- import.data(scheme.test3, "tmp_test3", celdir=paste(.path.package("xps"), "raw",
                      celfiles=c("TestA1.CEL", "TestB2.CEL"), verbose=FALSE)

unlist(treeNames(data.test3))

## add further subset of CEL-files
data.test3 <- addData(data.test3, celdir=paste(.path.package("xps"), "raw", sep="/"),
                      celfiles=c("TestA2.CEL", "TestB1.CEL"), verbose=FALSE)

unlist(treeNames(data.test3))
```

---

AnalysisTreeSet-class

*Class AnalysisTreeSet*

---

**Description**

This class provides the link to the [ROOT](#) analysis file and the [ROOT](#) trees contained therein. It extends class [ProcesSet](#).

**Objects from the Class**

Objects are currently created using function [unifilter](#).

**Slots**

**fltrset:** Object of class "FilterTreeSet" providing indirect access to the [ExprTreeSet](#) used and the [UniFilter](#) settings.

**scheme:** Object of class "SchemeTreeSet" providing access to [ROOT](#) scheme file.

**data:** Object of class "data.frame". The data.frame contains the data of the unittest stored in [ROOT](#) data trees.

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

**setname:** Object of class "character" representing the name to the [ROOT](#) file subdirectory where the [ROOT](#) trees are stored, currently 'UniFilterSet'.

**settype:** Object of class "character" describing the type of treeset stored in setname, currently 'unifilter'.

**rootfile:** Object of class "character" representing the name of the [ROOT](#) file, including 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 subdirectory setname.

**treenames:** Object of class "list" representing the names of the [ROOT](#) trees stored in subdirectory setname.

**Extends**

Class "[ProcesSet](#)", directly. Class "[TreeSet](#)", by class "[ProcesSet](#)", distance 2.

**Methods**

**filterTreeset** signature(object = "[AnalysisTreeSet](#)"): extracts slot fltrset.

**getTreeData** signature(object = "[AnalysisTreeSet](#)"): exports tree data and returns a data.frame.

**validData** signature(object = "[AnalysisTreeSet](#)"): extracts data.frame data.

**validFilter** signature(object = "[AnalysisTreeSet](#)"): extracts data.frame data from fltrset.

**volcanoplot** signature(x = "[AnalysisTreeSet](#)"): creates a volcano-plot.

**Author(s)**

Christian Stratowa

**See Also**

related classes [FilterTreeSet](#).

**Examples**

```
showClass("AnalysisTreeSet")
```

---

attachBgrd-methods *Attach/Remove Background Intensities*

---

**Description**

Attach/remove background intensities to/from [DataTreeSet](#).

*Usage*

```
attachBgrd(object, treenames = "*")
```

```
removeBgrd(object)
```

**Arguments**

object            Object of class "[DataTreeSet](#)".

treenames        Object of class "[list](#)" representing the names of the [ROOT](#) background trees.

**Details**

Whenever one of the [bgcorrect](#) methods will be applied to raw CEL intensities, the background intensities will be stored in [ROOT](#) background trees. However, the background intensities will not be saved as data.frame bgrd, thus avoiding memory problems. Function [attachBgrd](#) allows to fill slot bgrd on demand.

[attachBgrd](#) exports intensities from background trees from [ROOT](#) data file and saves as data.frame bgrd. treenames is a vector of tree names to attach; for treenames="\*" all trees from slot treenames will be exported and background intensities attached as data.frame bgrd.

[removeBgrd](#) removes background intensities from [DataTreeSet](#) and replaces data.frame bgrd with an empty data.frame of dim(0,0).

**Value**

A [DataTreeSet](#) object.

**Note**

Do not use `attachBgrd` unless you know that your computer has sufficient RAM, especially when using exon arrays. It may be advisable to use a subset of `treenames` only.

**Author(s)**

Christian Stratowa

**See Also**

[attachInten](#), [removeInten](#)

---

attachCall-methods *Attach/Remove Detecion Call Measures*

---

**Description**

Attach/remove detection call and detection p-value to/from [CallTreeSet](#).

*Usage*

```
attachCall(object, treenames = "*")
attachPVal(object, treenames = "*")
removeCall(object)
removePVal(object)
```

**Arguments**

<code>object</code>	Object of class "CallTreeSet".
<code>treenames</code>	Object of class "list" representing the names of the <a href="#">ROOT</a> call trees.

**Details**

By default detection calls will be saved in class [CallTreeSet](#) in slots `data` and `detcall`, respectively, since usually the `data.frames` obtained as result of e.g. `mas5.call` are of reasonable size. However, when computing many arrays, especially exon arrays at probeset levels, it may be better to compute detection calls with slot `add.data=FALSE` thus avoiding memory problems. In this case, functions `attachCall` and `attachPVal` allow to fill slots `detcall` and `data`, respectively, on demand.

`attachCall` exports detection calls from call trees from [ROOT](#) call file and and saves as `data.frame detcall`. `treenames` is a vector of tree names to attach; for `treenames="*"` all trees from slot `treenames` will be exported and detection calls attached as `data.frame detcall`.

`attachPVal` exports detection p-values from call trees from [ROOT](#) call file and and saves as `data.frame data`. `treenames` is a vector of tree names to attach; for `treenames="*"` all trees from slot `treenames` will be exported and detection p-values attached as `data.frame data`.

`removeCall` removes detection calls from [CallTreeSet](#) and replaces `data.frame detcall` with an empty `data.frame` of `dim(0,0)`.

removePVal removes detection p-values from `CallTreeSet` and replaces `data.frame` data with an empty `data.frame` of `dim(0,0)`.

**Value**

A `CallTreeSet` object.

**Note**

Do not use `attachCall` and `attachPVal` unless you know that your computer has sufficient RAM, especially when using exon arrays. It may be advisable to use a subset of `treenames` only.

**Author(s)**

Christian Stratowa

**See Also**

`attachExpr`, `removeExpr`

**Examples**

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel

## MAS5 detection call
call.mas5 <- mas5.call(data.test3, "tmp_Test3Call10", tmpdir="", add.data=FALSE, verbose=FALSE

## attach data
call.mas5 <- attachPVal(call.mas5)
call.mas5 <- attachCall(call.mas5)

## get data.frames
pval.mas5 <- pvalData(call.mas5)
pres.mas5 <- presCall(call.mas5)
head(pval.mas5)
head(pres.mas5)

## remove data
call.mas5 <- removePVal(call.mas5)
call.mas5 <- removeCall(call.mas5)

rm(scheme.test3, data.test3)
gc()
```

**Description**

Attach/remove expression levels to/from [ExprTreeSet](#).

*Usage*

```
attachExpr(object, treenames = "*")
removeExpr(object)
```

**Arguments**

`object`            Object of class "ExprTreeSet".  
`treenames`        Object of class "list" representing the names of the [ROOT](#) expression trees.

**Details**

By default expression levels will be saved in class [ExprTreeSet](#) as slot data, since usually the `data.frame` obtained as result of e.g. `rma` normalization is of reasonable size. However, when normalizing many arrays, especially exon arrays at probeset levels, it may be better to compute `rma` with slot `add.data=FALSE` thus avoiding memory problems. In this case, function `attachExpr` allows to fill slot data on demand.

`attachExpr` exports expression levels from expression trees from [ROOT](#) expression file and saves as `data.frame` `data`. `treenames` is a vector of tree names to attach; for `treenames="*"` all trees from slot `treenames` will be exported and expression levels attached as `data.frame` `data`.

`removeExpr` removes expression levels from [ExprTreeSet](#) and replaces `data.frame` `data` with an empty `data.frame` of `dim(0,0)`.

**Value**

A [ExprTreeSet](#) object.

**Note**

Do not use `attachExpr` unless you know that your computer has sufficient RAM, especially when using exon arrays. It may be advisable to use a subset of `treenames` only.

**Author(s)**

Christian Stratowa

**See Also**

[attachCall](#), [removeCall](#)

**Examples**

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

data.rma <- rma(data.test3, "tmp_Test3RMA0", tmpdir="", background="pmonly", normalize=TRUE, a

## attach data
data.rma <- attachExpr(data.rma)
```



```
## get data.frame
expr.rma <- validData(data.rma)
head(expr.rma)

## remove data
data.rma <- removeExpr(data.rma)

rm(scheme.test3, data.test3)
gc()
```

---

attachInten-methods

*Attach/Remove Intensities*

---

## Description

Attach/remove raw CEL intensities to/from [DataTreeSet](#).

### Usage

```
attachInten(object, treenames = "*")
removeInten(object)
```

## Arguments

`object`            Object of class "DataTreeSet".  
`treenames`        Object of class "list" representing the names of the [ROOT](#) data trees.

## Details

When CEL files will be imported using function [import.data](#), the raw intensities will be stored in [ROOT](#) data trees. However, the intensities will not be saved in class [DataTreeSet](#) as slot data, thus avoiding memory problems. Function `attachInten` allows to fill slot data on demand.

`attachInten` exports intensities from data trees from [ROOT](#) data file and and saves as `data.frame` data. `treenames` is a vector of tree names to attach; for `treenames="*"` all trees from slot `treenames` will be exported and intensities attached as `data.frame` data.

`removeInten` removes intensities from [DataTreeSet](#) and replaces `data.frame` data with an empty `data.frame` of `dim(0,0)`.

## Value

A [DataTreeSet](#) object.

## Note

Do not use `attachInten` unless you know that your computer has sufficient RAM, especially when using exon arrays. It may be advisable to use a subset of `treenames` only.

## Author(s)

Christian Stratowa

**See Also**

[attachBgrd](#), [removeBgrd](#)

**Examples**

```
## load existing ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.
dim(intensity(data.test3))

data.test3 <- attachInten(data.test3)
dim(intensity(data.test3))
head(intensity(data.test3))

data.test3 <- removeInten(data.test3)
dim(intensity(data.test3))
```

---

attachMask-methods *Attach/Remove Scheme Mask*

---

**Description**

Attach/remove scheme mask to/from [SchemeTreeSet](#) or to slot scheme of [DataTreeSet](#).

**Usage**

```
attachMask(object)
```

```
removeMask(object)
```

**Arguments**

object            Object of class "SchemeTreeSet" or "DataTreeSet".

**Details**

`attachMask` exports mask from scheme tree from [ROOT](#) scheme file and and saves mask as `data.frame` mask of slot scheme.

`removeMask` removes mask from [SchemeTreeSet](#) or from slot scheme of [DataTreeSet](#) and replaces `data.frame` mask with an empty `data.frame` of `dim(0,0)`.

**Value**

A [DataTreeSet](#) object or [SchemeTreeSet](#).

**Note**

Do not use `attachMask` unless you know that your computer has sufficient RAM, especially for exon array schemes.

**Author(s)**

Christian Stratowa

**See Also**

[import.expr.scheme](#), [import.exon.scheme](#)

**Examples**

```
## load existing ROOT scheme file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
dim(chipMask(scheme.test3))

scheme.test3 <- attachMask(scheme.test3)
dim(chipMask(scheme.test3))
head(chipMask(scheme.test3))

scheme.test3 <- removeMask(scheme.test3)
dim(chipMask(scheme.test3))
```

---

 bgcorrect

*Background Correction*


---

**Description**

Background corrects probe intensities in an object of class `DataTreeSet`.

**Usage**

```
bgcorrect(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE)
bgcorrect.gc(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE)
bgcorrect.mas4(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE)
bgcorrect.mas5(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE)
bgcorrect.rma(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE)
xpsBgCorrect(object, ...)
```

**Arguments**

<code>xps.data</code>	object of class <code>DataTreeSet</code> .
<code>filename</code>	file name of ROOT data file.
<code>filedir</code>	system directory where ROOT data file should be stored.
<code>tmpdir</code>	optional temporary directory where temporary ROOT files should be stored.
<code>update</code>	logical. If <code>TRUE</code> the existing ROOT data file <code>filename</code> will be updated.
<code>select</code>	type of probes to select for background correction.
<code>method</code>	background method to use.
<code>option</code>	type of background correction to use.
<code>exonlevel</code>	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.

params	vector of parameters for background method.
verbose	logical, if TRUE print status information.
object	object of class DataSet.
...	the arguments described above.

## Details

Background corrects probe intensities in an object of class `DataTreeSet`.

`xpsBgCorrect` is the `DataSet` method called by function `bgcorrect`, containing the same parameters.

## Value

An `DataTreeSet`

## Author(s)

Christian Stratowa

## See Also

[express](#)

## Examples

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/"))
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## MAS4 sector background
data.bg.mas4 <- bgcorrect.mas4(data.test3, "tmp_Test3MAS4Bgrd", filedir=getwd(), tmpdir="", v

## need to attach background intensities
data.bg.mas4 <- attachBgrd(data.bg.mas4)

## get data.frame
bg.mas4 <- validBgrd(data.bg.mas4)
head(bg.mas4)

## plot images
if (interactive()) {
  image.dev(data.bg.mas4, bg=TRUE, col=rainbow(32))
  image(matrix(bg.mas4[,1], ncol=ncols(schemeSet(data.bg.mas4)), nrow=nrows(schemeSet(data.
})

## Not run:
## examples using Affymetrix human tissue dataset (see also xps/examples/script4exon.R)

## example - exon array, e.g. HuEx-1_0-st-v2:
scmdir <- "/Volumes/GigaDrive/CRAN/Workspaces/Schemes"
datdir <- "/Volumes/GigaDrive/CRAN/Workspaces/ROOTData"
scheme.exon <- root.scheme(paste(scmdir, "Scheme_HuEx10stv2r2_na25.root", sep="/"))
data.exon <- root.data(scheme.exon, paste(datdir, "HuTissuesExon_cel.root", sep="/"))

## compute rma background
```

```

workdir <- "/Volumes/GigaDrive/CRAN/Workspaces/Exon/hutissues/exon"
data.bg.rma <- bgcorrect(data.exon, "HuExonRMABgrd", filedir=workdir, tmpdir="",
                        method="rma", select="antigenomic", option="pmonly:epanechnikov",
                        params=c(16384), exonlevel="metacore+affx")

# or alternatively:
data.bg.rma <- bgcorrect.rma(data.exon, "HuExonRMABgrd", filedir=workdir, tmpdir="",
                            select="antigenomic", exonlevel="metacore+affx")

## End(Not run)

```

---

boxplot.dev

*Box Plots for Device*


---

## Description

Produce box-and-whisker plot(s) of the samples for the selected device.

## Usage

```
boxplot.dev(x, which = "", size = 0, transfo = log2, range = 0, names = "namepar
```

## Arguments

x	object of class <a href="#">DataTreeSet</a> or <a href="#">ExprTreeSet</a> .
which	type of probes to be used, for details see <a href="#">validData</a> .
size	length of sequence to be generated as subset.
transfo	a valid function to transform the data, usually <code>log2</code> , or <code>0</code> .
range	determines how far the plot whiskers extend out from the box.
names	optional vector of sample names.
mar	plot margin.
las	style of axis labels.
dev	graphics device to plot to, i.e. one of “screen”, “jpeg”, “png”, “pdf” or “ps”.
outfile	the name of the output file.
w	the width of the device in pixels.
h	the height of the device in pixels.
...	optional arguments to be passed to <code>boxplot</code> .

## Details

Produces a boxplot for slot `data` for an object of class [DataTreeSet](#) or [ExprTreeSet](#) for the selected graphics device.

## Note

For a [DataTreeSet](#) object, data must first be attached using method [attachInten](#).

## Author(s)

Christian Stratowa

**See Also**

[boxplot](#)

---

boxplot-methods      *Box Plots*

---

**Description**

Produce box-and-whisker plot(s) of the samples.

*Usage*

```
boxplot(x, which = "", size = 0, transfo = log2, range = 0, names  
= "namepart", ...)
```

**Arguments**

x	object of class <a href="#">DataTreeSet</a> or <a href="#">ExprTreeSet</a> .
which	type of probes to be used, for details see <a href="#">validData</a> .
size	length of sequence to be generated as subset.
transfo	a valid function to transform the data, usually “log2”, or “0”.
range	determines how far the plot whiskers extend out from the box.
names	optional vector of sample names.
...	optional arguments to be passed to <a href="#">boxplot</a> .

**Details**

Creates a boxplot for slot data for an object of class [DataTreeSet](#) or [ExprTreeSet](#).

For names=NULL full column names of slot data will be displayed while for names="namepart" column names will be displayed without name extension. If names is a vector of column names, only these columns will displayed as boxplot.

**Note**

For a [DataTreeSet](#) object, data must first be attached using method [attachInten](#).

**Author(s)**

Christian Stratowa

**See Also**

[boxplot.dev](#), [boxplot](#)

**Examples**

```
## load existing ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## need to attach scheme mask and probe intensities
data.test3 <- attachMask(data.test3)
data.test3 <- attachInten(data.test3)

if (interactive()) {
  boxplot(data.test3)
}

## optionally remove mask and data to free memory
data.test3 <- removeInten(data.test3)
data.test3 <- removeMask(data.test3)

rm(scheme.test3, data.test3)
gc()
```

---

callFilter-methods *Detection Call Filter*

---

**Description**

Detection Call Filter.

The `cutoff` value defines the upper threshold for allowed detection call p-values. If e.g. the number of samples exceeding this cutoff value is greater than `samples` then the corresponding expression dataframe row is flagged, i.e. `flag = 0`.

The Detection Call Filter flags all rows with: `flag = (sum(call[i] >= cutoff) >= samples)`

*Usage*

```
callFilter(object)
callFilter(object, value)<-
```

**Arguments**

`object`            object of class `PreFilter` or `UniFilter`.  
`value`             character vector `c(cutoff, samples, condition)`.

**Details**

The method `callFilter` initializes the following parameters:

<code>cutoff</code> :	the cutoff value for the filter: <code>cutoff = 1.0</code> : present/absent call is used. <code>cutoff &lt; 1.0</code> : detection p-value is used as cutoff.
<code>samples</code> :	this value depends on the condition used:
<code>condition</code> :	<code>condition="samples"</code> : number of samples (default): <code>condition="percent"</code> : percent of samples.

**Value**

An initialized `PreFilter` or `UniFilter` object.

**Author(s)**

Christian Stratowa

**Examples**

```
## initialize PreFilter
prefltr <- PreFilter()
callFilter(prefltr) <- c(0.02, 80.0, "percent")
str(prefltr)

## initialize UniFilter
unifltr <- UniFilter()
callFilter(unifltr) <- c(0.02, 80.0, "percent")
str(unifltr)
```

---

callplot-methods    *Barplot of Percent Present and Absent Calls.*

---

**Description**

Creates a barplot of percent Present/Marginal/Absent calls.

*Usage*

```
callplot(x, beside = TRUE, names = "namepart", col = c("red", "green", "blue"),
legend = c("P", "M", "A"), ...)
```

**Arguments**

<code>x</code>	object of class <code>CallTreeSet</code> .
<code>beside</code>	logical. If <code>FALSE</code> , the columns of height are portrayed as stacked bars, and if <code>TRUE</code> the columns are portrayed as juxtaposed bars.
<code>names</code>	optional vector of sample names.
<code>col</code>	color for P/M/A bars
<code>legend</code>	legend for the plot, defaults to P/M/A.
<code>...</code>	optional arguments to be passed to <code>barplot</code> .

**Details**

Creates a barplot of percent Present/Marginal/Absent calls.

For `names=NULL` full column names of slot data will be displayed while for `names="namepart"` column names will be displayed without name extension. If `names` is a vector of column names, only these columns will displayed as callplot.

**Author(s)**

Christian Stratowa



**See Also**

[pmpplot](#)

---

CallTreeSet-class    *Class CallTreeSet*

---

**Description**

This class provides the link to the [ROOT](#) call file and the [ROOT](#) trees contained therein. It extends class [ProcesSet](#).

**Objects from the Class**

Objects are created using functions [mas5.call](#) or [dabg.call](#), respectively.

**Slots**

**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.

**scheme:** Object of class "SchemeTreeSet" providing access to [ROOT](#) scheme file.

**data:** Object of class "data.frame". The data.frame can contain the data (i.e. p-values) stored in [ROOT](#) call trees.

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

**setname:** Object of class "character" representing the name to the [ROOT](#) file subdirectory where the [ROOT](#) call trees are stored, usually 'CallTreeSet'.

**settype:** Object of class "character" describing the type of treeset stored in setname, usually 'preprocess'.

**rootfile:** Object of class "character" representing the name of the [ROOT](#) call file, including 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 subdirectory setname.

**treenames:** Object of class "list" representing the names of the [ROOT](#) trees stored in subdirectory setname.

**Extends**

Class "[ProcesSet](#)", directly. Class "[TreeSet](#)", by class "[ProcesSet](#)", distance 2.

**Methods**

**attachCall** signature(object = "CallTreeSet"): exports detection call data from [ROOT](#) call file and and saves as data.frame detcall.

**attachPVal** signature(object = "CallTreeSet"): exports call p-values from [ROOT](#) call file and and saves as data.frame data.

**callplot** signature(x = "CallTreeSet"): creates a barplot of percent present and absent calls.

**presCall** signature(object = "CallTreeSet"): extracts the detection call data.frame.

**presCall<-** signature(object = "CallTreeSet", value = "data.frame"): replaces the detection call data.frame.

**pvalData** signature(object = "CallTreeSet"): extracts the detection p-value data.frame.

**pvalData<-** signature(object = "CallTreeSet", value = "data.frame"): replaces the detection p-value data.frame.

**removeCall** signature(object = "CallTreeSet"): replaces data.frame detcall with an empty data.frame of dim(0,0).

**removePVal** signature(object = "CallTreeSet"): replaces data.frame data with an empty data.frame of dim(0,0).

**validCall** signature(object = "CallTreeSet"): extracts a subset of columns from data.frame detcall.

**Author(s)**

Christian Stratowa

**See Also**

related classes [DataTreeSet](#), [ExprTreeSet](#).

**Examples**

```
showClass("CallTreeSet")
```

---

cvFilter-methods      *Coefficient of Variation Filter*

---

**Description**

This method initializes the Coefficient of Variation Filter.

The coefficient of variation is the standard deviation divided by the absolute value of the mean.

The CV Filter flags all rows with: `flag = (cv >= cutoff)`

*Usage*

```
cvFilter(object)
cvFilter(object, value)<-
```

**Arguments**

object                      object of class PreFilter.

value            numeric vector `c(cutoff, trim, epsilon)`.

### Details

The method `cvFilter` initializes the following parameters:

```

cutoff:    the cutoff level for the filter.
trim:      the trim value for trimmed mean (default is trim=0).
epsilon:   value to replace mean (default is epsilon=0.01):
           epsilon > 0: replace mean=0 with epsilon.
           epsilon = 0: always set mean=1.

```

Note, that for `epsilon = 0` the filter flags all rows with: `stdev >= cutoff`

### Value

An initialized `PreFilter` object.

### Author(s)

Christian Stratowa

### Examples

```

prefltr <- PreFilter()
cvFilter(prefltr) <- c(0.3,0.0,0.01)
str(prefltr)

```

---

dabg.call

*Detection Above Background Call*

---

### Description

Computes the Detection Above Background Call first implemented for the Exon arrays.

### Usage

```

dabg.call(xps.data, filename = character(0), filedir = getwd(),
          alpha1 = 0.04, alpha2 = 0.06,
          option = "transcript", exonlevel = "", xps.scheme = NULL, add.data = T)

xpsDABGCall(object, ...)

```

### Arguments

```

xps.data        object of class DataTreeSet.
filename        file name of ROOT data file.
filedir        system directory where ROOT data file should be stored.
alpha1         a significance threshold in (0,alpha2).
alpha2         a significance threshold in (alpha1,0.5).

```

<code>option</code>	option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon arrays only.
<code>exonlevel</code>	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
<code>xps.scheme</code>	optional alternative <code>SchemeTreeSet</code> .
<code>add.data</code>	logical. If TRUE call data will be added to slots data and <code>detcall</code> .
<code>verbose</code>	logical, if TRUE print status information.
<code>object</code>	object of class <code>DataTreeSet</code> .
<code>...</code>	the arguments described above.

### Details

This function generates a detection p-value based on comparing the perfect match probe intensity to the intensity distribution provided by background probes sharing the same GC-content as the PM probe under consideration. For exon/genome arrays special 'antigenomic' background probes of defined GC-content are used, while for expression arrays the Mismatch probes will be grouped by their GC-content.

For exon/genome arrays it is necessary to supply `option` and `exonlevel`.

Following options are valid for exon arrays only:

<code>transcript:</code>	expression levels are computed for transcript clusters, i.e. probe sets containing the same 'transcript_cluster_id'.
<code>exon:</code>	expression levels are computed for exon clusters, i.e. probe sets containing the same 'exon_id', where each probe set is associated with a single exon.
<code>probeset:</code>	expression levels are computed for individual probe sets, i.e. for each 'probeset_id'.

Following `exonlevel` annotations are valid for exon arrays:

<code>core:</code>	probesets supported by RefSeq and full-length GenBank transcripts.
<code>metacore:</code>	core meta-probesets.
<code>extended:</code>	probesets with other cDNA support.
<code>metaextended:</code>	extended meta-probesets.
<code>full:</code>	probesets supported by gene predictions only.
<code>metafull:</code>	full meta-probesets.
<code>ambiguous:</code>	ambiguous probesets only.
<code>affx:</code>	standard AFFX controls.
<code>all:</code>	combination of above.

Following `exonlevel` annotations are valid for whole genome arrays:

<code>core:</code>	probesets with category 'unique' and 'mixed'.
<code>metacore:</code>	probesets with category 'unique' only.
<code>affx:</code>	standard AFFX controls.
<code>all:</code>	combination of above.

Exon levels can also be combined, with following combinations being most useful:

<code>exonlevel="metacore+affx":</code>	core meta-probesets plus AFFX controls
<code>exonlevel="core+extended":</code>	probesets with cDNA support
<code>exonlevel="core+extended+full":</code>	supported plus predicted probesets

Exon level annotations are described in the Affymetrix whitepaper 'exon\_probeset\_trans\_clust\_whitepaper.pdf'.

In order to use an alternative [SchemeTreeSet](#) set the corresponding `SchemeTreeSet` `xps.scheme`.

`xpsDABGCall` is the `DataTreeSet` method called by function `dabg.call`, containing the same parameters.

### Value

A [CallTreeSet](#)

### Note

Yes, it is possible to compute DABG detection call for expression arrays, but it is very slow and thus not recommended.

### Author(s)

Christian Stratowa

### References

Affymetrix (2005) Exon Probeset Annotations and Transcript Cluster Groupings, Affymetrix Inc., Santa Clara, CA, [exon\\_probeset\\_trans\\_clust\\_whitepaper.pdf](#).

### See Also

[mas5.call](#)

### Examples

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## DABG detection call
call.dabg <- dabg.call(data.test3, "tmp_Test3DABG", verbose=FALSE)

## get data.frames
pval.dabg <- pvalData(call.dabg)
pres.dabg <- presCall(call.dabg)
head(pval.dabg)
head(pres.dabg)

## plot results
if (interactive()) {
  callplot(call.dabg)
}

rm(scheme.test3, data.test3)
gc()
```

---

DataTreeSet-class    *Class DataTreeSet*

---

### Description

This class provides the link to the [ROOT](#) data file and the [ROOT](#) trees contained therein. It extends class [ProcesSet](#).

### Objects from the Class

Objects can be created using the functions [import.data](#) or [root.data](#).

### Slots

**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.

**scheme:** Object of class "SchemeTreeSet" providing access to [ROOT](#) scheme file.

**data:** Object of class "data.frame". The data.frame can contain the data (e.g. intensities) stored in [ROOT](#) data trees.

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

**setname:** Object of class "character" representing the name to the [ROOT](#) file subdirectory where the [ROOT](#) data trees are stored, usually 'DataTreeSet'.

**settype:** Object of class "character" describing the type of treeset stored in setname, usually 'rawdata'.

**rootfile:** Object of class "character" representing the name of the [ROOT](#) data file, including 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 subdirectory setname.

**treenames:** Object of class "list" representing the names of the [ROOT](#) trees stored in subdirectory setname.

### Extends

Class "[ProcesSet](#)", directly. Class "[TreeSet](#)", by class "[ProcesSet](#)", distance 2.

### Methods

**addData** signature(object = "DataTreeSet"): import additional CEL-files and update [ROOT](#) data file rootfile.

**attachBgrd** signature(object = "DataTreeSet"): exports background trees from [ROOT](#) data file and and saves as data.frame bgrd.

**attachInten** signature(object = "DataTreeSet"): exports intensity trees from [ROOT](#) data file and and saves as data.frame data.

**attachMask** signature(object = "DataTreeSet"): exports scheme tree from `ROOT` scheme file and saves as `data.frame` mask of slot scheme.

**background** signature(object = "DataTreeSet"): extracts slot `bgrd`.

**background<-** signature(object = "DataTreeSet", value = "data.frame"): replaces slot `bgrd`.

**bgtreeNames** signature(object = "DataTreeSet"): extracts slot `bgtreenames`.

**image** signature(x = "DataTreeSet"): creates an image for each column from `data.frame` `data` or `bgrd`, respectively.

**intensity** signature(object = "DataTreeSet"): extracts slot `data`.

**intensity<-** signature(object = "DataTreeSet", value = "data.frame"): replaces slot `data`.

**mm** signature(object = "DataTreeSet"): extracts the mismatch intensities.

**ncols** signature(object = "DataTreeSet"): extracts the physical number of array columns from slot scheme.

**nrows** signature(object = "DataTreeSet"): extracts the physical number of array rows from slot scheme.

**pm** signature(object = "DataTreeSet"): extracts the perfect match intensities.

**pmplot** signature(x = "DataTreeSet"): creates a barplot of mean perfect match and mismatch intensities.

**projectInfo** signature(object = "DataTreeSet"): extracts slot `projectinfo`.

**projectInfo<-** signature(object = "DataTreeSet", value = "ProjectInfo"): replaces slot `projectinfo`.

**rawCELName** signature(object = "DataTreeSet"): returns the name(s) of the imported raw CEL-files.

**removeBgrd** signature(object = "DataTreeSet"): replaces `data.frame` `bgrd` with an empty `data.frame` of `dim(0,0)`.

**removeInten** signature(object = "DataTreeSet"): replaces `data.frame` `data` with an empty `data.frame` of `dim(0,0)`.

**removeMask** signature(object = "DataTreeSet"): replaces `data.frame` `mask` from slot scheme with an empty `data.frame` of `dim(0,0)`.

**validBgrd** signature(object = "DataTreeSet"): extracts the valid data from `data.frame` `bgrd`.

**validData** signature(object = "DataTreeSet"): extracts a subset of valid data from `data.frame` `data`.

**xpsBgCorrect** signature(object = "DataTreeSet"): applies background correction methods. See `bgcorrect`.

**xpsDABGCall** signature(object = "DataTreeSet"): computes DABG call.

**xpsINICall** signature(object = "DataTreeSet"): computes I/NI call.

**xpsMAS4** signature(object = "DataTreeSet"): computes MAS4 expression levels.

**xpsMAS5** signature(object = "DataTreeSet"): computes MAS5 expression levels.

**xpsMAS5Call** signature(object = "DataTreeSet"): computes MAS5 detection call.

**xpsNormalize** signature(object = "DataTreeSet"): applies normalization methods.

**xpsPreprocess** signature(object = "DataTreeSet"): applies normalization methods.

**xpsRMA** signature(object = "DataTreeSet"): computes RMA expression levels.

**xpsSummarize** signature(object = "DataTreeSet"): applies summarization methods.

**Author(s)**

Christian Stratowa

**See Also**related classes [ExprTreeSet](#), [CallTreeSet](#).**Examples**

```
showClass("DataTreeSet")
```

---

dfw

*Distribution Free Weighted Expression Measure*


---

**Description**

This function converts a [DataTreeSet](#) into an [ExprTreeSet](#) using the Distribution Free Weighted Fold Change (DFW) method.

**Usage**

```
dfw(xps.data,
    filename = character(0),
    filedir  = getwd(),
    tmpdir   = "",
    normalize = TRUE,
    m        = 3,
    n        = 1,
    c        = 0.01,
    option   = "transcript",
    exonlevel = "",
    xps.scheme = NULL,
    add.data  = TRUE,
    verbose   = TRUE)
```

**Arguments**

xps.data	object of class <a href="#">DataTreeSet</a> .
filename	file name of ROOT data file.
filedir	system directory where ROOT data file should be stored.
tmpdir	optional temporary directory where temporary ROOT files should be stored.
normalize	logical. If TRUE normalize data using quantile normalization.
m	positive number as exponent of the weighted range WR.
n	positive number as exponent of the weighted standard deviation WSD.
c	scaling parameter.
option	option determining the grouping of probes for summarization, one of ‘transcript’, ‘exon’, ‘probeset’; exon arrays only.
exonlevel	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.



`xps.scheme` optional alternative `SchemeTreeSet`.  
`add.data` logical. If TRUE expression data will be included as slot data.  
`verbose` logical, if TRUE print status information.

### Details

This function computes the DFW (Distribution Free Weighted Fold Change) expression measure described in Chen et al. for both expression arrays and exon arrays. For exon arrays it is necessary to supply the requested `option` and `exonlevel`.

Following `options` are valid for exon arrays:

`transcript:` expression levels are computed for transcript clusters, i.e. probe sets containing the same ‘`transcript_clu`  
`exon:` expression levels are computed for exon clusters, i.e. probe sets containing the same ‘`exon_id`’, where ea  
`probeset:` expression levels are computed for individual probe sets, i.e. for each ‘`probeset_id`’.

Following `exonlevel` annotations are valid for exon arrays:

`core:` probesets supported by RefSeq and full-length GenBank transcripts.  
`metacore:` core meta-probesets.  
`extended:` probesets with other cDNA support.  
`metaextended:` extended meta-probesets.  
`full:` probesets supported by gene predictions only.  
`metafull:` full meta-probesets.  
`affx:` standard AFFX controls.  
`all:` combination of above (including `affx`).

Following `exonlevel` annotations are valid for whole genome arrays:

`core:` probesets with category ‘unique’, ‘similar’ and ‘mixed’.  
`metacore:` probesets with category ‘unique’ only.  
`affx:` standard AFFX controls.  
`all:` combination of above (including `affx`).

Exon levels can also be combined, with following combinations being most useful:

`exonlevel="metacore+affy":` core meta-probesets plus AFFX controls  
`exonlevel="core+extended":` probesets with cDNA support  
`exonlevel="core+extended+full":` supported plus predicted probesets

Exon level annotations are described in the Affymetrix whitepaper `exon_probeset_trans_clust_whitepaper.pdf`: “Exon Probeset Annotations and Transcript Cluster Groupings”.

In order to use an alternative `SchemeTreeSet` set the corresponding `SchemeSet` `xps.scheme`.

### Value

An `ExprTreeSet`

### Note

The expression measure obtained with DFW is given in linear scale, analogously to the expression measures computed with `mas5` and `rma`.

For the analysis of many exon arrays it may be better to define a `tmpdir`, since this will store only the results in the main file and not e.g. background and normalized intensities, and thus will reduce the file size of the main file. For quantile normalization memory should not be an issue, however DFW depends on RAM unless you are using a temporary file.

### Author(s)

Christian Stratowa

### References

Chen, Z., McGee M., Liu Q., and Scheuermann, R.H. (2007), A distribution free summarization method for Affymetrix GeneChip arrays. *Bioinformatics* 23(3):321-327

### See Also

[express](#)

### Examples

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

data.dfw <- dfw(data.test3, "tmp_Test3DFW", verbose=FALSE)

## get data.frame
expr.dfw <- validData(data.dfw)
head(expr.dfw)
```

---

diffFilter-methods *Difference Filter*

---

### Description

This method initializes the Difference Filter.

The difference is the maximum value minus minimum value for each row of the expression dataframe divided by the mean value of each row.

The Difference Filter flags all rows with:  $flag = (max - min) / mean \geq cutoff$

#### Usage

```
diffFilter(object)
diffFilter(object, value)<-
```

### Arguments

<code>object</code>	object of class <code>PreFilter</code> .
<code>value</code>	numeric vector <code>c(cutoff, trim, epsilon)</code> .

### Details

The method `diffFilter` initializes the following parameters:

cutoff: the cutoff level for the filter.  
 trim: the trim value for trimmed mean (default is trim=0).  
 epsilon: value to replace mean (default is epsilon=0.01):  
     epsilon > 0: replace mean=0 with epsilon.  
     epsilon = 0: always set mean=1.

Note, that for `epsilon = 0` the filter flags all rows with:  $(\max - \min) \geq \text{cutoff}$

### Value

An initialized `PreFilter` object.

### Author(s)

Christian Stratowa

### Examples

```
prefltr <- PreFilter()
diffFilter(prefltr) <- c(2.2, 0.0, 0.01)
str(prefltr)
```

---

existsROOTFile      *Test for Existing ROOT File*

---

### Description

Test if a ROOT file does already exist.

### Usage

```
existsROOTFile(filename, tmp.rm = TRUE)
```

### Arguments

filename      name of ROOT file, including full path.  
 tmp.rm      logical, if TRUE then exclude filenames beginning with `dQuote(tmp_)`.

### Value

Return TRUE if file `filename` is an already existing `ROOT` file.

### Note

It is possible to create temporary `ROOT` files called “tmp” or with `filename` starting with “tmp\_” which can be overwritten. Thus by default temporary files will not be recognized by `existsROOTFile`. If you want to recognize temporary files, set `tmp.rm = TRUE`.

### Author(s)

Christian Stratowa

**See Also**[isROOTFile](#)**Examples**

```
existsROOTFile(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/"))
```

---

exonLevel	<i>Conversion of Parameter exonlevel to Integer</i>
-----------	---

---

**Description**

Conversion of parameter `exonlevel` to an integer vector.

**Usage**

```
exonLevel(exonlevel = "", chiptype = "GeneChip", as.sum = TRUE)
```

**Arguments**

<code>exonlevel</code>	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
<code>chiptype</code>	chip tpye, one of ‘GeneChip’, ‘GenomeChip’, ‘ExonChip’.
<code>as.sum</code>	logical, if TRUE an integer vector of size three will be returned, if FALSE then the levels will be split into the basic integer representations.

**Details**

Conversion of parameter `exonlevel` to an integer; this function is a utility function, which is usually only used internally.

Following `exonlevel` annotations are valid for exon arrays:

<code>core:</code>	(=8192+1024) probesets supported by RefSeq and full-length GenBank transcripts.
<code>metacore:</code>	(=8192) core meta-probesets.
<code>extended:</code>	(=4096+512) probesets with other cDNA support.
<code>metaextended:</code>	(=4096) extended meta-probesets.
<code>full:</code>	(=2048+256) probesets supported by gene predictions only.
<code>metafull:</code>	(=2048) full meta-probesets.
<code>ambiguous:</code>	(=128) probesets that fall within multiple genes.
<code>affx:</code>	(=60) standard AFFX controls.
<code>all:</code>	(=16316) combination of above (including affx).

Following `exonlevel` annotations are valid for whole genome arrays:

<code>core:</code>	(=8192+1024) probesets with category ‘unique’, ‘similar’ and ‘mixed’.
<code>metacore:</code>	(=8192) probesets with category ‘unique’ only.
<code>affx:</code>	(=60) standard AFFX controls.
<code>all:</code>	(=9276) combination of above (including affx).

Exon levels can also be combined, with following combinations being most useful:

exonlevel="metacore+affx":	core meta-probesets plus AFFX controls
exonlevel="core+extended":	probesets with cDNA support
exonlevel="core+extended+full":	supported plus predicted probesets

Exon level annotations are described in the Affymetrix whitepaper `exon_probeset_trans_clust_whitepaper.pdf`: “Exon Probeset Annotations and Transcript Cluster Groupings”.

Parameter `exonlevel` determines not only which probes are used for medianpolish, but also the probes used for background calculation and for quantile normalization. If you want to use separate probes for background calculation, quantile normalization and medianpolish summarization, you can pass a numeric vector containing three integer values corresponding to the respective `exonlevel`. These integers must be the sum of the integers shown above, e.g. you can use `exonlevel=c(16316, 8252, 8252)`, where `8252=8192+60` for "metacore+affx".

### Value

an integer vector.

### Note

The following `exonlevels` are unsupported:

<code>control-&gt;bgp-&gt;genomic:</code>	(=32768)	genomic background probes.
<code>control-&gt;bgp-&gt;antigenomic:</code>	(=65536)	antigenomic background probes.
<code>normgene-&gt;intron:</code>	(=131072)	intronic controls.
<code>normgene-&gt;exon:</code>	(=262144)	exonic controls.
<code>rescue-&gt;FLmRNA-&gt;unmapped:</code>	(=524288)	unmapped mRNAs.

For whole genome arrays it is possible (but not recommended) to use all probesets by using `exonlevel=c(992316, 992316, 992316)`. For exon arrays it is possible to use e.g. `exonlevel=c(1032124, 1032124, 631868)`. However, please note that these settings are not recommended and not supported.

### Author(s)

Christian Stratowa

### See Also

[rma](#), [mas5](#)

### Examples

```
exonLevel("core", "GenomeChip")
exonLevel("all", "GenomeChip")
exonLevel("core+extended+full", "ExonChip")
exonLevel("core+extended+full", "ExonChip", as.sum=FALSE)
exonLevel(c(16316, 8252, 8252), "ExonChip")
```

---

export.filter	<i>Export filter data as text files</i>
---------------	---

---

## Description

Export data from classes `FilterTreeSet` or `AnalysisTreeSet` to outfile.

## Usage

```
export.filter(xps.fltr, treename = "*", treetype = character(0), varlist = "*",
```

## Arguments

<code>xps.fltr</code>	an object of type <code>FilterTreeSet</code> or <code>AnalysisTreeSet</code> .
<code>treename</code>	tree name to export.
<code>treetype</code>	type of tree(s) to export, 'pfr', 'ufr' or 'stt'.
<code>varlist</code>	names of tree leaves to export.
<code>outfile</code>	name of output file.
<code>sep</code>	column separator
<code>as.dataframe</code>	if TRUE a data.frame will be returned.
<code>verbose</code>	logical, if TRUE print status information.

## Details

Export data from classes `FilterTreeSet`, or `AnalysisTreeSet` to outfile.

Parameter `varlist` lists the parameters to export:

- parameters are separated by ":", e.g. `varlist="fUnitName:fFlag"`.
- for `varlist="*"` all valid parameters will be exported.

For class `FilterTreeSet` the following `varlist` parameters are valid:

<code>fUnitName:</code>	unit name (probeset ID).
<code>fFlag:</code>	mask.

For class `AnalysisTreeSet` `varlist` can contain annotation parameters and parameters of the resulting data.

Following `varlist` annotation parameters are valid:

<code>fUnitName:</code>	unit name (probeset ID).
<code>fTranscriptID:</code>	transcript_id (probeset ID).
<code>fName:</code>	gene name.
<code>fSymbol:</code>	gene symbol.
<code>fAccession:</code>	mRNA accession such as Refseq ID.
<code>fEntrezID:</code>	entrez ID.
<code>fChromosome:</code>	chromosome.
<code>fStart:</code>	start position.
<code>fStop:</code>	stop position.
<code>fStrand:</code>	strand on chromosome.
<code>fCytoBand:</code>	cytoband.

For class `AnalysisTreeSet` the following `varlist` parameters are valid:

mn1: mean of group 1.  
mn2: mean of group 2.  
fc: fold-change  $fc=mn2/mn1$ .  
se: standard error.  
df: degree of freedom.  
stat: t-statistic.  
pval: p-value.  
nper: number of permutations.  
pcha: p-chance.  
padj: adjusted p-value.  
flag: flag.  
mask: only rows with `flag=1` will be exported.

### Value

If `as.dataframe` is `TRUE`, the data will be imported into the current R session as `data.frame`. Otherwise, `NULL` will be returned.

### Author(s)

Christian Stratowa

### See Also

[export-methods](#)

---

export

*Export data as text files*

---

### Description

Export data from classes [SchemeTreeSet](#), [DataTreeSet](#), [ExprTreeSet](#), or [CallTreeSet](#) to `outfile`.

### Usage

```
export.scheme(xps.scheme, treetype = character(0), varlist = "*", outfile = char
export.data(xps.data, treename = "*", treetype = "cel", varlist = "*", outfile =
export.expr(xps.expr, treename = "*", treetype = character(0), varlist = "*", ou
export.call(xps.call, treename = "*", treetype = character(0), varlist = "*", ou
export(object, ...)
```

**Arguments**

<code>xps.scheme</code>	an object of type <a href="#">SchemeTreeSet</a> .
<code>xps.data</code>	an object of type <a href="#">DataTreeSet</a> .
<code>xps.expr</code>	an object of type <a href="#">ExprTreeSet</a> .
<code>xps.call</code>	an object of type <a href="#">CallTreeSet</a> .
<code>treename</code>	vector of tree names to export.
<code>treetype</code>	type of tree(s) to export, see <a href="#">validTreetype</a>
<code>varlist</code>	names of tree leaves to export
<code>outfile</code>	name of output file.
<code>sep</code>	column separator
<code>as.dataframe</code>	if TRUE a data.frame will be returned.
<code>verbose</code>	logical, if TRUE print status information.
<code>object</code>	object of class <a href="#">DataTreeSet</a> .
<code>...</code>	arguments <code>treenames,treetype,varlist,outfile,sep,as.dataframe</code> .

**Details**

Export data from classes [SchemeTreeSet](#), [DataTreeSet](#), [ExprTreeSet](#), or [CallTreeSet](#) to `outfile`.

Parameter `varlist` lists the parameters to export:

- parameters are separated by ":", e.g. `varlist="fInten:fStdev"`.

- for `varlist="*"` all valid parameters will be exported.

For class [DataTreeSet](#) the following `varlist` parameters are valid:

<code>fInten:</code>	intensities from e.g. <code>tree.cel</code> .
<code>fStdev:</code>	standard deviation from e.g. <code>tree.cel</code> .
<code>fNPixels:</code>	number of pixels from e.g. <code>tree.cel</code> .
<code>fBg:</code>	background values (background trees only).

For classes [ExprTreeSet](#) and [CallTreeSet](#) `varlist` can contain annotation parameters and parameters of the resulting data.

Following `varlist` annotation parameters are valid:

<code>fUnitName:</code>	unit name (probeset ID).
<code>fTranscriptID:</code>	transcript_id (probeset ID).
<code>fName:</code>	gene name.
<code>fSymbol:</code>	gene symbol.
<code>fAccession:</code>	mRNA accession such as Refseq ID.
<code>fEntrezID:</code>	entrez ID.
<code>fChromosome:</code>	chromosome.
<code>fStart:</code>	start position.
<code>fStop:</code>	stop position.
<code>fStrand:</code>	strand on chromosome.
<code>fCytoBand:</code>	cytoband.

Following `varlist` parameters are valid for [ExprTreeSet](#):

<code>fLevel:</code>	expression level.
----------------------	-------------------



fStdev: standard deviation.  
 fNPairs: number of pairs.

Following varlist parameters are valid for CallTreeSet:

fCall: detection call.  
 fPValue: detection p-value.

An example: `varlist="fUnitName:fName:fSymbol:fLevel:fStdev:fEntrezID"`  
 export is a generic method to export data from [ROOT](#) trees as text file.

### Value

If `as.dataframe` is TRUE, the data will be imported into the current R session as `data.frame`.  
 Otherwise, NULL will be returned.

### Author(s)

Christian Stratowa

### See Also

[export-methods](#)

### Examples

```
## load existing ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## export as table only
export(scheme.test3, treetype="idx", outfile="Test3_idx.txt", verbose=FALSE)

## export as table and import as data.frame
ann <- export.scheme(scheme.test3, treetype="ann", outfile="Test3_ann.txt", as.dataframe=T
head(ann)
data <- export.data(data.test3, outfile="Test3_cel.txt", as.dataframe=TRUE, verbose=FALSE)
head(data)
```

---

export.root

*Export data from ROOT file*

---

### Description

Export data as text files directly from a [ROOT](#) file.

### Usage

```
export.root(datafile = character(0), schemefile = character(0), treeset = charac
```

## Arguments

<code>datafile</code>	name of ROOT data file including full path
<code>schemefile</code>	name of ROOT scheme file including full path
<code>treerset</code>	name of subdirectory in ROOT file where trees are stored
<code>treename</code>	name of ROOT tree to export.
<code>treetype</code>	type of tree(s) to export, see <a href="#">validTreetype</a> .
<code>varlist</code>	names of tree leaves to export.
<code>outfile</code>	name of output file.
<code>sep</code>	column separator
<code>as.dataframe</code>	if TRUE a <code>data.frame</code> will be returned.
<code>verbose</code>	logical, if TRUE print status information.

## Details

Export data as text files directly from a [ROOT](#) file.

## Value

If `as.dataframe` is TRUE, the data will be imported into the current R session as `data.frame`. Otherwise, NULL will be returned.

## Author(s)

Christian Stratowa

## See Also

[export](#), [export-methods](#)

## Examples

```
## export data directly from root file
schemefile <- paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
datafile <- paste(.path.package("xps"), "rootdata/DataTest3_cel.root", sep="/")
data <- export.root(datafile, schemefile, "DataSet", "*", "cel", "*", "DataOutFile.txt",
head(data)
```

---

express

*Compute expression levels from raw data*

---

## Description

This function allows to combine different algorithms to compute expression levels, or to return the result for different algorithms only.

**Usage**

```

express(xps.data,
        filename = character(),
        filedir = getwd(),
        tmpdir = "",
        update = FALSE,
        # background correction
        bgcorrect.method = NULL,
        bgcorrect.select = character(),
        bgcorrect.option = character(),
        bgcorrect.params = list(),
        # normalization
        normalize.method = NULL,
        normalize.select = character(),
        normalize.option = character(),
        normalize.logbase = character(),
        normalize.params = list(),
        # expression values
        summarize.method = NULL,
        summarize.select = character(),
        summarize.option = character(),
        summarize.logbase = character(),
        summarize.params = list(),
        # reference values
        reference.index = 0,
        reference.method = "mean",
        reference.params = list(),
        # misc.
        exonlevel = "",
        xps.scheme = NULL,
        add.data = TRUE,
        verbose = TRUE)

xpsPreprocess(object, ...)

```

**Arguments**

<code>xps.data</code>	object of class <code>DataTreeSet</code> .
<code>filename</code>	file name of ROOT data file.
<code>filedir</code>	system directory where ROOT data file should be stored.
<code>tmpdir</code>	optional temporary directory where temporary ROOT files should be stored.
<code>update</code>	logical. If TRUE the existing ROOT data file <code>filename</code> will be updated.
<code>bgcorrect.method</code>	background method to use.
<code>bgcorrect.select</code>	type of probes to select for background correction.
<code>bgcorrect.option</code>	type of background correction to use.
<code>bgcorrect.params</code>	vector of parameters for background method.

<code>normalize.method</code>	normalization method to use.
<code>normalize.select</code>	type of probes to select for normalization.
<code>normalize.option</code>	normalization option.
<code>normalize.logbase</code>	logarithm base as character, one of '0', 'log', 'log2', 'log10'.
<code>normalize.params</code>	vector of parameters for normalization method.
<code>summarize.method</code>	summarization method to use.
<code>summarize.select</code>	type of probes to select for summarization.
<code>summarize.option</code>	option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon arrays only.
<code>summarize.logbase</code>	logarithm base as character, one of '0', 'log', 'log2', 'log10'.
<code>summarize.params</code>	vector of parameters for summarization method.
<code>reference.index</code>	index of reference tree to use, or 0.
<code>reference.method</code>	for <code>refindex=0</code> , either trimmed mean or median of trees.
<code>reference.params</code>	vector of parameters for reference method.
<code>exonlevel</code>	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
<code>xps.scheme</code>	optional alternative <code>SchemeSet</code> .
<code>add.data</code>	logical. If TRUE expression data will be included as slot data.
<code>verbose</code>	logical, if TRUE print status information.
<code>object</code>	object of class <code>DataSet</code> .
<code>...</code>	the arguments described above.

### Details

This function allows to combine different algorithms to compute expression levels, or to return the result for different algorithms only.

`xpsPreprocess` is the `DataSet` method called by function `express`, containing the same parameters.

### Value

An object of type `DataTreeSet` or `ExprTreeSet`.

### Author(s)

Christian Stratowa

**See Also**

[bgcorrect](#), [normalize](#), [summarize](#)

**Examples**

```
## load existing ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## compute rma with a single call to express()
expr.rma <- express(data.test3, "tmp_Test3Exprs", filedir=getwd(), tmpdir="", update=FALSE,
                    bgcorrect.method="rma", bgcorrect.select="none", bgcorrect.option="pmonly:epane
                    normalize.method="quantile", normalize.select="pmonly", normalize.option="trans
                    summarize.method="medianpolish", summarize.select="pmonly", summarize.option="t
                    verbose=FALSE)

## get expression data.frame
expr <- exprs(expr.rma)
head(expr)

## plot expression levels
if (interactive()) {
  boxplot(expr.rma)
  boxplot(log2(expr[, 3:6]))
}

## Not run:
## examples using Affymetrix human tissue dataset (see also xps/examples/script4exon.R)

## example - exon array, e.g. HuEx-1_0-st-v2:
scmdir <- "/Volumes/GigaDrive/CRAN/Workspaces/Schemes"
datdir <- "/Volumes/GigaDrive/CRAN/Workspaces/ROOTData"
scheme.exon <- root.scheme(paste(scmdir, "Scheme_HuEx10stv2r2_na25.root", sep="/"))
data.exon <- root.data(scheme.exon, paste(datdir, "HuTissuesExon_cel.root", sep="/"))

workdir <- "/Volumes/GigaDrive/CRAN/Workspaces/Exon/hutissues/exon"
expr.rma <- express(data.exon, "HuExonExprs", filedir=workdir, tmpdir="", update=F,
                    bgcorrect.method="rma", bgcorrect.select="antigenomic", bgcorrect.option="pmonl
                    normalize.method="quantile", normalize.select="pmonly", normalize.option="trans
                    summarize.method="medianpolish", summarize.select="pmonly", summarize.option="t
                    exonlevel="metacore+affx")

## End(Not run)
```

---

exprs-methods

*Get/Set Expression Values*

---

**Description**

Get/set expression values from/for class ExprTreeSet.

*Usage*

```
exprs(object)
exprs(object, treenames = NULL) <- value
```

**Arguments**

<code>object</code>	object of class <code>ExprTreeSet</code> .
<code>treenames</code>	character vector containing optional tree names to be used as subset.
<code>value</code>	<code>data.frame</code> containing expression values.

**Details**

Get the expression values from slot `data` or set slot `data` to `value`.

Method `exprs` returns the expression values from slot `data` as `data.frame`, while replacement method `exprs<-` allows to replace slot `data` with a `data.frame`.

In order to create an `ExprTreeSet` containing only a subset of slot `data`, first export slot `data` using method `exprs`, create a character vector containing only `treenames` to be used in the subset, and then use replacement method `exprs<-` to replace slot `data` with the subset. Slots `treenames` and `numtrees` will be updated automatically.

Note: When creating character vector `treenames` it is sufficient to use the name part of the tree name w/o the extension.

Note: If you do not want to replace your current object, create first a copy of type `ExprTreeSet` by simply writing `newobj <- oldobj`, and use `newobj` for replacement. This is important since `exprs<-` does also update slots `treenames` and `numtrees` as already mentioned.

**Author(s)**

Christian Stratowa

**See Also**

[pvalData](#), [presCall](#)

**Examples**

```
## Not run:
## load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## create an ExprTreeSet
data.rma <- rma(data.test3, "tmp_TestRMA", tmpdir="", background="pmonly", normalize=TRUE, ver

## get expression values
value <- exprs(data.rma)

## selected treenames only
treenames <- c("TestA2", "TestB1")

## make a copy of your object if you do not want to replace it
subset.rma <- data.rma

## replace slot data with subset
exprs(subset.rma, treenames) <- value
str(subset.rma)
## End(Not run)
```

---

ExprTreeSet-class    *Class ExprTreeSet*

---

## Description

This class provides the link to the [ROOT](#) expression file and the [ROOT](#) trees contained therein. It extends class [ProcesSet](#).

## Objects from the Class

Objects are created using functions [express](#), [summarize](#) or [normalize](#), or the specialized functions [rma](#), [mas5](#) or [mas4](#).

## Slots

**exprtype:** Object of class "character" representing the expression type, i.e. 'rma', 'mas5', 'mas4' or 'custom'.

**normtype:** Object of class "character" representing the normalization type, i.e. 'mean', 'median', 'lowess', 'supsmu'.

**scheme:** Object of class "SchemeTreeSet" providing access to [ROOT](#) scheme file.

**data:** Object of class "data.frame". The data.frame can contain the data (e.g. expression levels) stored in [ROOT](#) data trees.

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

**setname:** Object of class "character" representing the name to the [ROOT](#) file subdirectory where the [ROOT](#) data trees are stored, usually 'PreprocesSet'.

**settype:** Object of class "character" describing the type of treeset stored in setname, usually 'preprocess'.

**rootfile:** Object of class "character" representing the name of the [ROOT](#) data file, including 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 subdirectory setname.

**treenames:** Object of class "list" representing the names of the [ROOT](#) trees stored in subdirectory setname.

## Extends

Class "[ProcesSet](#)", directly. Class "[TreeSet](#)", by class "[ProcesSet](#)", distance 2.

## Methods

**attachExpr** signature(object = "ExprTreeSet"): exports expression trees from [ROOT](#) expression file and saves as data.frame data.

**exprType** signature(object = "ExprTreeSet"): extracts slot exprtype.

**exprType<-** signature(object = "ExprTreeSet", value = "character"): replaces slot exprtype.

**exprs** signature(object = "ExprTreeSet"): extracts the expression data.frame.

**exprs<-** signature(object = "ExprTreeSet", value = "data.frame"): replaces the expression data.frame.

**mvaplot** signature(x = "ExprTreeSet"): creates an MvA-plot.

**normType** signature(object = "ExprTreeSet"): extracts slot normtype.

**normType<-** signature(object = "ExprTreeSet", value = "character"): replaces slot normtype.

**removeExpr** signature(object = "ExprTreeSet"): replaces data.frame data with an empty data.frame of dim(0,0).

**se.exprs** signature(object = "ExprTreeSet"): extracts the standard deviation data.frame.

**xpsNormalize** signature(object = "ExprTreeSet"): applies normalization methods.

**xpsPreFilter** signature(object = "ExprTreeSet"): applies prefiltering methods.

**xpsUniFilter** signature(object = "ExprTreeSet"): applies unfiltering methods.

### Author(s)

Christian Stratowa

### See Also

related classes [DataTreeSet](#), [CallTreeSet](#).

### Examples

```
showClass("ExprTreeSet")
```

---

farms

*Factor Analysis for Robust Microarray Summarization Expression Measure*

---

### Description

This function converts a [DataTreeSet](#) into an [ExprTreeSet](#) using the Factor Analysis for Robust Microarray Summarization (FARMS) method.

### Usage

```
farms(xps.data,
      filename = character(0),
      filedir  = getwd(),
      tmpdir   = "",
      normalize = TRUE,
      weight   = 0.5,
      mu       = 0.0,
      scale    = 1.0,
      tol      = 0.00001,
      cyc      = 100,
      weighted = TRUE,
      version  = "1.3.1",
```



```

option      = "transcript",
exonlevel  = "",
xps.scheme = NULL,
add.data   = TRUE,
verbose    = TRUE)

```

## Arguments

<code>xps.data</code>	object of class <code>DataTreeSet</code> .
<code>filename</code>	file name of ROOT data file.
<code>filedir</code>	system directory where ROOT data file should be stored.
<code>tmpdir</code>	optional temporary directory where temporary ROOT files should be stored.
<code>normalize</code>	logical. If TRUE normalize data using quantile normalization.
<code>weight</code>	hyperparameter, usually set to 0.5 for <code>version="1.3.1"</code> and to 8.0 for <code>version="1.3.0"</code> .
<code>mu</code>	hyperparameter allowing to correct for potential bias.
<code>scale</code>	scaling parameter, usually set to 1.0 for <code>version="1.3.1"</code> and to 2.0 for <code>version="1.3.0"</code> .
<code>tol</code>	termination tolerance for EM algorithm.
<code>cyc</code>	maximum number of cycles of EM algorithm.
<code>weighted</code>	logical, used only with <code>version="1.3.1"</code> . Default is TRUE.
<code>version</code>	version of original farms package. Currently, <code>version="1.3.1"</code> and <code>version="1.3.0"</code> are implemented. Default is <code>version="1.3.1"</code> .
<code>option</code>	option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon arrays only.
<code>exonlevel</code>	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
<code>xps.scheme</code>	optional alternative <code>SchemeTreeSet</code> .
<code>add.data</code>	logical. If TRUE expression data will be included as slot data.
<code>verbose</code>	logical, if TRUE print status information.

## Details

This function computes the FARMS (Factor Analysis for Robust Microarray Summarization) expression measure described in Hochreiter et al. for both expression arrays and exon arrays.

Parameter `version` currently allows the user to choose between the original implementation of FARMS as implemented in package 'farms\_1.3.0' or enhanced FARMS as implemented in package 'farms\_1.3.1'. By default `version="1.3.1"` is used.

Parameter `weight` is a hyperparameter which determines the influence of the prior. For `version="1.3.1"` the value in the range of [0,1].

Parameter `mu` is a hyperparameter which allows to quantify different aspects of potential prior knowledge. Values near zero assume that most genes do not contain a signal and introduce a bias for loading matrix elements near zero.

Parameter `weighted` is a logical and indicates whether a weighted mean or a least square fit is used to summarize the loading matrix. It is applicable only to `version="1.3.1"`.

For exon arrays it is necessary to supply the requested `option` and `exonlevel`.

Following options are valid for exon arrays:

transcript: expression levels are computed for transcript clusters, i.e. probe sets containing the same ‘transcript\_clu  
 exon: expression levels are computed for exon clusters, i.e. probe sets containing the same ‘exon\_id’, where ea  
 probeset: expression levels are computed for individual probe sets, i.e. for each ‘probeset\_id’.

Following `exonlevel` annotations are valid for exon arrays:

<code>core:</code>	probesets supported by RefSeq and full-length GenBank transcripts.
<code>metacore:</code>	core meta-probesets.
<code>extended:</code>	probesets with other cDNA support.
<code>metaextended:</code>	extended meta-probesets.
<code>full:</code>	probesets supported by gene predictions only.
<code>metafull:</code>	full meta-probesets.
<code>affx:</code>	standard AFFX controls.
<code>all:</code>	combination of above (including affx).

Following `exonlevel` annotations are valid for whole genome arrays:

<code>core:</code>	probesets with category ‘unique’, ‘similar’ and ‘mixed’.
<code>metacore:</code>	probesets with category ‘unique’ only.
<code>affx:</code>	standard AFFX controls.
<code>all:</code>	combination of above (including affx).

Exon levels can also be combined, with following combinations being most useful:

<code>exonlevel="metacore+affy":</code>	core meta-probesets plus AFFX controls
<code>exonlevel="core+extended":</code>	probesets with cDNA support
<code>exonlevel="core+extended+full":</code>	supported plus predicted probesets

Exon level annotations are described in the Affymetrix whitepaper `exon_probeset_trans_clust_whitepaper.pdf`: “Exon Probeset Annotations and Transcript Cluster Groupings”.

In order to use an alternative `SchemeTreeSet` set the corresponding `SchemeSet` `xps.scheme`.

## Value

An `ExprTreeSet`

## Note

The expression measure obtained with FARMS is given in linear scale, analogously to the expression measures computed with `mas5` and `rma`.

For the analysis of many exon arrays it may be better to define a `tmpdir`, since this will store only the results in the main file and not e.g. background and normalized intensities, and thus will reduce the file size of the main file. For quantile normalization memory should not be an issue, however DFW depends on RAM unless you are using a temporary file.

## Author(s)

Christian Stratowa

## References

Hochreiter, S., Clevert D.-A., and Obermayer, K. (2006), A new summarization method for Affymetrix probe level data. *Bioinformatics* 22(8):943-949

## See Also

[express](#)

## Examples

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

data.farms <- farms(data.test3, "tmp_Test3FARMS", verbose=FALSE)

## get data.frame
expr.farms <- validData(data.farms)
head(expr.farms)
```

---

fcFilter-methods     *Fold-Change Filter*

---

## Description

This method initializes the Fold-Change Filter.  
The fold-change is determined by the mean value of group 2 divided by the mean value of group 1.  
The Fold-Change Filter flags all rows with: `flag = (fc >= cutoff)`

### Usage

```
fcFilter(object)
fcFilter(object, value) <-
```

## Arguments

<code>object</code>	object of class <code>UniFilter</code> .
<code>value</code>	numeric vector <code>c(cutoff, direction)</code>

## Details

The method `fcFilter` initializes the following parameters:

<code>cutoff:</code>	the cutoff level for the filter.
<code>direction:</code>	<code>direction="both"</code> (default): select up and downregulated genes. <code>direction="up"</code> : select upregulated genes only. <code>direction="down"</code> : select downregulated genes only.

## Value

An initialized `UniFilter` object.

**Author(s)**

Christian Stratowa

**Examples**

```
unifltr <- UniFilter()
fcFilter(unifltr) <- c(1.5, "both")
str(unifltr)
```

---

Filter-class

*Base Class Filter*

---

**Description**

Base class for classes [PreFilter](#) and [UniFilter](#).

**Slots**

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

**Methods**

**numberFilters** signature(object = "Filter"): number of filters applied.

**Author(s)**

Christian Stratowa

**See Also**

related classes [PreFilter](#), [UniFilter](#).

**Examples**

```
showClass("Filter")
```

---

FilterTreeSet-class

*Class FilterTreeSet*

---

**Description**

This class provides the link to the [ROOT](#) filter file and the [ROOT](#) trees contained therein. It extends class [ProcesSet](#).

**Objects from the Class**

Objects are currently created using function [prefilter](#).

**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.
- scheme:** Object of class "SchemeTreeSet" providing access to [ROOT](#) scheme file.
- data:** Object of class "data.frame". The data.frame contains the data of the filter stored in [ROOT](#) filter trees.
- params:** Object of class "list" representing relevant parameters.
- setname:** Object of class "character" representing the name to the [ROOT](#) file subdirectory where the [ROOT](#) trees are stored, currently 'PreFilterSet'.
- settype:** Object of class "character" describing the type of treeset stored in setname, currently 'prefilter'.
- rootfile:** Object of class "character" representing the name of the [ROOT](#) file, including 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 subdirectory setname.
- treenames:** Object of class "list" representing the names of the [ROOT](#) trees stored in subdirectory setname.

**Extends**

Class "[ProcesSet](#)", directly. Class "[TreeSet](#)", by class "[ProcesSet](#)", distance 2.

**Methods**

- callTreeset** signature(object = "FilterTreeSet"): extracts slot callset.
- exprTreeset** signature(object = "FilterTreeSet"): extracts slot exprset.
- getTreeData** signature(object = "FilterTreeSet"): exports tree data and returns a data.frame.
- validData** signature(object = "FilterTreeSet"): extracts data.frame data.

**Author(s)**

Christian Stratowa

**See Also**

related classes [AnalysisTreeSet](#).

**Examples**

```
showClass("FilterTreeSet")
```

## Description

This method initializes the Gap Filter.

The `gapFilter` looks for genes that might usefully discriminate between two groups. To do this we look for a gap in the ordered expression values. The gap should come in the central portion, thus a parameter `window` is defined to exclude jumps in the initial window values and the final window values.

The Gap Filter flags all rows with: `flag = ((gap[i+1] - gap[i])/mean >= cutoff)`

```
gapFilter(object)
gapFilter(object, value)<-
```

## Arguments

<code>object</code>	object of class <code>PreFilter</code> .
<code>value</code>	numeric vector <code>c(cutoff, window, trim, epsilon)</code> .

## Details

The method `gapFilter` initializes the following parameters:

<code>cutoff</code> :	the cutoff level for the filter.
<code>window</code> :	trim value for the ordered expression levels (default is <code>window=0.05</code> ).
<code>trim</code> :	the trim value for trimmed mean (default is <code>trim=0</code> ).
<code>epsilon</code> :	value to replace mean (default is <code>epsilon=0.01</code> ): <code>epsilon &gt; 0</code> : replace mean=0 with <code>epsilon</code> . <code>epsilon = 0</code> : always set mean=1.

Note, that for `epsilon = 0` the filter flags all rows with: `(gap[i+1] - gap[i]) >= cutoff`

## Value

An initialized `PreFilter` object.

## Author(s)

Christian Stratowa

## Examples

```
prefltr <- PreFilter()
gapFilter(prefltr) <- c(0.3,0.05,0.0,0.01)
str(prefltr)
```

---

getChipName	<i>Get Chip Name</i>
-------------	----------------------

---

**Description**

Get chip name from ROOT scheme file.

**Usage**

```
getChipName(rootfile)
```

**Arguments**

rootfile      name of ROOT scheme file, including full path.

**Details**

Extracts the chip name directly from ROOT scheme file `rootfile`.

**Value**

a character with the chip name.

**Author(s)**

Christian Stratowa

**See Also**

[getChipType](#), [getNameType](#)

**Examples**

```
## correct usage
getChipName(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/"))
## incorrect usage
getChipName(paste(.path.package("xps"), "rootdata/DataTest3_cel.root", sep="/"))
```

---

getChipType	<i>Get Chip Type</i>
-------------	----------------------

---

**Description**

Get chip type from ROOT scheme file.

**Usage**

```
getChipType(rootfile)
```

**Arguments**

`rootfile`      name of ROOT scheme file, including full path.

**Details**

Extracts the chip type directly from [ROOT](#) scheme file `rootfile`.

**Value**

a character with the chip type, either 'GeneChip' or 'ExonChip'.

**Author(s)**

Christian Stratowa

**See Also**

[getChipName](#), [getNameType](#)

**Examples**

```
## correct usage
getChipType(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/"))
## incorrect usage
getChipType(paste(.path.package("xps"), "rootdata/DataTest3_cel.root", sep="/"))
```

---

`getDatatype`      *Get Data Type*

---

**Description**

Get data type corresponding to tree type.

**Usage**

```
getDatatype(treetype)
```

**Arguments**

`treetype`      tree type.

**Details**

Get data type corresponding to tree type. Valid tree types are described in [validTreetype](#).

**Value**

a character with the correct data type, i.e. 'rawdata', 'preprocess' or 'normation'.

**Author(s)**

Christian Stratowa



**See Also**

[type2Exten](#), [validTreetype](#)

**Examples**

```
getDatatype("cel")
getDatatype("tbw")
```

---

getNameType	<i>Get Chip Name and Type</i>
-------------	-------------------------------

---

**Description**

Get chip name and type from ROOT scheme file.

**Usage**

```
getNameType(rootfile)
```

**Arguments**

rootfile      name of ROOT scheme file, including full path.

**Details**

Extracts the chip name and type directly from [ROOT](#) scheme file `rootfile`.

**Value**

a list with parameters:

chipname      chip name.  
chiptype      chip type, either 'GeneChip' or 'ExonChip'.

**Author(s)**

Christian Stratowa

**See Also**

[getChipName](#), [getChipType](#)

**Examples**

```
## correct usage
getNameType(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/"))
## incorrect usage
getNameType(paste(.path.package("xps"), "rootdata/DataTest3_cel.root", sep="/"))
```

---

getNumberTrees      *Get Number of Trees*

---

### Description

Get number of trees stored in a ROOT file.

### Usage

```
getNumberTrees(rootfile, treetype = "*", setname = NULL)
```

### Arguments

rootfile      name of ROOT file, including full path.  
treetype      tree type.  
setname      name of ROOT subdirectory containing trees.

### Details

Extracts the number of trees of `treetype` stored in `ROOT` file `rootfile`. Valid tree types are listed in [validTreetype](#). For `treetype="*"` the total number of trees in `rootfile` are returned. If `setname` is provided, only trees in subdirectory `setname` are counted.

### Value

Number of trees.

### Author(s)

Christian Stratowa

### Examples

```
getNumberTrees(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/"))  
getNumberTrees(paste(.path.package("xps"), "rootdata/DataTest3_cel.root", sep="/"))
```

---

getProbeInfo      *Get Probe Information*

---

### Description

Get GeneChip probe information from root scheme file.

### Usage

```
getProbeInfo(rootfile)
```

**Arguments**

`rootfile` name of ROOT scheme file, including full path.

**Details**

Extracts GeneChip probe information directly from ROOT scheme file `rootfile`.

**Value**

a list with parameters:

<code>nrows</code>	physical number of rows in the array.
<code>ncols</code>	physical number of columns in the array.
<code>nprobes</code>	number of probes on the array.
<code>ncontrols</code>	number of controls on the array.
<code>ngenes</code>	number of genes on the array.
<code>nunits</code>	number of units on the array.
<code>nprobesets</code>	umber of probesets on the array.
<code>naffx</code>	number of AFFX controls on the array.

**Author(s)**

Christian Stratowa

**Examples**

```
getProbeInfo(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/"))
```

---

```
getTreeData-methods
```

*Export Tree Data*

---

**Description**

Exports tree data from ROOT data file and and saves as `data.frame`.

*Usage*

```
getTreeData(object, treetype = "cel", varlist = "fInten")
```

**Arguments**

<code>object</code>	Object of class "ProcessSet".
<code>treetype</code>	type of tree to export, see <a href="#">validTreetype</a>
<code>varlist</code>	names of tree leaves to export.

**Details**

Exports tree leaves from ROOT data file and and saves as `data.frame`.

**Value**

A `data.frame`.

**Author(s)**

Christian Stratowa

**See Also**

`export`

---

getTreeNames	<i>Get Tree Names</i>
--------------	-----------------------

---

**Description**

Get tree names stored in a ROOT file.

**Usage**

```
getTreeNames(rootfile, treetype = "*", setname = NULL, gettitle = FALSE)
```

**Arguments**

<code>rootfile</code>	name of ROOT file, including full path.
<code>treetype</code>	tree type.
<code>setname</code>	name of ROOT subdirectory containing trees.
<code>gettitle</code>	If TRUE the titles of the trees will be returned.

**Details**

Extracts the tree names of `treetype` stored in ROOT file `rootfile`.

Valid tree types are listed in `validTreetype`. For `treetype="*"` names for all trees in `rootfile` are returned.

If `setname` is provided, only tree names in subdirectory `setname` are returned.

**Value**

A vector of tree names. For `gettitle=TRUE` a vector of tree titles.

**Author(s)**

Christian Stratowa

**Examples**

```
getTreeNames(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/"))
getTreeNames(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/"), "scm")
getTreeNames(paste(.path.package("xps"), "rootdata/DataTest3_cel.root", sep="/"))
```

---

*highFilter-methods Upper Threshold Filter*

---

**Description**

This method initializes the Upper Threshold Filter.

The `cutoff` value defines the upper threshold for allowed expression levels. If e.g. the number of samples exceeding this cutoff value is greater than `parameter` then the corresponding dataframe row is flagged, i.e. `flag = 0`.

The Upper Threshold Filter flags all rows with: `flag = (sum(expression[i] <= cutoff) >= parameter)`

*Usage*

```
highFilter(object)
highFilter(object, value)<-
```

**Arguments**

<code>object</code>	object of class <code>PreFilter</code> .
<code>value</code>	character vector <code>c(cutoff, parameter, condition)</code> .

**Details**

The method `highFilter` initializes the following parameters:

<code>cutoff</code> :	the upper threshold level for the filter.
<code>parameter</code> :	this value depends on the condition used:
<code>condition</code> :	<code>condition="samples"</code> : number of samples (default);
	<code>condition="percent"</code> : percent of samples.
	<code>condition="mean"</code> : mean value of samples.
	<code>condition="percentile"</code> : percentile of samples.

**Value**

An initialized `PreFilter` object.

**Author(s)**

Christian Stratowa

**Examples**

```
prefltr <- PreFilter()
highFilter(prefltr) <- c(14.5, 75.0, "percent")
str(prefltr)
```

**Description**

Plot the density estimates for each sample.

*Usage*

```
hist(x, which = "", size = 0, transfo = log2, ylab = "density", xlab = "log intensity", type = "l", col = 1:6, ...)
```

**Arguments**

x	object of class <a href="#">DataTreeSet</a> or <a href="#">ExprTreeSet</a> .
which	type of probes to be used, for details see <a href="#">validData</a> .
size	length of sequence to be generated as subset.
transfo	a valid function to transform the data, usually “log2”, or “0”.
ylab	a title for the y axis.
xlab	a title for the x axis.
type	type for the plot.
col	colors to use for the different arrays.
...	optional arguments to be passed to <code>plot</code> .

**Details**

Plots the non-parametric density estimates using values contained in the columns of slot `data`.

For a [DataTreeSet](#) object, data must first be attached using method [attachInten](#).

**Note**

For exon array raw data only a limited number of samples can be displayed as density plot due to memory limitations. To display all samples it is proposed to use function [root.density](#) instead.

**Author(s)**

Christian Stratowa

**Examples**

```
## load existing ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## need to attach scheme mask and probe intensities
data.test3 <- attachMask(data.test3)
data.test3 <- attachInten(data.test3)

if (interactive()) {
  hist(data.test3)
}
```

```
## optionally remove mask and data to free memory
data.test3 <- removeInten(data.test3)
data.test3 <- removeMask(data.test3)
```

---

 image.dev

*Image for Device*


---

## Description

Creates an image for each sample for the selected device.

## Usage

```
image.dev(x, bg = FALSE, transfo = log2, col = gray((0:64)/64), names = "namepart")
```

## Arguments

x	object of class <a href="#">DataTreeSet</a> .
bg	logical. If FALSE, intensities from slot <code>data</code> will be used; if TRUE, background intensities from slot <code>bgrd</code> will be used.
transfo	a valid function to transform the data, usually “log2”, or “0”.
col	color range for intensities.
names	optional vector of sample names.
xlab	a title for the x axis.
ylab	a title for the y axis.
mar	plot margins.
dev	graphics device to plot to, i.e. one of “screen”, “jpeg”, “png”, “pdf” or “ps”.
outfile	the name of the output file.
w	the width of the device in pixels.
h	the height of the device in pixels.
...	optional arguments to be passed to <code>image</code> .

## Details

Creates an image for each array for the selected graphics device.

For `bgrd=TRUE` the distribution of the computed background intensities will be shown; this can be useful to see potential density gradients caused by hybridization conditions. For the computation of background intensities see function [bgcorrect](#); it is suggested to use [bgcorrect.mas4](#) to identify density gradients.

For `names=NULL` full column names of slot `data` will be displayed while for `names="namepart"` column names will be displayed without name extension. If `names` is a vector of column names, only these columns will be displayed as image.

## Note

Data must first be attached to class [DataTreeSet](#) using method [attachInten](#).

**Author(s)**

Christian Stratowa

**See Also**[image-methods](#), [image](#)**Examples**

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## need to attach scheme mask and data
data.test3 <- attachMask(data.test3)
data.test3 <- attachInten(data.test3)

if (interactive()) {
  image.dev(data.test3)
}

## to avoid memory consumption of R remove data:
data.test3 <- removeInten(data.test3)
data.test3 <- removeMask(data.test3)
```

---

image-methods

*Display an Image*

---

**Description**

Creates an image for each sample.

*Usage*

```
image(x, bg = FALSE, transfo = log2, col = gray((0:64)/64), names
= "namepart", xlab = "", ylab = "", ...)
```

**Arguments**

x	object of class <a href="#">DataTreeSet</a> .
bg	logical. If FALSE, intensities from slot data will be used; if TRUE, background intensities from slot bgrd will be used.
transfo	a valid function to transform the data, usually “log2”, or “0”.
col	color range for intensities.
names	optional vector of sample names.
...	optional arguments to be passed to <code>image</code> .



**Details**

Creates an image for each array. For `bgrd=TRUE` the distribution of the computed background intensities will be shown; this can be useful to see potential density gradients caused by hybridization conditions. For the computation of background intensities see function `bgcorrect`; it is suggested to use `bgcorrect.mas4` to identify density gradients.

For `names=NULL` full column names of slot data will be displayed while for `names="namepart"` column names will be displayed without name extension. If `names` is a vector of column names, only these columns will displayed as image.

**Note**

Data must first be attached to class `DataTreeSet` using method `attachInten`.

**Author(s)**

Christian Stratowa

**See Also**

`image.dev`, `image`

---

import.data

---

*Import CEL files into a DataTreeSet*


---

**Description**

Import the Affymetrix CEL files into a ROOT file and create S4 class `DataTreeSet`

**Usage**

```
import.data(xps.scheme,
            filename = character(0),
            filedir  = getwd(),
            celdir   = NULL,
            celfiles = "*",
            celnames = NULL,
            project  = NULL,
            verbose  = TRUE)
```

**Arguments**

<code>xps.scheme</code>	a <code>SchemeTreeSet</code> containing the correct scheme for the CEL-files
<code>filename</code>	file name of ROOT data file.
<code>filedir</code>	system directory where ROOT data file should be stored.
<code>celdir</code>	system directory containing the CEL-files for corresponding scheme.
<code>celfiles</code>	optional vector of CEL-files to be imported.
<code>celnames</code>	optional vector of names which should replace the CEL-file names.
<code>project</code>	optional class <code>ProjectInfo</code> .
<code>verbose</code>	logical, if TRUE print status information.

## Details

`import.data` is used to import CEL-files from directory `celdir` into a [ROOT](#) data file. To import only a subset of CEL-files, list these CEL-files as vector `celfiles`.

To import CEL-files from different directories, vector `celfiles` must contain the full path for each CEL-file and `celdir` must be `celdir=NULL`.

Currently, the following types of Affymetrix CEL-files can be imported: text (version 3), xml, binary (xda), generic (agcc,calvin)

An S4 class `DataTreeSet` will be created, serving as R wrapper to the [ROOT](#) data file `filename`.

Use function `root.data` to access the [ROOT](#) data file from new R sessions to avoid creating a new [ROOT](#) data file for every session.

## Value

A `DataTreeSet` object.

## Note

As mentioned above, use function `root.data` to access the [ROOT](#) data file from new R sessions to avoid creating a new [ROOT](#) data file for every R session.

Do not separate `filename` of [ROOT](#) files with dots, use underscores, e.g. do not use `filename="Data.Test3"` but use `filename="Data_Test3"` or `filename="DataTest3"` instead.

To every [ROOT](#) data file the extension “\_cel” is attached to `filename` to easily recognize [ROOT](#) data files containing the raw CEL data, e.g. for `filename="Data_Test3"` the final name is “Data\_Test3\_cel.root”. Extension “root” is added automatically, so that [ROOT](#) is able to recognize the file as [ROOT](#) file.

Once a [ROOT](#) file is created it can not be overwritten, it must be deleted manually first. Only [ROOT](#) files called “tmp” or with `filename` starting with “tmp\_” will be re-created automatically.

If CEL-file names contain dots and/or colons as characters, these characters will be replaced by underscores. It is recommended to use parameter `celnames` to create shorter CEL names and to replace special characters.

## Author(s)

Christian Stratowa

## See Also

[root.data](#), [DataTreeSet](#)

## Examples

```
## get scheme and import CEL-files from package
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- import.data(scheme.test3, "tmp_data_test3", celdir=paste(.path.package("xps")
unlist(treeNames(data.test3))

## import only subset of CEL-files
subdata.test3 <- import.data(scheme.test3, "tmpdt_data_test3", celdir=paste(.path.package("
celfiles=c("TestA1.CEL", "TestB2.CEL"), verbose=FALSE)
unlist(treeNames(subdata.test3))
```

---

import.exon.scheme *Import CLF, PGF and annotation files into a SchemeTreeSet*

---

## Description

Import the Affymetrix CLF, PGF, and probeset and transcript annotation files into a ROOT file and create S4 class SchemeTreeSet

## Usage

```
import.exon.scheme(filename = character(0),
                  filedir  = getwd(),
                  layoutfile = character(0),
                  schemefile = character(0),
                  probeset  = character(0),
                  transcript = character(0),
                  control   = "",
                  add.mask  = FALSE,
                  verbose   = TRUE)
```

## Arguments

filename	file name of ROOT scheme file.
filedir	system directory where ROOT scheme file should be stored.
layoutfile	name of CLF-file, including full path.
schemefile	name of PGF-file, including full path.
probeset	name of probeset annotation-file, including full path.
transcript	name of transcript annotation-file, including full path.
control	optional name of controls.ps-file, including full path.
add.mask	logical. If TRUE mask information will be included as slot mask.
verbose	logical, if TRUE print status information.

## Details

import.exon.scheme is used to import all information for an Affymetrix exon array into a [ROOT](#) scheme file, including CLF and PGF-files, and the current Afymetrix probeset and transcript annotation files.

An S4 class [SchemeTreeSet](#) will be created, serving as R wrapper to the [ROOT](#) scheme file filename.

Since a new [ROOT](#) scheme file needs only to be created when new annotation files are available from the Affymetrix website, it is recommended to store all [ROOT](#) scheme files in a commonly accessible system directory `filedir`.

Use function [root.scheme](#) to access the [ROOT](#) scheme file from new R sessions to avoid creating a new [ROOT](#) scheme file for every session.

## Value

A [SchemeTreeSet](#) object.

**Warning**

The current version of 'xps' should be able to import all Affymetrix exon array annotation files up to November 2008. However, since Affymetrix is still changing the headers and/or columns of the annotation files, future annotation files may require adaptation of the source code, thus the current version of 'xps' may not be able to read these files.

**Note**

As mentioned above, use function `root.scheme` to access the `ROOT` scheme file from new R sessions to avoid creating a new `ROOT` scheme file for every R session.

Do not separate filename of `ROOT` files with dots, use underscores, e.g. do not use `filename="Scheme.HuEx10stv2r2.na27"` but use `filename="Scheme_HuEx10stv2r2_na27"` instead. Extension "root" is added automatically, so that `ROOT` is able to recognize the file as `ROOT` file.

Do not set `add.mask=TRUE` unless you know that your computer has sufficient RAM.

Do not add item `control` unless you want to use one of the old annotation files where the probeset annotation file does not contain the `AFFX` controls.

**Author(s)**

Christian Stratowa

**See Also**

`import.expr.scheme`, `root.scheme`, `SchemeTreeSet`

**Examples**

```
## Not run:
## define paths
scmdir <- "/common/path/schemes"
libdir <- "/my/path/Affy/libraryfiles"
anndir <- "/my/path/Affy/Annotation"

## create scheme for HuEx-1_0-st-v2.r2 Exon array
scheme.huex10stv2r2.na27 <- import.exon.scheme("Scheme_HuEx10stv2r2_na27",filedir=scmdir,
layoutfile=paste(libdir,"HuEx-1_0-st-v2_libraryfile/HuEx-1_0-
schemefile=paste(libdir,"HuEx-1_0-st-v2_libraryfile/HuEx-1_0-
probeset=paste(anndir,"HuEx-1_0-st-v2.na27.hg18.probeset.csv"
transcript=paste(anndir,"HuEx-1_0-st-v2.na27.hg18.transcript.

## access ROOT scheme file from new R session
scheme.huex10stv2r2 <- root.scheme(paste(scmdir,"Scheme_HuEx10stv2r2_na27.root",sep="/"))

## create scheme for HuGene-1_0-st-v1.r4 as exon array
scheme.hugene10stv1r4.na27 <- import.exon.scheme("Scheme_HuGene10stv1r4_na27_2",filedir=s
layoutfile=paste(libdir,"HuGene-1_0-st-v1.r4.analysis-lib-f
schemefile=paste(libdir,"HuGene-1_0-st-v1.r4.analysis-lib-f
probeset=paste(anndir,"HuGene-1_0-st-v1.na27.2.hg18.probes
transcript=paste(anndir,"HuGene-1_0-st-v1.na27.hg18.transcr

## access ROOT scheme file from new R session
scheme.hugene10stv1r4 <- root.scheme(paste(scmdir,"Scheme_HuGene10stv1r4_na27_2.root",sep

## create scheme for HuEx-1_0-st-v2.r2 Exon array with the old annotation file
```

```
scheme.huex10stv2r2.old <- import.exon.scheme("Scheme_HuEx10stv2r2_old",filedir=scmdir,
layoutfile=paste(libdir,"HuEx-1_0-st-v2_libraryfile/HuEx-1_0-s
schemefile=paste(libdir,"HuEx-1_0-st-v2_libraryfile/HuEx-1_0-s
probeset=paste(anndir,"HuEx-1_0-st-probeset-annot.csv",sep="/"
transcript=paste(anndir,"HuEx-1_0-st-transcript-annot.csv",sep="/"
control=paste(libdir,"HuEx-1_0-st-v2_libraryfile/HuEx-1_0-st-r

## End(Not run)
```

---

import.expr.scheme *Import CDF, probe and annotation files into a SchemeTreeSet*

---

## Description

Import the Affymetrix CDF, probe and annotation files into a ROOT file and create S4 class SchemeTreeSet

## Usage

```
import.expr.scheme(filename = character(0),
filedir = getwd(),
schemefile = character(0),
probefile = character(0),
annotfile = character(0),
chipname = NULL,
add.mask = FALSE,
verbose = TRUE)
```

## Arguments

filename	file name of ROOT scheme file.
filedir	system directory where ROOT scheme file should be stored.
schemefile	name of CDF-file, including full path.
probefile	name of probe-file, including full path.
annotfile	name of annotation-file, including full path.
chipname	optional chip name when using an alternative CDF-file.
add.mask	logical. If TRUE mask information will be included as slot mask.
verbose	logical, if TRUE print status information.

## Details

import.expr.scheme is used to import all information for an Affymetrix expression array into a [ROOT](#) scheme file, including CDF-file, the corresponding probe file, and the current Affymetrix annotation file.

Usually, chipname is extracted from the name of the CDF-file, however, when using an alternative CDF-file, e.g. from BrainArray or AffyProbeMiner, a chipname must be supplied which starts with (or contains) the exact Affymetrix chip name.

An S4 class [SchemeTreeSet](#) will be created, serving as R wrapper to the [ROOT](#) scheme file filename.

Since a new `ROOT` scheme file needs only to be created when a new annotation file is available from the Affymetrix website, it is recommended to store all `ROOT` scheme files in a commonly accessible system directory `filedir`.

Use function `root.scheme` to access the `ROOT` scheme file from new R sessions to avoid creating a new `ROOT` scheme file for every session.

### Value

A `SchemeTreeSet` object.

### Note

As mentioned above, use function `root.scheme` to access the `ROOT` scheme file from new R sessions to avoid creating a new `ROOT` scheme file for every R session.

Do not separate `filename` of `ROOT` files with dots, use underscores, e.g. do not use `filename="Scheme.Test3.na27"` but use `filename="Scheme_Test3_na27"` or simply `filename="SchemeTest3na27"` instead. Extension “root” is added automatically, so that `ROOT` is able to recognize the file as `ROOT` file.

For a few probesets, parsing the Affymetrix annotation files will provide ambiguous results. Setting `verbose=11` will list these probesets.

### Author(s)

Christian Stratowa

### See Also

`import.exon.scheme`, `import.genome.scheme`, `root.scheme`, `SchemeTreeSet`

### Examples

```
## Not run:
## define paths
smdir <- "/common/path/schemes"
libdir <- "/my/path/Affy/libraryfiles"
anndir <- "/my/path/Affy/Annotation"

## create scheme for Test3 GeneChip
scheme.test3.na27 <- import.expr.scheme("Scheme_Test3_na27",filedir=smdir,
  schemefile=paste(libdir,"Test3.CDF",sep="/"),
  probefile=paste(libdir,"Test3_probe.tab",sep="/"),
  annotfile=paste(anndir,"Test3.na27.annot.csv",sep="/"))

## access ROOT scheme file from new R session
scheme.test3 <- root.scheme(paste(smdir,"Scheme_Test3_na27.root",sep="/"))

## create scheme for HG-U133_Plus_2 GeneChip
scheme.hgu133p2.na27 <- import.expr.scheme("Scheme_HGU133p2_na27",filedir=smdir,
  schemefile=paste(libdir,"HG-U133_Plus_2.cdf",sep="/"),
  probefile=paste(libdir,"HG-U133-PLUS_probe.tab",sep="/"),
  annotfile=paste(anndir,"HG-U133_Plus_2.na27.annot.csv",sep="/"))

## access ROOT scheme file from new R session
scheme.hgu133p2 <- root.scheme(paste(smdir,"Scheme_HGU133p2_na27.root",sep="/"))
## End(Not run)
```

---

```
import.genome.scheme
```

*Import CLF, PGF and annotation files into a SchemeTreeSet*

---

## Description

Import the Affymetrix CLF, PGF and transcript annotation files into a ROOT file and create S4 class SchemeTreeSet

## Usage

```
import.genome.scheme(filename = character(0),
                     filedir  = getwd(),
                     layoutfile = character(0),
                     schemefile = character(0),
                     transcript = character(0),
                     add.mask   = FALSE,
                     verbose    = TRUE)
```

## Arguments

filename	file name of ROOT scheme file.
filedir	system directory where ROOT scheme file should be stored.
layoutfile	name of CLF-file, including full path.
schemefile	name of PGF-file, including full path.
transcript	name of transcript annotation-file, including full path.
add.mask	logical. If TRUE mask information will be included as slot mask.
verbose	logical, if TRUE print status information.

## Details

`import.genome.scheme` is used to import all information for an Affymetrix whole genome array into a [ROOT](#) scheme file, including CLF and PGF-files, and the current Affymetrix transcript annotation files.

An S4 class `SchemeTreeSet` will be created, serving as R wrapper to the [ROOT](#) scheme file `filename`.

Since a new [ROOT](#) scheme file needs only to be created when new annotation files are available from the Affymetrix website, it is recommended to store all [ROOT](#) scheme files in a commonly accessible system directory `filedir`.

Use function `root.scheme` to access the [ROOT](#) scheme file from new R sessions to avoid creating a new [ROOT](#) scheme file for every session.

## Value

A `SchemeTreeSet` object.

**Warning**

The current version of 'xps' is able to import all Affymetrix genome array annotation files up to November 2008, i.e. all files of release 3 (r3) and earlier. However, in January 2009 Affymetrix has updated all CLF, PGF and annotation files to release 4 (r4) and added a new probeset annotation file, thus in effect changing the whole genome arrays to exon arrays!

Thus, for release 4 (r4) files, function `import.genome.scheme` can no longer be used, but you must use function `import.exon.scheme` instead (see examples).

**Note**

As mentioned above, use function `root.scheme` to access the `ROOT` scheme file from new R sessions to avoid creating a new `ROOT` scheme file for every R session.

Do not separate `filename` of `ROOT` files with dots, use underscores, e.g. do not use `filename="Scheme.HuGene10stvlr3.na27"` but use `filename="Scheme_HuGene10stvlr3_na27"` instead. Extension "root" is added automatically, so that `ROOT` is able to recognize the file as `ROOT` file.

Do not set `add.mask=TRUE` unless you know that your computer has sufficient RAM.

Do not add item `control` unless you want to use one of the old annotation files where the probeset annotation file does not contain the `AFFX` controls.

**Author(s)**

Christian Stratowa

**See Also**

`import.exon.scheme`, `root.scheme`, `SchemeTreeSet`

**Examples**

```
## Not run:
## define paths
smdir <- "/common/path/schemes"
libdir <- "/my/path/Affy/libraryfiles"
anndir <- "/my/path/Affy/Annotation"

## create scheme for HuGene-1_0-st-v1 whole genome array
scheme.hugene10stvlr3.na27 <- import.genome.scheme("Scheme_HuEx10stvlr3_na27",
  filedir=smdir,
  layoutfile=paste(libdir,"HuGene-1_0-st-v1.r3.analysis_libraryfiles.layout",
  sep="/"),
  schemefile=paste(libdir,"HuGene-1_0-st-v1.r3.analysis_libraryfiles.scheme",
  sep="/"),
  transcript=paste(anndir,"HuGene-1_0-st-v1.na27.hg18.transcript",
  sep="/"))

## access ROOT scheme file from new R session
scheme.hugene10stvlr3 <- root.scheme(paste(smdir,"Scheme_HuEx10stvlr3_na27.root",
  sep="/"))
## End(Not run)
```



ini.call

*Informative/Non-Informative Call***Description**

Computes the Informative/Non-Informative Call for the exclusion of non-informative probe sets.

**Usage**

```
ini.call(xps.data,
        filename = character(0),
        filedir  = getwd(),
        tmpdir   = "",
        weight   = 0.5,
        mu       = 0.0,
        scale    = 1.0,
        tol      = 0.00001,
        cyc      = 100,
        alpha1   = 0.4,
        alpha2   = 0.6,
        version  = "1.3.1",
        option   = "transcript",
        exonlevel = "",
        xps.scheme = NULL,
        add.data  = TRUE,
        verbose  = TRUE)
```

```
xpsINICall(object, ...)
```

**Arguments**

xps.data	object of class DataTreeSet.
filename	file name of ROOT data file.
filedir	system directory where ROOT data file should be stored.
tmpdir	optional temporary directory where temporary ROOT files should be stored.
weight	hyperparameter, usually set to 0.5 for version="1.3.1" and to 8.0 for version="1.3.0".
mu	hyperparameter allowing to correct for potential bias.
scale	scaling parameter, usually set to 1.0 for version="1.3.1" and to 2.0 for version="1.3.0".
tol	termination tolerance for EM algorithm.
cyc	maximum number of cycles of EM algorithm.
alpha1	a significance threshold in (0,alpha2).
alpha2	a significance threshold in (alpha1,1.0).
version	version of original farms package. Currently, version="1.3.1" and version="1.3.0" are implemented. Default is version="1.3.1".
option	option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon arrays only.

exonlevel	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
xps.scheme	optional alternative SchemeTreeSet.
add.data	logical. If TRUE call data will be added to slots data and detcall.
verbose	logical, if TRUE print status information.
object	object of class DataTreeSet.
...	the arguments described above.

### Details

In contrast to `mas5.call` this function quantifies the signal-to-noise ratio for each probe set, as described in Talloen et al. Thus, the returned p-values and detection calls have a different meaning:

The p-value that is returned estimates the signal-to-noise ratio (SNR):

P-values (SNR) of less than 0.5 indicate that there is more signal than noise and the corresponding genes are considered to be 'informative' for further analysis. In contrast, values greater than 0.5 indicate 'non-informative' genes.

The informative call is computed by thresholding the p-value as in:

```
call "P" if p-value < alpha1
call "M" if alpha1 <= p-value < alpha2
call "A" if alpha2 <= p-value
```

Here "P" should be considered as informative "I", "M" as marginally informative, and "A" as non-informative "NI".

The defaults for `alpha1=0.4` and `alpha2=0.6` are set to allow "M" calls. In order to get the same results as package 'farms\_1.3.1', you need to set `alpha1=0.5` and `alpha2=0.5`.

For exon/genome arrays it is necessary to supply `option` and `exonlevel`.

Following options are valid for exon arrays only:

```
transcript: expression levels are computed for transcript clusters, i.e. probe sets containing the same 'transcript_cluster_id'
exon:       expression levels are computed for exon clusters, i.e. probe sets containing the same 'exon_id', where each probe set is associated with a single exon
probeset:  expression levels are computed for individual probe sets, i.e. for each 'probeset_id'.
```

Following `exonlevel` annotations are valid for exon arrays:

```
core:           probesets supported by RefSeq and full-length GenBank transcripts.
metacore:      core meta-probesets.
extended:      probesets with other cDNA support.
metaextended:  extended meta-probesets.
full:          probesets supported by gene predictions only.
metafull:      full meta-probesets.
ambiguous:     ambiguous probesets only.
affx:          standard AFFX controls.
all:           combination of above.
```

Following `exonlevel` annotations are valid for whole genome arrays:

```
core:           probesets with category 'unique' and 'mixed'.
metacore:      probesets with category 'unique' only.
affx:          standard AFFX controls.
all:           combination of above.
```

Exon levels can also be combined, with following combinations being most useful:

exonlevel="metacore+affy":	core meta-probesets plus AFFX controls
exonlevel="core+extended":	probesets with cDNA support
exonlevel="core+extended+full":	supported plus predicted probesets

Exon level annotations are described in the Affymetrix whitepaper 'exon\_probeset\_trans\_clust\_whitepaper.pdf'.

In order to use an alternative [SchemeTreeSet](#) set the corresponding SchemeTreeSet `xps.scheme`.

`xpsINICall` is the `DataTreeSet` method called by function `ini.call`, containing the same parameters.

### Value

A [CallTreeSet](#)

### Note

Since I/NI-calls distinguish only between informative and non-informative genes, the calls are identical for all samples.

### Author(s)

Christian Stratowa

### References

Talloon, W., Clevert D.-A., Hochreiter, S., Amaratunga, D., Bijmens, J., Kass, S., and Gohlmann, H.W.H. (2006), I/NI-calls for the exclusion of non-informative genes: a highly effective filtering tool for microarray data. *Bioinformatics* 23(21):2897-2902

### See Also

[farms](#), [mas5.call](#)

### Examples

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## I/NI call
call.ini <- ini.call(data.test3, "tmp_Test3INI", verbose=FALSE)

## get data.frames
snr.ini <- pvalData(call.ini)
inf.ini <- presCall(call.ini)
head(snr.ini)
head(inf.ini)

## plot results
if (interactive()) {
  callplot(call.ini)
}
```

```
rm(scheme.test3, data.test3)
gc()
```

---

initialize-methods *Initialize Classes*

---

### Description

Initialize S4 classes.

### Methods

Internal method to initialize S4 classes.

---

intensity-methods *Get/Set Data Values*

---

### Description

Get/set data values from/for class `DataTreeSet`.

#### Usage

```
intensity(object)
intensity(object, filename = NULL, verbose = FALSE) <- value
```

### Arguments

<code>object</code>	object of class <code>DataTreeSet</code> .
<code>filename</code>	character vector containing optional ROOT file name.
<code>verbose</code>	logical, if TRUE print status information.
<code>value</code>	<code>data.frame</code> containing expression values.

### Details

Get the intensity values from slot `data` or set slot `data` to `value`.

Method `intensity` returns the data values from slot `data` as `data.frame`, while replacement method `intensity<-` allows to replace slot `data` with a `data.frame`.

Using replacement method `intensity<-` with default settings will not change the data stored in the ROOT data file, and thus will not have any effect on subsequent processing methods. If you really want to use the replacement data for further processing you must supply a new ROOT filename. This will export each intensity column of `value` as CEL-file (version 3), which will then be imported into the new ROOT data file `filename`.

**Warning:** Do not use replacement method `intensity<-` until you really know what you are doing!

**Note:** The first two columns of replacement `data.frame` `value` must be the (X,Y) coordinates, followed by the intensities whereby the number of intensity columns must be identical to the columns to be replaced.

Note: If you do not want to replace your current object, create first a copy of type `DataTreeSet` by simply writing `newobj <- oldobj`, and use `newobj` for replacement. This is important since `intensity<-` does also update slots `rootfile`, `filedir` and `treenames` when a new filename was chosen.

Note: The CEL-files created are fully functional CEL-files (version 3), however some header rows such as `GridCornerUL`, `AlgorithmParameters`, and some of the data in `DatHeader` are placeholders only.

Warning: The CEL-files created WILL REPLACE THE ORIGINAL CEL-files, if they have identical names to the original CEL-files and the original CEL-files are located in the working directory. Thus the original CEL-files should preferably be located in directory `celdir` of function `import.data`.

### Author(s)

Christian Stratowa

### See Also

`validData`

### Examples

```
## Not run:
## load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## get intensity values
value <- intensity(data.test3)

## make a copy of your object if you do not want to replace it
newdata.test3 <- data.test3

## replace slot data with value
intensity(newdata.test3, "ReplacementData", FALSE) <- value
str(newdata.test3)

## now you can create an ExprTreeSet using the new intensity data
data.rma <- rma(newdata.test3, "ReplacementRMA", tmpdir="", background="none", normalize=TRUE)
## End(Not run)
```

---

isROOTFile

*Test for ROOT File*

---

### Description

Test if a file is a valid ROOT file.

### Usage

```
isROOTFile(filename)
```

**Arguments**

filename      name of ROOT file, including full path.

**Value**

Return TRUE if file filename is a valid ROOT file.

**Author(s)**

Christian Stratowa

**See Also**

[existsROOTFile](#)

**Examples**

```
isROOTFile(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/"))
```

lowFilter-methods    *Lower Threshold Filter*

**Description**

This method initializes the Lower Threshold Filter. The `cutoff` value defines the lower threshold for allowed expression levels. If e.g. the number of samples lower than this cutoff value is greater than `parameter` then the corresponding dataframe row is flagged, i.e. `flag = 0`.

The Lower Threshold Filter flags all rows with: `flag = (sum(expression[i] >= cutoff) >= parameter)`

*Usage*

```
lowFilter(object)
lowFilter(object, value)<-
```

**Arguments**

object      object of class PreFilter.  
value      character vector `c(cutoff, parameter, condition)`.

**Details**

The method `lowFilter` initializes the following parameters:

<code>cutoff</code> :	the lower threshold level for the filter.
<code>parameter</code> :	this value depends on the condition used:
<code>condition</code> :	<code>condition="samples"</code> : number of samples (default);
	<code>condition="percent"</code> : percent of samples.
	<code>condition="mean"</code> : mean value of samples.
	<code>condition="percentile"</code> : percentile of samples.

**Value**

An initialized `PreFilter` object.

**Author(s)**

Christian Stratowa

**Examples**

```
prefltr <- PreFilter()
lowFilter(prefltr) <- c(4.0, 3, "samples")
str(prefltr)
```

---

madFilter-methods *Median Absolute Deviation Filter*

---

**Description**

This method initializes the Median Absolute Deviation Filter.  
The MAD Filter flags all rows with: `flag = (mad >= cutoff)`

*Usage*

```
madFilter(object)
madFilter(object, value)<-
```

**Arguments**

<code>object</code>	object of class <code>PreFilter</code> .
<code>value</code>	numeric vector <code>c(cutoff, epsilon)</code> .

**Details**

The method `madFilter` initializes the following parameters:

<code>cutoff</code> :	the cutoff level for the filter.
<code>epsilon</code> :	value to replace mean (default is <code>epsilon=0.01</code> ).

Note, that `epsilon` has no effect on `mad`.

**Value**

An initialized `PreFilter` object.

**Author(s)**

Christian Stratowa

**Examples**

```
prefltr <- PreFilter()
madFilter(prefltr) <- c(0.5, 0.01)
str(prefltr)
```

---

mas4

*MAS 4.0 Expression Measure*


---

## Description

This function converts a `DataTreeSet` into an `ExprTreeSet` using the XPS implementation of Affymetrix's MAS 4.0 expression measure.

## Usage

```
mas4(xps.data,
      filename = character(0),
      filedir   = getwd(),
      tmpdir    = "",
      normalize = FALSE,
      sc        = 500,
      option    = "transcript",
      exonlevel = "",
      update    = FALSE,
      xps.scheme = NULL,
      add.data  = TRUE,
      verbose   = TRUE)
```

```
xpsMAS4(object, ...)
```

## Arguments

<code>xps.data</code>	object of class <code>DataTreeSet</code> .
<code>filename</code>	file name of ROOT data file.
<code>filedir</code>	system directory where ROOT data file should be stored.
<code>tmpdir</code>	optional temporary directory where temporary ROOT files should be stored.
<code>normalize</code>	logical. If <code>TRUE</code> scale normalization is used after an <code>ExprTreeSet</code> is obtained.
<code>sc</code>	value at which all arrays will be scaled to.
<code>option</code>	option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon arrays only.
<code>exonlevel</code>	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
<code>update</code>	logical. If <code>TRUE</code> the existing ROOT data file <code>filename</code> will be updated.
<code>xps.scheme</code>	optional alternative <code>SchemeTreeSet</code> .
<code>add.data</code>	logical. If <code>TRUE</code> expression data will be included as slot data.
<code>verbose</code>	logical, if <code>TRUE</code> print status information.
<code>object</code>	object of class <code>DataTreeSet</code> .
<code>...</code>	arguments <code>filename, filedir, tmpdir, option, exonlevel, xps.scheme</code> .



## Details

This function computes the Affymetrix MAS 4.0 expression measure, i.e. the ‘Average Difference’ expression level, as implemented in XPS.

If `normalize=TRUE` then the expression levels will be scaled to `sc`. For `sc=0` the expression levels will be scaled to the mean expression level.

`xpsMAS4` is the `DataTreeSet` method called by function `mas4`, however, expression levels will not be scaled to a common mean expression level.

For further details see [mas5](#).

## Value

An `ExprTreeSet`

## Note

In contrast to function `mas4`, expression levels computed with `xpsMAS4` will not be scaled to a common mean expression level.

## Author(s)

Christian Stratowa

## References

Affymetrix (1999) GeneChip Expression Analysis Algorithm Tutorial, Affymetrix Inc., Santa Clara, CA.

## See Also

[xpsMAS4](#), [express](#)

## Examples

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

data.mas4 <- mas4(data.test3, "tmp_Test3MAS4", tmpdir="", normalize=TRUE, sc=500, update=TRUE

## get data.frame
expr.mas4 <- validData(data.mas4)
head(expr.mas4)

## plot results (negative expression values!)
if (interactive()) {
  boxplot(expr.mas4)
}

rm(scheme.test3, data.test3)
gc()
```

mas5.call

*MAS 5.0 Absolute Detection Call***Description**

Performs the Wilcoxon signed rank-based gene expression presence/absence detection algorithm first implemented in the Affymetrix Microarray Suite version 5.

**Usage**

```
mas5.call(xps.data,
          filename = character(0), filedir = getwd(), tmpdir = "",
          tau = 0.015, alpha1 = 0.04, alpha2 = 0.06, ignore.saturated = TRUE,
          option = "transcript", exonlevel = "", xps.scheme = NULL, add.data = T

xpsMAS5Call(object, ...)
```

**Arguments**

xps.data	object of class DataTreeSet.
filename	file name of ROOT data file.
filedir	system directory where ROOT data file should be stored.
tmpdir	optional temporary directory where temporary ROOT files should be stored.
tau	a small positive constant.
alpha1	a significance threshold in (0,alpha2).
alpha2	a significance threshold in (alpha1,0.5).
ignore.saturated	logical. If TRUE do the saturation correction described in the paper, with a saturation level of 46000.
option	option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon arrays only.
exonlevel	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
xps.scheme	optional alternative SchemeTreeSet.
add.data	logical. If TRUE call data will be added to slots data and detcall.
verbose	logical, if TRUE print status information.
object	object of class DataTreeSet.
...	the arguments described above.

**Details**

This function performs the hypothesis test:

H0: median(Ri) = tau, corresponding to absence of transcript  
 H1: median(Ri) > tau, corresponding to presence of transcript

where  $R_i = (PM_i - MM_i) / (PM_i + MM_i)$  for each  $i$  a probe-pair in the probe-set represented by data.

The p-value that is returned estimates the usual quantity:

Pr(observing a more "present looking" probe-set than data | data is absent)

Small p-values imply presence while large ones imply absence of transcript. The detection call is computed by thresholding the p-value as in:

```
call "P" if p-value < alpha1
call "M" if alpha1 <= p-value < alpha2
call "A" if alpha2 <= p-value
```

The defaults for `tau`, `alpha1` and `alpha2` correspond to those in MAS5.0 for expression arrays. However, when using this function for exon or whole genome arrays, new values for `alpha1` and `alpha2` must be determined. The recommended function for exon/genome arrays is [dabg.call](#).

In order to use an alternative [SchemeTreeSet](#) set the corresponding `SchemeTreeSet` `xps.scheme`.

`xpsMAS5Call` is the `DataTreeSet` method called by function `mas5.call`, containing the same parameters.

### Value

A [CallTreeSet](#)

### Author(s)

Christian Stratowa

### References

Liu, W. M. and Mei, R. and Di, X. and Ryder, T. B. and Hubbell, E. and Dee, S. and Webster, T. A. and Harrington, C. A. and Ho, M. H. and Baid, J. and Smeekens, S. P. (2002) Analysis of high density expression microarrays with signed-rank call algorithms, *Bioinformatics*, 18(12), pp. 1593-1599.

Liu, W. and Mei, R. and Bartell, D. M. and Di, X. and Webster, T. A. and Ryder, T. (2001) Rank-based algorithms for analysis of microarrays, *Proceedings of SPIE, Microarrays: Optical Technologies and Informatics*, 4266.

Affymetrix (2002) Statistical Algorithms Description Document, Affymetrix Inc., Santa Clara, CA, whitepaper. [http://www.affymetrix.com/support/technical/whitepapers/sadd\\_whitepaper.pdf](http://www.affymetrix.com/support/technical/whitepapers/sadd_whitepaper.pdf)

### See Also

[dabg.call](#)

### Examples

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## MAS5 detection call
call.mas5 <- mas5.call(data.test3, "tmp_Test3Call", tmpdir="", verbose=FALSE)

## get data.frames
pval.mas5 <- pvalData(call.mas5)
pres.mas5 <- presCall(call.mas5)
head(pval.mas5)
head(pres.mas5)
```

```
## plot results
if (interactive()) {
  callplot(call.mas5, beside=FALSE, ylim=c(0,125))
}

rm(scheme.test3, data.test3)
gc()
```

---

mas5

*MAS 5.0 Expression Measure*


---

## Description

This function converts a [DataTreeSet](#) into an [ExprTreeSet](#) using the XPS implementation of Affymetrix's MAS 5.0 expression measure.

## Usage

```
mas5(xps.data,
     filename = character(0),
     filedir  = getwd(),
     tmpdir   = "",
     normalize = FALSE,
     sc       = 500,
     option   = "transcript",
     exonlevel = "",
     update   = FALSE,
     xps.scheme = NULL,
     add.data = TRUE,
     verbose  = TRUE)
```

```
xpsMAS5(object, ...)
```

## Arguments

<code>xps.data</code>	object of class <code>DataTreeSet</code> .
<code>filename</code>	file name of ROOT data file.
<code>filedir</code>	system directory where ROOT data file should be stored.
<code>tmpdir</code>	optional temporary directory where temporary ROOT files should be stored.
<code>normalize</code>	logical. If TRUE scale normalization is used after an <code>ExprTreeSet</code> is obtained.
<code>sc</code>	value at which all arrays will be scaled to.
<code>option</code>	option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon arrays only.
<code>exonlevel</code>	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
<code>update</code>	logical. If TRUE the existing ROOT data file <code>filename</code> will be updated.
<code>xps.scheme</code>	optional alternative <code>SchemeTreeSet</code> .

<code>add.data</code>	logical. If TRUE expression data will be included as slot data.
<code>verbose</code>	logical, if TRUE print status information.
<code>object</code>	object of class <code>DataTreeSet</code> .
<code>...</code>	arguments <code>filename</code> , <code>filedir</code> , <code>tmpdir</code> , <code>option</code> , <code>exonlevel</code> , <code>xps.scheme</code> .

## Details

This function computes the Affymetrix MAS 5.0 expression measure as implemented in XPS. Although this implementation is based on the Affymetrix ‘sadd\_whitepaper.pdf’, it can be used to compute an expression level for both expression arrays and exon arrays. For exon arrays it is necessary to supply the requested `option` and `exonlevel`.

Following `options` are valid for exon arrays:

<code>transcript:</code>	expression levels are computed for transcript clusters, i.e. probe sets containing the same ‘ <code>transcript_cluster</code> ’.
<code>exon:</code>	expression levels are computed for exon clusters, i.e. probe sets containing the same ‘ <code>exon_id</code> ’, where each ‘ <code>exon_id</code> ’ is a unique identifier for an exon.
<code>probeset:</code>	expression levels are computed for individual probe sets, i.e. for each ‘ <code>probeset_id</code> ’.

Following `exonlevel` annotations are valid for exon arrays:

<code>core:</code>	probesets supported by RefSeq and full-length GenBank transcripts.
<code>metacore:</code>	core meta-probesets.
<code>extended:</code>	probesets with other cDNA support.
<code>metaextended:</code>	extended meta-probesets.
<code>full:</code>	probesets supported by gene predictions only.
<code>metafull:</code>	full meta-probesets.
<code>ambiguous:</code>	ambiguous probesets only.
<code>affx:</code>	standard AFFX controls.
<code>all:</code>	combination of above (including <code>affx</code> ).

Following `exonlevel` annotations are valid for whole genome arrays:

<code>core:</code>	probesets with category ‘unique’, ‘similar’ and ‘mixed’.
<code>metacore:</code>	probesets with category ‘unique’ only.
<code>affx:</code>	standard AFFX controls.
<code>all:</code>	combination of above (including <code>affx</code> ).

Exon levels can also be combined, with following combinations being most useful:

<code>exonlevel="metacore+affx":</code>	core meta-probesets plus AFFX controls
<code>exonlevel="core+extended":</code>	probesets with cDNA support
<code>exonlevel="core+extended+full":</code>	supported plus predicted probesets

Exon level annotations are described in the Affymetrix whitepaper ‘`exon_probeset_trans_clust_whitepaper.pdf`’.

If `normalize=TRUE` then the expression levels will be scaled to `sc`. For `sc=0` the expression levels will be scaled to the mean expression level.

If `update=TRUE` then the existing `ROOT` file `filename` will be updated, however, this is usually only recommended as option for function `express`.

In order to use an alternative `SchemeTreeSet` set the corresponding `SchemeTreeSet` `xps.scheme`.

xpsMAS5 is the `DataTreeSet` method called by function `mas5`, however, expression levels will not be scaled to a common mean expression level.

### Value

An `ExprTreeSet`

### Note

In contrast to function `mas5`, expression levels computed with `xpsMAS5` will not be scaled to a common mean expression level.

### Author(s)

Christian Stratowa

### References

Affymetrix (2002) Statistical Algorithms Description Document, Affymetrix Inc., Santa Clara, CA, whitepaper. [http://www.affymetrix.com/support/technical/whitepapers/sadd\\_whitepaper.pdf](http://www.affymetrix.com/support/technical/whitepapers/sadd_whitepaper.pdf)

Affymetrix (2005) Exon Probeset Annotations and Transcript Cluster Groupings, Affymetrix Inc., Santa Clara, CA, `exon_probeset_trans_clust_whitepaper.pdf`.

### See Also

[express](#)

### Examples

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

data.mas5 <- mas5(data.test3, "tmp_Test3MAS5", tmpdir="", normalize=TRUE, sc=500, update=TRUE,

## get data.frame
expr.mas5 <- validData(data.mas5)
head(expr.mas5)

## plot results
if (interactive()) {
  boxplot(data.mas5)
  boxplot(log2(expr.mas5))
}

rm(scheme.test3, data.test3)
gc()
```

## Description

Produce boxplots of relative M values for the set of arrays.

### Usage

```
mboxplot(x, which = "", size = 0, transfo = log2, method = "mean",
range = 0, ylim = c(-1,1), outline = FALSE, names = "namepart", ...)
```

## Arguments

<code>x</code>	object of class <code>DataTreeSet</code> or <code>ExprTreeSet</code> .
<code>which</code>	type of probes to be used, for details see <code>validData</code> .
<code>size</code>	length of sequence to be generated as subset.
<code>transfo</code>	a valid function to transform the data, usually “log2”, or “0”.
<code>method</code>	method to create the reference data, “mean” or “median”.
<code>range</code>	determines how far the plot whiskers extend out from the box.
<code>ylim</code>	range for the plotted y values.
<code>outline</code>	if <code>outline</code> is not true, the outliers are not drawn.
<code>names</code>	optional vector of sample names.
<code>...</code>	optional arguments to be passed to <code>boxplot</code> .

## Details

Create boxplots of M plots, where M is determined relative to a pseudo-mean reference chip.

For `names=NULL` full column names of slot `data` will be displayed while for `names="namepart"` column names will be displayed without name extension. If `names` is a vector of column names, only these columns will displayed as boxplot.

## Note

For a `DataTreeSet` object, data must first be attached using method `attachInten`.

## Author(s)

Christian Stratowa

## See Also

`boxplot.dev`, `boxplot`

**Examples**

```
# load existing ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cell1.root", sep="/"))

# need to attach scheme mask and probe intensities
data.test3 <- attachMask(data.test3)
data.test3 <- attachInten(data.test3)

if (interactive()) {
mboxplot(data.test3, ylim=c(-6,6))
}

# optionally remove mask and data to free memory
data.test3 <- removeInten(data.test3)
data.test3 <- removeMask(data.test3)
```

---

metaProbesets

*Create MetaProbeset File for APT*


---

**Description**

Create MetaProbeset File for APT function “apt-probeset-summarize”.

**Usage**

```
metaProbesets(xps.scheme, infile = character(0), outfile = character(0), exonlevel = 1)
```

**Arguments**

xps.scheme	exon SchemeTreeSet.
infile	Name of file containing exon transcript_cluster_ids.
outfile	Name of resulting file containing meta probeset definitions.
exonlevel	exon annotation level determining which probes should be used.

**Details**

This function allows to create a metaprobeset file for APT function “apt-probeset-summarize” to be used with option “-m”. The *infile* must contain exon transcript\_cluster\_ids, one per line, e.g. one can export the `rownames(data.rma)`

The resulting file may be useful if you want to compare results created with *xps* to results created with APT function “apt-probeset-summarize”.

**Value**

None.

**Author(s)**

Christian Stratowa



**Examples**

```
## Not run:
## first, load ROOT exon scheme file:
scmdir <- "/Volumes/GigaDrive/CRAN/Workspaces/Schemes"
scheme.exon <- root.scheme(paste(scmdir, "Scheme_HuEx10stv2r2_na25.root", sep="/"))

metaProbesets(scheme.exon, "metacore.txt", "metacoreList.mps", "metacore")
## End(Not run)
```

mvaplot.dev

*MvA Scatter Plot for Device***Description**

Produce scatter plots of M values vs A values of the samples for the selected device.

**Usage**

```
mvaplot.dev(x, transfo = log2, method = "median", names = "namepart", ylim = c(-
```

**Arguments**

x	object of class <a href="#">ExprTreeSet</a> .
transfo	a valid function to transform the data, usually “log2”, or “0”.
method	method to compute M, “mean” or “median”.
names	optional vector of sample names.
ylim	range for the plotted M values.
xlab	a title for the x axis.
ylab	a title for the y axis.
pch	either an integer specifying a symbol or a single character to be used in plotting points.
mar	plot margin.
dev	graphics device to plot to, i.e. one of “screen”, “jpeg”, “png”, “pdf” or “ps”.
outfile	the name of the output file.
w	the width of the device in pixels.
h	the height of the device in pixels.
...	optional arguments to be passed to <code>plot</code> .

**Details**

Produces mvaplots for slot data for an object of class [ExprTreeSet](#) for the selected graphics device.

For `names=NULL` full column names of slot data will be displayed while for `names="namepart"` column names will be displayed without name extension. If `names` is a vector of column names, only these columns will displayed as mvaplot.

**Author(s)**

Christian Stratowa

**See Also**[mvaplot](#)

---

mvaplot-methods      *M vs A Plot*

---

**Description**

Produce scatter plots of M values vs A values of the samples.

*Usage*

```
mvaplot(x, transfo = log2, method = "median", names = "namepart",  
        ylim = c(-6,6), ...)
```

**Arguments**

x	object of class <a href="#">ExprTreeSet</a> .
transfo	a valid function to transform the data, usually “log2”, or “0”.
method	method to compute M, “mean” or “median”.
names	optional vector of sample names.
ylim	range for the plotted M values.
...	optional arguments to be passed to <code>plot</code> .

**Details**

Produces mvaplots for slot data for an object of class [ExprTreeSet](#).

For `names=NULL` full column names of slot data will be displayed while for `names="namepart"` column names will be displayed without name extension. If `names` is a vector of column names, only these columns will displayed as mvaplot.

**Author(s)**

Christian Stratowa

**See Also**[mvaplot.dev](#)

---

normalize	<i>Normalization on Affymetrix Probe Level Data or on Expression Levels</i>
-----------	---

---

### Description

Functions that allow to normalize Affymetrix arrays both at the probe level (“low-level normalization”) and/or at the expression level (“high-level normalization”).

### Usage

```
normalize(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE, select = "all", method = "lowess", option = "none", logbase = "0", exonlevel = 0, refindex = 0, refmethod = "median", params = character(0), add.data = FALSE, verbose = FALSE, object, ...)
```

### Arguments

xps.data	object of class <code>DataTreeSet</code> or <code>ExprTreeSet</code> .
filename	file name of ROOT data file.
filedir	system directory where ROOT data file should be stored.
tmpdir	optional temporary directory where temporary ROOT files should be stored.
update	logical. If TRUE the existing ROOT data file <code>filename</code> will be updated.
select	type of probes to select for normalization.
method	normalization method to use.
option	option determining the grouping of probes for normalization, and the selection of the probes.
logbase	logarithm base as character, one of ‘0’, ‘log’, ‘log2’, ‘log10’.
exonlevel	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
refindex	index of reference tree to use, or 0.
refmethod	for <code>refindex=0</code> , either trimmed mean or median of trees.
params	vector of parameters for normalization method.
add.data	logical. If TRUE expression data will be included as slot data.
verbose	logical, if TRUE print status information.
object	object of class <code>DataTreeSet</code> or <code>ExprTreeSet</code> .
...	the arguments described above.

**Details**

Functions that allow to normalize Affymetrix arrays both at the probe level (“low-level normalization”) and/or at the expression level (“high-level normalization”).

xpsNormalize are the `DataTreeSet` or `ExprTreeSet` methods, respectively, called by function `normalize`, containing the same parameters.

**Value**

An object of type `DataTreeSet` or `ExprTreeSet`.

**Warning**

Functions `normalize.lowess` and `normalize.supsmu` have only be tested for objects of type `ExprTreeSet` but not for objects of type `DataTreeSet`, i.e. for probe level intensities.

**Author(s)**

Christian Stratowa

**See Also**

[express](#)

**Examples**

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## RMA background
data.bg.rma <- bgcorrect.rma(data.test3, "tmp_Test3NormRMA", filedir=getwd(), tmpdir="", verb
## normalize quantiles
data.qu.rma <- normalize.quantiles(data.bg.rma, "tmp_Test3NormRMA", filedir=getwd(), tmpdir=
## summarize medianpolish
data.mp.rma <- summarize.rma(data.qu.rma, "tmp_Test3NormRMA", filedir=getwd(), tmpdir="", upo
```

---

pm-methods

*Methods for accessing perfect matches and mismatches*

---

**Description**

Methods for accessing perfect match (PM) and mismatch (MM) probes.

*Usage*

```
pm(object, which = "pm")
mm(object, which = "mm")
```

**Arguments**

`object`            object of class `DataTreeSet`.  
`which`            type of perfect match or mismatch probes to be returned.

**Details**

For expression arrays all the perfect match (pm) or mismatch (mm) probes on the arrays the object represents are returned as `data.frame`.

For exon arrays, `pm` returns the probes of the different exon levels as `data.frame`, i.e. which can have one of the following values:

<code>core:</code>	probesets supported by RefSeq and full-length GenBank transcripts.
<code>metacore:</code>	core meta-probesets.
<code>extended:</code>	probesets with other cDNA support.
<code>metaextended:</code>	extended meta-probesets.
<code>full:</code>	probesets supported by gene predictions only.
<code>metafull:</code>	full meta-probesets.
<code>affx:</code>	standard AFFX controls.

For whole genome arrays, `pm` returns the probes of the different exon levels as `data.frame`, i.e. which can have one of the following values:

<code>core:</code>	probesets with category 'unique' and 'mixed'.
<code>metacore:</code>	probesets with category 'unique' only.
<code>affx:</code>	standard AFFX controls.

For exon/genome arrays, `mm` returns the background probes as `data.frame`, i.e. which is either "genomic" or "antigenomic".

**Value**

A `data.frame`.

**Author(s)**

Christian Stratowa

**See Also**

[validData](#)

**Examples**

```
## load existing ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## need to attach scheme mask and probe intensities
data.test3 <- attachMask(data.test3)
data.test3 <- attachInten(data.test3)

pm <- pm(data.test3)
mm <- mm(data.test3)
head(pm)
head(mm)

## optionally remove mask and data to free memory
data.test3 <- removeInten(data.test3)
data.test3 <- removeMask(data.test3)
```

**Description**

Creates a barplot of mean perfect match and mismatch intensities.

*Usage*

```
pmplot(x, which = "", size = 0, transfo = NULL, method = "mean", names
= "namepart", beside = TRUE, col = c("red", "blue"), legend = c("PM", "MM"),
...)
```

**Arguments**

x	object of class <a href="#">DataTreeSet</a> .
which	type of probes to be used, for details see <a href="#">validData</a> .
size	length of sequence to be generated as subset.
transfo	a valid function to transform the data, usually “log2”, or “0”.
method	method to compute average intensities, “mean” or “median”.
names	optional vector of sample names.
beside	logical. If FALSE, mean intensities are portrayed as stacked bars, and if TRUE the columns are portrayed as juxtaposed bars.
col	color of PM, MM bars.
legend	a vector of text used to construct a legend for the plot, or a logical indicating whether a legend should be included.
...	optional arguments to be passed to <a href="#">barplot</a> .

**Details**

Produces barplots of mean perfect match and mismatch intensities for slot data for an object of class [ExprTreeSet](#).

For names=NULL full column names of slot data will be displayed while for names="namepart" column names will be displayed without name extension. If names is a vector of column names, only these columns will displayed as pmplot.

**Note**

Data must first be attached to class [DataTreeSet](#) using method [attachInten](#).

**Author(s)**

Christian Stratowa

**See Also**

[boxplot.dev](#), [boxplot](#), [barplot](#)

**Examples**

```
## load existing ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## need to attach scheme mask and probe intensities
data.test3 <- attachMask(data.test3)
data.test3 <- attachInten(data.test3)

if (interactive()) {
  pmpplot(data.test3)
}

## optionally remove mask and data to free memory
data.test3 <- removeInten(data.test3)
data.test3 <- removeMask(data.test3)
```

---

PreFilter-class	<i>Class PreFilter</i>
-----------------	------------------------

---

**Description**

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

**Objects from the Class**

Objects can be created by calls of the form `new("PreFilter", ...)`. Alternatively, the constructor `PreFilter` can be used.

**Slots**

**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 `ratioFilter`.  
**gap:** Object of class "list" describing parameters for `gapFilter`.  
**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`.  
**numfilters:** Object of class "numeric" giving the number of filters applied.

**Extends**

Class "`Filter`", directly.

**Methods**

**callFilter** signature(object = "PreFilter"): extracts slot prescall.

**callFilter<-** signature(object = "PreFilter", value = "character"): replaces slot prescall with character vector c(cutoff, samples, condition).

**cvFilter** signature(object = "PreFilter"): extracts slot cv.

**cvFilter<-** signature(object = "PreFilter", value = "numeric"): replaces slot cv with numeric vector c(cutoff, trim, epsilon).

**diffFilter** signature(object = "PreFilter"): extracts slot difference.

**diffFilter<-** signature(object = "PreFilter", value = "numeric"): replaces slot difference with numeric vector c(cutoff, trim, epsilon).

**gapFilter** signature(object = "PreFilter"): extracts slot gap.

**gapFilter<-** signature(object = "PreFilter", value = "numeric"): replaces slot gap with numeric vector c(cutoff, window, trim, epsilon).

**highFilter** signature(object = "PreFilter"): extracts slot hithreshold.

**highFilter<-** signature(object = "PreFilter", value = "character"): replaces slot hithreshold with character vector c(cutoff, parameter, condition).

**lowFilter** signature(object = "PreFilter"): extracts slot lothreshold.

**lowFilter<-** signature(object = "PreFilter", value = "character"): replaces slot lothreshold with character vector c(cutoff, parameter, condition).

**madFilter** signature(object = "PreFilter"): extracts slot mad.

**madFilter<-** signature(object = "PreFilter", value = "numeric"): replaces slot mad with numeric vector c(cutoff, epsilon).

**quantileFilter** signature(object = "PreFilter"): extracts slot quantile.

**quantileFilter<-** signature(object = "PreFilter", value = "numeric"): replaces slot quantile with numeric vector c(cutoff, loquantile, hiquantile).

**ratioFilter** signature(object = "PreFilter"): extracts slot ratio.

**ratioFilter<-** signature(object = "PreFilter", value = "numeric"): replaces slot ratio with numeric vector c(cutoff).

**varFilter** signature(object = "PreFilter"): extracts slot variance.

**varFilter<-** signature(object = "PreFilter", value = "numeric"): replaces slot variance with numeric vector c(cutoff, trim, epsilon).

**Author(s)**

Christian Stratowa

**See Also**

related classes [Filter](#), [UniFilter](#).

**Examples**

```
## for demonstration purposes only: initialize all pre-filters
prefltr <- new("PreFilter")
madFilter(prefltr) <- c(0.5,0.01)
cvFilter(prefltr) <- c(0.3,0.0,0.01)
varFilter(prefltr) <- c(0.6,0.02,0.01)
```



```

diffFilter(prefltr) <- c(2.2,0.0,0.01)
ratioFilter(prefltr) <- c(1.5)
gapFilter(prefltr) <- c(0.3,0.05,0.0,0.01)
lowFilter(prefltr) <- c(4.0,3,"samples")
highFilter(prefltr) <- c(14.5,75.0,"percent")
quantileFilter(prefltr) <- c(3.0, 0.05, 0.95)
callFilter(prefltr) <- c(0.02,80.0,"percent")
str(prefltr)

```

---

PreFilter-constructor

*Constructor for Class PreFilter*


---

## Description

Constructor for class PreFilter allows to apply different filters to class [ExprTreeSet](#), i.e. to the expression level data.frame data.

## Usage

```

PreFilter(mad          = character(),
          cv           = character(),
          variance     = character(),
          difference   = character(),
          ratio        = character(),
          gap          = character(),
          lothreshold  = character(),
          hithreshold  = character(),
          quantile     = character(),
          prescall     = character())

```

## Arguments

mad	"character" vector describing parameters for <a href="#">madFilter</a> .
cv	"character" vector describing parameters for <a href="#">cvFilter</a> .
variance	"character" vector describing parameters for <a href="#">varFilter</a> .
difference	"character" vector describing parameters for <a href="#">diffFilter</a> .
ratio	"character" vector describing parameters for <a href="#">ratioFilter</a> .
gap	"character" vector describing parameters for <a href="#">gapFilter</a> .
lothreshold	"character" vector describing parameters for <a href="#">lowFilter</a> .
hithreshold	"character" vector describing parameters for <a href="#">highFilter</a> .
quantile	"character" vector describing parameters for <a href="#">quantileFilter</a> .
prescall	"character" vector describing parameters for <a href="#">callFilter</a> .

**Details**

The PreFilter constructor allows to apply the following filters to class `ExprTreeSet`:

mad:	character vector c(cutoff,epsilon).
cv:	character vector c(cutoff,trim,epsilon).
variance:	character vector c(cutoff,trim,epsilon).
difference:	character vector c(cutoff,trim,epsilon).
ratio:	character vector c(cutoff).
gap:	character vector c(cutoff>window,trim,epsilon).
lothreshold:	character vector c(cutoff,parameter,condition).
hithreshold:	character vector c(cutoff,parameter,condition).
quantile:	character vector c(cutoff,loquantile,hiquantile).
prescall:	character vector c(cutoff,samples,condition).

**Value**

An object of type "`PreFilter`"

**Note**

Function `PreFilter` is used as constructor for class `PreFilter` so that the user need not know details for creating S4 classes.

**Author(s)**

Christian Stratowa

**See Also**

`Filter`, `UniFilter`

**Examples**

```
## fill character vectors within constructor
prefltr <- PreFilter(mad=c(0.5,0.01), prescall=c(0.002, 6,"samples"),
                   lothreshold=c(6.0,0.02,"mean"), hithreshold=c(10.5,80.0,"percent"))
str(prefltr)

## alternatively add character vectors as methods after creation of constructor
prefltr <- PreFilter()
madFilter(prefltr) <- c(0.5,0.01)
gapFilter(prefltr) <- c(0.3,0.05,0.0,0.01)
lowFilter(prefltr) <- c(4.0,3,"samples")
highFilter(prefltr) <- c(14.5,75.0,"percent")
str(prefltr)
```

---

 prefilter

*Function for Applying a PreFilter to an ExprTreeSet*


---

### Description

This function applies a [PreFilter](#) to an [ExprTreeSet](#).

### Usage

```

prefilter(xps.expr,
          filename = character(0),
          filedir  = getwd(),
          filter   = NULL,
          minfilters = 999,
          logbase  = "log2",
          treename = "PreFilter",
          xps.call = NULL,
          verbose  = TRUE)

xpsPreFilter(object, ...)
```

### Arguments

<code>xps.expr</code>	object of class <code>ExprTreeSet</code> .
<code>filename</code>	file name of ROOT filter file.
<code>filedir</code>	system directory where ROOT filter file should be stored.
<code>filter</code>	object of class <code>PreFilter</code> .
<code>minfilters</code>	minimum number of initialized filter methods to satisfy (default is all filters).
<code>logbase</code>	convert data to logarithm of base: "0", "log", "log2" (default), "log10"
<code>treename</code>	tree name to be used in ROOT filter file.
<code>xps.call</code>	optional object of class <code>CallTreeSet</code> .
<code>verbose</code>	logical, if TRUE print status information.
<code>object</code>	object of class <code>ExprTreeSet</code> .
<code>...</code>	same arguments as function <code>prefilter</code> .

### Details

This function applies the different filters initialized with constructor [PreFilter](#) to the [ExprTreeSet](#) `xps.expr`.

Slot `minfilters` determines the minimum number of initialized filters, which must be satisfied so that the mask is set to `flag=1`. For `minfilters=1` at least one filter must be satisfied, equivalent to logical 'OR'; for `minfilters=999` all filters must be satisfied, equivalent to logical 'AND'.

If method `callFilter` was initialized with constructor [PreFilter](#) then [CallTreeSet](#) `xps.call` must be supplied, usually created with function `mas5.call`.

### Value

A [FilterTreeSet](#)

**Author(s)**

Christian Stratowa

**See Also**[PreFilter](#), [unifilter](#)**Examples**

```
## Not run:
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## second, create an ExprTreeSet
data.rma <- rma(data.test3, "tmp_TestRMA", tmpdir="", background="pmonly", normalize=TRUE, ver
## note: do not copy/paste this code, it is necessary only because R CMD check fails sinc
data.rma@rootfile <- paste(.path.package("xps"), "rootdata/tmp_Test3RMA.root", sep="/")
data.rma@filedir <- paste(.path.package("xps"), "rootdata", sep="/")

## third, construct a PreFilter
prefltr <- PreFilter(mad=c(0.5, 0.01), lothreshold=c(6.0, 0.02, "mean"), hithreshold=c(10.5, 80

## finally, create a FilterTreeSet
rma.pfr <- prefilter(data.rma, "tmp_Test3Prefilter", getwd(), prefltr, 2, verbose=FALSE)
str(rma.pfr)
## End(Not run)
```

---

presCall-methods    *Get/Set Present Call Values*

---

**Description**

Get/set present call values from/for class `CallTreeSet`.

*Usage*

```
presCall(object)
presCall(object, treenames = NULL) <- value
```

```
pvalData(object)
pvalData(object, treenames = NULL) <- value
```

**Arguments**

<code>object</code>	object of class <code>CallTreeSet</code> .
<code>treenames</code>	character vector containing optional tree names to be used as subset.
<code>value</code>	<code>data.frame</code> containing present call values.

**Details**

Get the p-values from slot `data` or present calls from slot `detcall`, or set slot `data` or `detcall`, respectively, to `value`.

Method `presCall` returns the present calls from slot `detcall` as `data.frame`, while replacement method `presCall<-` allows to replace slot `detcall` with a `data.frame`.

Method `pvalData` returns the p-values from slot `data` as `data.frame`, while replacement method `pvalData<-` allows to replace slot `data` with a `data.frame`.

In order to create an `CallTreeSet` containing only a subset of e.g. slot `data`, first export slot `data` using method `pvalData`, create a character vector containing only `treenames` to be used in the subset, and then use replacement method `pvalData<-` to replace slot `data` with the subset. Slots `treenames` and `numtrees` will be updated automatically for `pvalData<-` but not for `presCall<-`.

Note: When creating character vector `treenames` it is sufficient to use the name part of the tree name w/o the extension.

Note: If you do not want to replace your current object, create first a copy of type `CallTreeSet` by simply writing `newobj <- oldobj`, and use `newobj` for replacement.

**Author(s)**

Christian Stratowa

**See Also**

[exprs](#)

**Examples**

```
## Not run:
## load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## create an CallTreeSet
call.mas5 <- mas5.call(data.test3, "tmp_TestMAS5Call", tmpdir="", verbose=FALSE)

## get p-values
value <- pvalData(call.mas5)

## selected treenames only
treenames <- c("TestA2", "TestB1")

## make a copy of your object if you do not want to replace it
subset.call <- call.mas5

## replace slot data with subset
exprs(subset.call, treenames) <- value
str(subset.call)
## End(Not run)
```

---

ProcesSet-class      *Class ProcesSet*

---

### Description

This class provides access to class [SchemeTreeSet](#) for the derived classes [DataTreeSet](#), [ExprTreeSet](#) and [CallTreeSet](#). It extends class [TreeSet](#).

### Objects from the Class

Usually, no objects are created from it.

### Slots

- scheme:** Object of class "SchemeTreeSet" providing access to [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.
- 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 'rawdata', 'preprocess'.
- rootfile:** Object of class "character" representing the name of the [ROOT](#) file, including 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 subdirectory setname.
- treenames:** Object of class "list" representing the names of the [ROOT](#) trees stored in subdirectory setname.

### Extends

Class "[TreeSet](#)", directly.

### Methods

- boxplot** signature (x = "ProcesSet"): creates a [boxplot](#) of the data from data.frame data.
- chipName** signature (object = "ProcesSet"): extracts slot chipname from slot scheme.
- chipType** signature (object = "ProcesSet"): extracts slot chiptype from slot scheme.
- export** signature (object = "ProcesSet"): exports [ROOT](#) trees as text file, see [export-methods](#).
- getTreeData** signature (object = "ProcesSet"): exports tree data from [ROOT](#) file rootfile, and saves as data.frame data.
- hist** signature (x = "ProcesSet"): creates a plot showing the histograms for data.frame data.

**mboxplot** signature(x = "ProcesSet"): creates an M-boxplot of the data from data.frame data.

**schemeFile** signature(object = "ProcesSet"): extracts the [ROOT](#) scheme file from slot scheme.

**schemeFile<-** signature(object = "ProcesSet"), value = "character"): replaces the [ROOT](#) scheme file from slot scheme.

**schemeSet** signature(object = "ProcesSet"): extracts slot scheme.

**schemeSet<-** signature(object = "ProcesSet"), value = "SchemeTreeSet"): replaces slot scheme with a different SchemeTreeSet.

**validData** signature(object = "ProcesSet"): extracts a subset of columns from data.frame data.

### Author(s)

Christian Stratowa

### See Also

derived classes [DataTreeSet](#), [ExprTreeSet](#), [CallTreeSet](#).

### Examples

```
showClass("ProcesSet")
```

---

ProjectInfo-class *Class ProjectInfo*

---

### Description

This class allows to save the relevant project information in the [ROOT](#) data file and in class [DataTreeSet](#).

### Objects from the Class

Objects can be created by calls of the form

```
new("ProjectInfo", submitter=[character], laboratory=[character], contact=[character] ...).
```

Alternatively, the constructor [ProjectInfo](#) can be used.

### Slots

**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.

## Methods

**projectInfo** signature(object = "ProjectInfo"): extracts slot project.

**projectInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot project with character vector c(name,date,type,description,comments).

**authorInfo** signature(object = "ProjectInfo"): extracts slot author.

**authorInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot author with character vector c(lastname,firstname,type,company,department,email,phone,comments).

**datasetInfo** signature(object = "ProjectInfo"): extracts slot dataset.

**datasetInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot dataset with character vector c(name,type,sample,submitter,date,description,comments).

**sourceInfo** signature(object = "ProjectInfo"): extracts slot source.

**sourceInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot source with character vector c(name,type,species,subspecies,description,comments).

**sampleInfo** signature(object = "ProjectInfo"): extracts slot sample.

**sampleInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot sample with character vector c(name,type,sex,phenotype,genotype,extraction,ixxenograft,xenostain,xenosex,xenoage,xenoageunit,comments).

**cellineInfo** signature(object = "ProjectInfo"): extracts slot celline.

**cellineInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot celline with character vector c(name,type,parent,atcc,modification,sex,phenotype,genotype,extraction,ixxenograft,xenostain,xenosex,xenoage,xenoageunit,comments).

**primcellInfo** signature(object = "ProjectInfo"): extracts slot primarycell.

**primcellInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot primarycell with character vector c(name,type,date,description,sex,phenotype,genotype,extraction,ixxenograft,xenostain,xenosex,xenoage,xenoageunit,comments).

**tissueInfo** signature(object = "ProjectInfo"): extracts slot tissue.

**tissueInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot tissue with character vector c(name,type,development,morphology,disease,stage,donorage,ageunit,status,sex,phenotype,genotype,extraction,ixxenograft,xenostain,xenosex,xenoage,xenoageunit,comments).

**biopsyInfo** signature(object = "ProjectInfo"): extracts slot biopsy.

**biopsyInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot biopsy with character vector c(name,type,morphology,disease,stage,donorage,ageunit,status,sex,phenotype,genotype,extraction,ixxenograft,xenostain,xenosex,xenoage,xenoageunit,comments).



**arrayInfo** signature(object = "ProjectInfo"): extracts slot arraytype.

**arrayInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot arraytype with character vector c(chipname,chiptype,description,comments).

**hybridizInfo** signature(object = "ProjectInfo"): extracts slot hybridizations.

**hybridizInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot hybridizations with vector of character vectors with each containing c(name,type,inputname,date,p

**treatmentInfo** signature(object = "ProjectInfo"): extracts slot treatments.

**treatmentInfo<-** signature(object = "ProjectInfo", value = "character"): replaces slot treatments with vector of character vectors with each containing c(name,type.concentration,concer

**show** signature(object = "ProjectInfo"): shows the content of ProjectInfo.

### Author(s)

Christian Stratowa

### Examples

```
project <- new("ProjectInfo",submitter="Christian", laboratory="home",contact="email")
projectInfo(project) <- c("TestProject","20060106","Project Type","use Test3 data for
authorInfo(project) <- c("Stratowa","Christian","Project Leader","Company","Dept","cs
datasetInfo(project) <- c("Test3Set","MC","Tissue","Stratowa","20060106","description"
sourceInfo(project) <- c("Unknown","source type","Homo sapiens","caucasian","descript
primcellInfo(project) <- c("Mel31","primary cell",20071123,"extracted from patient","ma
arrayInfo(project) <- c("Test3","GeneChip","description","my comment")
hybridizInfo(project) <- c(c("TestA1","hyb type","TestA1.CEL",20071117,"my prep1","star
c("TestA2","hyb type","TestA2.CEL",20071117,"my prep2","star
c("TestB1","hyb type","TestB1.CEL",20071117,"my prep1","star
c("TestB2","hyb type","TestB2.CEL",20071117,"my prep2","star
treatmentInfo(project) <- c(c("TestA1","DMSO",4.3,"mM",1.0,"hours","intravenous","my com
c("TestA2","DMSO",4.3,"mM",8.0,"hours","intravenous","my com
c("TestB1","DrugA2",4.3,"mM",1.0,"hours","intravenous","my c
c("TestB2","DrugA2",4.3,"mM",8.0,"hours","intravenous","my c
show(project)
```

---

ProjectInfo-constructor

*Constructor for Class ProjectInfo*

---

### Description

Constructor for class ProjectInfo class allows to save the relevant project information in the [ROOT](#) data file and in class [DataTreeSet](#).

### Usage

```
ProjectInfo(submitter = character(),
            laboratory = character(),
            contact = character(),
            project = character(),
            author = character(),
            dataset = character(),
```

```

source          = character() ,
sample          = character() ,
celline         = character() ,
primarycell     = character() ,
tissue          = character() ,
biopsy          = character() ,
arraytype       = character() ,
hybridizations = character() ,
treatments      = character() )

```

### Arguments

submitter	"character" representing the name of the submitter.
laboratory	"character" representing the laboratory of the submitter.
contact	"character" representing the contact address of the submitter.
project	"character" vector representing the project information.
author	"character" vector representing the author information.
dataset	"character" vector representing the dataset information.
source	"character" vector representing the sample source information.
sample	"character" vector representing the sample information.
celline	"character" vector representing the sample information for cell lines.
primarycell	"character" vector representing the sample information for primary cells.
tissue	"character" vector representing the sample information for tissues.
biopsy	"character" vector representing the sample information for biopsies.
arraytype	"character" vector representing the array information.
hybridizations	"character" vector representing the hybridization information for each hybridization.
treatments	"character" vector representing the treatment information for each hybridization.

### Details

The ProjectInfo constructor allows to save the following project information in the [ROOT](#) data file and in class [DataTreeSet](#):

submitter:	name of the submitter.
laboratory:	laboratory of the submitter.
contact:	contact address of the submitter.
project:	character vector c(name,date,type,description,comments).
author:	character vector c(lastname,firstname,type,company,department,email, phone,comments)..
dataset:	character vector c(name,type,sample,submitter,date,description,comments).
source:	character vector c(name,type,species,subspecies,description,comments).
sample:	character vector c(name,type,sex,phenotype,genotype,extraction, isxenograft,xenostain,xenose).
celline:	character vector c(name,type,parent,atcc,modification,sex,phenotype, genotype,extraction,isxenograft,x).
primarycell:	character vector c(name,type,date,description,sex,phenotype, genotype,extraction,isxenograft,x).
tissue:	character vector c(name,type,development,morphology,disease,stage, donorage,ageunit,status,s).
biopsy:	character vector c(name,type,morphology,disease,stage,donorage,ageunit, status,sex,phenotype).
arraytype:	character vector c(chipname,chiptype,description,comments).
hybridizations:	vector of character vectors with each containing c(name,type,inputname,date,preparation,protocol).
treatments:	vector of character vectors with each containing c(name,type,concentration,concentrationunit,ti).

**Value**

An object of type "[ProjectInfo](#)"

**Note**

Function `ProjectInfo` is used as constructor for class `ProjectInfo` so that the user need not know details for creating S4 classes.

**Author(s)**

Christian Stratowa

**See Also**

[ProjectInfo](#)

**Examples**

```
## fill character vectors within constructor
project <- ProjectInfo(submitter="Christian", laboratory="home",contact="email",
                      project=c("TestProject","20060106","Project Type","use Test3 data
                                hybridizations=c(c("TestA1","hyb type","TestA1.CEL",20071117,"my p
                                                c("TestA2","hyb type","TestA2.CEL",20071117,"my p
                                                c("TestB1","hyb type","TestB1.CEL",20071117,"my p
                                                c("TestB2","hyb type","TestB2.CEL",20071117,"my p

str(project)

## alternatively add character vectors as methods after creation of constructor
authorInfo(project) <- c("Stratowa","Christian","Project Leader","Company","Dept","cst
datasetInfo(project) <- c("Test3Set","MC","Tissue","Stratowa","20060106","description",
treatmentInfo(project) <- c(c("TestA1","DMSO",4.3,"mM",1.0,"hours","intravenous","my comm
                             c("TestA2","DMSO",4.3,"mM",8.0,"hours","intravenous","my comm
                             c("TestB1","DrugA2",4.3,"mM",1.0,"hours","intravenous","my co
                             c("TestB2","DrugA2",4.3,"mM",8.0,"hours","intravenous","my co

str(project)
```

---

quantileFilter-methods

*Quantile Filter*

---

**Description**

This method initializes the Quantile Filter.

The Quantile Filter flags all rows with: `flag = (quantile[high]/quantile[low] >= cutoff)`

*Usage*

```
quantileFilter(object)
quantileFilter(object, value)<-
```

**Arguments**

`object`            **object of class** `PreFilter`.  
`value`             **numeric vector** `c(cutoff, loquantile, hiquantile)`.

**Details**

The method `quantileFilter` initializes the following parameters:

```
cutoff:      the cutoff level for the filter.
loquantile:  value for low quantile (default is loquantile=0.05).
hiquantile:  value for high quantile (default is hiquantile=0.95).
```

**Value**

An initialized `PreFilter` object.

**Author(s)**

Christian Stratowa

**Examples**

```
prefltr <- PreFilter()
quantileFilter(prefltr) <- c(3.0, 0.05, 0.95)
str(prefltr)
```

---

ratioFilter-methods

*Ratio Filter*

---

**Description**

This method initializes the Ratio Filter. The ratio is the maximum value divided by minimum value for each row of the expression dataframe.

The Ratio Filter flags all rows with: `flag = (max/min >= cutoff)`

*Usage*

```
ratioFilter(object)
ratioFilter(object, value)<-
```

**Arguments**

```
object      object of class PreFilter.
value       numeric value c(cutoff).
```

**Details**

The method `ratioFilter` initializes the following parameters:

```
cutoff:    the cutoff level for the filter.
```

**Value**

An initialized `PreFilter` object.

**Author(s)**

Christian Stratowa

**Examples**

```
prefltr <- PreFilter()
ratioFilter(prefltr) <- c(1.5)
str(prefltr)
```

---

rawCELName-methods *Method for getting names of the raw CEL-files*

---

**Description**

Method for getting names (and full path) of the original CEL-files.

*Usage*

```
rawCELName(object, treename = "*", fullpath = TRUE)
```

**Arguments**

object	object of class DataTreeSet.
treename	treename, for which the name of the original CEL-file should be returned.
fullpath	logical, if TRUE return full path.

**Details**

Since CEL-files can be imported with `import.data` using alternative `celnames`, method `rawCELName` allows to return the original name and optionally the full path for each CEL-file.

**Value**

A character vector.

**Author(s)**

Christian Stratowa

**See Also**

[import.data](#)

**Examples**

```
## load existing ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

rawCELName(data.test3)
rawCELName(data.test3, treename = "TestA2.cel", fullpath = FALSE)
```

rma

*Robust Multi-Array Average Expression Measure***Description**

This function converts a `DataTreeSet` into an `ExprTreeSet` using the robust multi-array average (RMA) expression measure.

**Usage**

```
rma(xps.data,
    filename = character(0),
    filedir  = getwd(),
    tmpdir   = "",
    background = "pmonly",
    normalize = TRUE,
    option    = "transcript",
    exonlevel = "",
    xps.scheme = NULL,
    add.data  = TRUE,
    verbose   = TRUE)
```

```
xpsRMA(object, ...)
```

**Arguments**

<code>xps.data</code>	object of class <code>DataTreeSet</code> .
<code>filename</code>	file name of ROOT data file.
<code>filedir</code>	system directory where ROOT data file should be stored.
<code>tmpdir</code>	optional temporary directory where temporary ROOT files should be stored.
<code>background</code>	probes used to compute background, one of 'pmonly', 'mmonly', 'both'; for genome/exon arrays one of 'genomic', 'antigenomic'
<code>normalize</code>	logical. If TRUE normalize data using quantile normalization.
<code>option</code>	option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon arrays only.
<code>exonlevel</code>	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
<code>xps.scheme</code>	optional alternative <code>SchemeTreeSet</code> .
<code>add.data</code>	logical. If TRUE expression data will be included as slot data.
<code>verbose</code>	logical, if TRUE print status information.
<code>object</code>	object of class <code>DataTreeSet</code> .
<code>...</code>	the arguments described above.

**Details**

This function computes the RMA (Robust Multichip Average) expression measure described in Irizarry et al. for both expression arrays and exon arrays. For exon arrays it is necessary to supply the requested `option` and `exonlevel`.

Following options are valid for exon arrays:

transcript: expression levels are computed for transcript clusters, i.e. probe sets containing the same 'transcript\_clu  
 exon: expression levels are computed for exon clusters, i.e. probe sets containing the same 'exon\_id', where ea  
 probeset: expression levels are computed for individual probe sets, i.e. for each 'probeset\_id'.

Following `exonlevel` annotations are valid for exon arrays:

<code>core:</code>	probesets supported by RefSeq and full-length GenBank transcripts.
<code>metacore:</code>	core meta-probesets.
<code>extended:</code>	probesets with other cDNA support.
<code>metaextended:</code>	extended meta-probesets.
<code>full:</code>	probesets supported by gene predictions only.
<code>metafull:</code>	full meta-probesets.
<code>ambiguous:</code>	ambiguous probesets only.
<code>affx:</code>	standard AFFX controls.
<code>all:</code>	combination of above (including affx).

Following `exonlevel` annotations are valid for whole genome arrays:

<code>core:</code>	probesets with category 'unique', 'similar' and 'mixed'.
<code>metacore:</code>	probesets with category 'unique' only.
<code>affx:</code>	standard AFFX controls.
<code>all:</code>	combination of above (including affx).

Exon levels can also be combined, with following combinations being most useful:

<code>exonlevel="metacore+affx":</code>	core meta-probesets plus AFFX controls
<code>exonlevel="core+extended":</code>	probesets with cDNA support
<code>exonlevel="core+extended+full":</code>	supported plus predicted probesets

Exon level annotations are described in the Affymetrix whitepaper `exon_probeset_trans_clust_whitepaper.pdf`:  
 "Exon Probeset Annotations and Transcript Cluster Groupings".

In order to use an alternative `SchemeTreeSet` set the corresponding `SchemeSet` `xps.scheme`.  
`xpsRMA` is the `DataSet` method called by function `rma`, containing the same parameters.

## Value

An `ExprTreeSet`

## Note

In contrary to other implementations of RMA the expression measure is given to you in linear scale, analogously to the expression measures computed with `mas5` and `mas4`.

It is also possible to skip background correction by setting parameter `background="none"`.

For the analysis of many exon arrays it may be better to define a `tmpdir`, since this will store only the results in the main file and not e.g. background and normalized intensities, and thus will reduce the file size of the main file. For quantile normalization memory should not be an issue, however medianpolish depends on RAM unless you are using a temporary file.

Parameter `exonlevel` determines not only which probes are used for medianpolish, but also the probes used for background calculation and for quantile normalization. If you want to use

separate probes for background calculation, quantile normalization and medianpolish summarization, you can pass a numeric vector containing three integer values corresponding to the respective exonlevel, e.g. you can use `exonlevel=c(16316, 8252, 8252)`, see function `exonLevel` for more details.

### Author(s)

Christian Stratowa

### References

Rafael. A. Irizarry, Benjamin M. Bolstad, Francois Collin, Leslie M. Cope, Bridget Hobbs and Terence P. Speed (2003), Summaries of Affymetrix GeneChip probe level data *Nucleic Acids Research* 31(4):e15

Bolstad, B.M., Irizarry R. A., Astrand M., and Speed, T.P. (2003), A Comparison of Normalization Methods for High Density Oligonucleotide Array Data Based on Bias and Variance. *Bioinformatics* 19(2):185-193

Irizarry, RA, Hobbs, B, Collin, F, Beazer-Barclay, YD, Antonellis, KJ, Scherf, U, Speed, TP (2003) Exploration, Normalization, and Summaries of High Density Oligonucleotide Array Probe Level Data. *Biostatistics* .Vol. 4, Number 2: 249-264

### See Also

[express](#)

### Examples

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

data.rma <- rma(data.test3, "tmp_Test3RMA", tmpdir="", background="ponly", normalize=TRUE, ve

## get data.frame
expr.rma <- validData(data.rma)
head(expr.rma)

## plot results
if (interactive()) {
  boxplot(data.rma)
  boxplot(log2(expr.rma))
}

rm(scheme.test3, data.test3)
gc()

## Not run:
## examples using Affymetrix human tissue dataset (see also xps/examples/script4exon.R)
## first, load ROOT scheme file and ROOT data file from e.g.:
scmdir <- "/Volumes/GigaDrive/CRAN/Workspaces/Schemes"
datdir <- "/Volumes/GigaDrive/CRAN/Workspaces/ROOTData"

## 1. example - expression array, e.g. HG-U133_Plus_2:
scheme.u133p2 <- root.scheme(paste(scmdir, "Scheme_HGU133p2_na25.root", sep="/"))
data.u133p2 <- root.data(scheme.u133p2, paste(datdir, "HuTissuesU133P2_cel.root", sep="/"))
```



```

workdir <- "/Volumes/GigaDrive/CRAN/Workspaces/Exon/hutissues/u133p2"
data.rma <- rma(data.u133p2, "MixU133P2RMA", filedir=workdir, tmpdir="",
               background="pmonly", normalize=TRUE)

## 2. example - whole genome array, e.g. HuGene-1_0-st-v1:
scheme.genome <- root.scheme(paste(scmdir, "Scheme_HuGene10stv1r3_na25.root", sep="/"))
data.genome <- root.data(scheme.genome, paste(datdir, "HuTissuesGenome_cel.root", sep="/"))

workdir <- "/Volumes/GigaDrive/CRAN/Workspaces/Exon/hutissues/hugene"
data.g.rma <- rma(data.genome, "HuGeneMixRMAMetacore", filedir=workdir, tmpdir="",
                 background="antigenomic", normalize=T, exonlevel="metacore+affx")

## 3. example - exon array, e.g. HuEx-1_0-st-v2:
scheme.exon <- root.scheme(paste(scmdir, "Scheme_HuEx10stv2r2_na25.root", sep="/"))
data.exon <- root.data(scheme.exon, paste(datdir, "HuTissuesExon_cel.root", sep="/"))

workdir <- "/Volumes/GigaDrive/CRAN/Workspaces/Exon/hutissues/exon"
data.x.rma <- rma(data.exon, "MixRMAMetacore", filedir=workdir, tmpdir="", background="antige
                normalize=T, option="transcript", exonlevel="metacore")
## End (Not run)

```

---

```
root.browser-methods
```

*Open the ROOT object browser*

---

## Description

Open the [ROOT](#) object browser to see all objects stored in a [ROOT](#) file including [ROOT](#) trees.

## Usage

```
root.browser(object)
```

## Arguments

`object`            an object of type [SchemeTreeSet](#), [DataTreeSet](#), [ExprTreeSet](#), or [CallTreeSet](#)

## Note

Always select menu item “Quit ROOT” from menu “File” to close the ROOT browser, otherwise you are in the CINT C/C++ interpreter from [ROOT](#). To exit CINT, you need to type “q”.

## Author(s)

Christian Stratowa

---

root.call	<i>Create class CallTreeSet accessing ROOT detection call file</i>
-----------	--

---

### Description

Create class CallTreeSet accessing ROOT detection call file.

### Usage

```
root.call(xps.scheme, rootfile = character(0), treetype = character(0), treenames
```

### Arguments

xps.scheme	A <a href="#">SchemeTreeSet</a> containing the correct scheme for the ROOT data file.
rootfile	name of ROOT data file, including full path.
treetype	tree type.
treenames	optional character vector of tree names to get only subset of trees.

### Details

An S4 class [CallTreeSet](#) will be created, serving as R wrapper to the existing [ROOT](#) detection call file `rootfile`.

Parameter `treetype` must be supplied to identify the ROOT trees for slots `data` and `detcall`. Valid tree types are listed in [validTreetype](#).

To get the names of all trees with their extensions `treetype`, which are stored in `rootfile`, you can call function [getTreeNames](#) first.

If the [CallTreeSet](#) should only handle a subset of the trees stored in `rootfile`, the tree names must be supplied as vector `treenames`.

### Value

A [CallTreeSet](#) object.

### Author(s)

Christian Stratowa

### See Also

[root.data](#), [root.expr](#)

### Examples

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## MAS5 detection call
detcall.mas5 <- mas5.call(data.test3, "tmp_Test3CallAll", tmpdir="", verbose=FALSE)

## use subset of trees
sub.call <- root.call(scheme.test3, "tmp_Test3CallAll.root", "dc5", c("TestA2", "TestB1")
```

---

root.data	<i>Create class DataTreeSet accessing ROOT data file</i>
-----------	--

---

## Description

Create class DataTreeSet accessing ROOT data file.

## Usage

```
root.data(xps.scheme, rootfile = character(0), celnames = "*")
```

## Arguments

xps.scheme	A <a href="#">SchemeTreeSet</a> containing the correct scheme for the ROOT data file.
rootfile	name of ROOT data file, including full path.
celnames	optional character vector of tree names to get only subset of trees.

## Details

An S4 class [DataTreeSet](#) will be created, serving as R wrapper to the existing [ROOT](#) data file `rootfile`.

If the [DataTreeSet](#) should only handle a subset of the trees stored in `rootfile`, the tree names must be supplied as vector `celnames`.

To get the names of all trees stored in `rootfile` you can call function [getTreeNames](#) first.

## Value

A [DataTreeSet](#) object.

## Note

Use `root.data` to access the ROOT data file from new R sessions to avoid creating a new [ROOT](#) data file for every R session.

## Author(s)

Christian Stratowa

## See Also

[import.data](#), [DataTreeSet](#)

## Examples

```
## get scheme and import CEL-files from package
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- import.data(scheme.test3, "tmp_datatest3", celdir=paste(.path.package("xps"),

## use subset of CEL-files
subdata.test3 <- root.data(scheme.test3, "tmp_datatest3_cel.root", celnames=c("TestA1.cel"
```

---

root.density      *ROOT Density Plot*

---

### Description

Creates a ROOT density plot for one or all ROOT tree(s).

### Usage

```
root.density(x, treename = "*", logbase = "log2", canvasname = "DensityPlot", sa
```

### Arguments

x	object of class <a href="#">DataTreeSet</a> or <a href="#">ExprTreeSet</a> .
treename	name of tree, must be present in <code>rootfile</code> of object x.
logbase	usually "log2", or "0", determines if leaf data should be converted to log.
canvasname	name of ROOT canvas
save.as	graphics type for saving canvas, one of "ps", "eps", "pdf", "jpg", "gif", "png", "tiff"
w	the width of the canvas in pixels.
h	the height of the canvas in pixels.

### Details

Creates a ROOT density plot for one or all tree(s) present in `rootfile`.

By selecting menu "File->Save->canvasname.xxx" you can save the figure as e.g. \*.gif, \*.jpg, \*.pdf, \*.ps or even as C++ macro.

Alternatively, you can save the plot by setting `save.as`. However, this will close the canvas immediately after opening it.

### Note

Always select menu item "Quit ROOT" from menu "File" to close the ROOT canvas, otherwise you are in the CINT C/C++ interpreter from [ROOT](#). To exit CINT, you need to type ".q".

### Author(s)

Christian Stratowa

### See Also

[root.hist1D](#)

## Examples

```
## Not run:
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

root.density(data.test3, "*")
root.density(data.test3, "TestA1.cel")
root.density(data.test3, "TestA1.cel", save.as="png")
## End(Not run)
```

---

root.expr

*Create class ExprTreeSet accessing ROOT expression file*

---

## Description

Create class ExprTreeSet accessing ROOT expression file.

## Usage

```
root.expr(xps.scheme, rootfile = character(0), treetype = character(0), treenames
```

## Arguments

xps.scheme	A <a href="#">SchemeTreeSet</a> containing the correct scheme for the ROOT data file.
rootfile	name of ROOT data file, including full path.
treetype	tree type.
treenames	optional character vector of tree names to get only subset of trees.

## Details

An S4 class [ExprTreeSet](#) will be created, serving as R wrapper to the existing [ROOT](#) expression file `rootfile`.

Parameter `treetype` must be supplied to identify the ROOT trees for slot data. Valid tree types are listed in [validTreetype](#).

To get the names of all trees with their extensions `treetype`, which are stored in `rootfile`, you can call function [getTreeNames](#) first.

If the [ExprTreeSet](#) should only handle a subset of the trees stored in `rootfile`, the tree names must be supplied as vector `treenames`.

## Value

A [ExprTreeSet](#) object.

## Author(s)

Christian Stratowa

## See Also

[root.data](#), [root.call](#)

**Examples**

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

# rma
all.rma <- rma(data.test3, "tmp_Test3RMAAll", tmpdir="", background="pmonly", normalize=TRUE,

## use subset of trees
sub.rma <- root.expr(scheme.test3, "tmp_Test3RMAAll.root", "mdp", c("TestA2.mdp", "TestB1
```

---

root.graph1D

*ROOT 1D-Graph*


---

**Description**

Creates a ROOT 1D-graph for a ROOT tree.

**Usage**

```
root.graph1D(x, treename = character(0), logbase = "log2", option = "P", canvasname = character(0))
```

**Arguments**

x	object of class <a href="#">DataTreeSet</a> or <a href="#">ExprTreeSet</a> .
treename	name of tree, must be present in rootfile of object x.
logbase	usually "log2", or "0", determines if leaf data should be converted to log.
option	ROOT TGraph::PaintGraph option, usually one of "P", "*", "L".
canvasname	name of ROOT canvas
save.as	graphics type for saving canvas, one of "ps", "eps", "pdf", "jpg", "gif", "png", "tiff"
w	the width of the canvas in pixels.
h	the height of the canvas in pixels.

**Details**

Creates a ROOT 1D-graph for tree `treename` present in `rootfile`.

By selecting menu "File->Save->canvasname.xxx" you can save the figure as e.g. \*.gif, \*.jpg, \*.pdf, \*.ps or even as C++ macro.

Alternatively, you can save the plot by setting `save.as`. However, this will close the canvas immediately after opening it.

**Note**

Always select menu item "Quit ROOT" from menu "File" to close the ROOT canvas, otherwise you are in the CINT C/C++ interpreter from [ROOT](#). To exit CINT, you need to type ".q".

**Author(s)**

Christian Stratowa

**See Also**

[root.graph2D](#)

**Examples**

```
## Not run:
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

root.graph1D(data.test3, "TestA1.cel")
## End(Not run)
```

---

root.graph2D	<i>ROOT 2D-Graph</i>
--------------	----------------------

---

**Description**

Creates a ROOT 2D-graph for a ROOT tree.

**Usage**

```
root.graph2D(x, treename1 = character(0), treename2 = character(0), logbase = "log2",
```

**Arguments**

x	object of class <a href="#">DataTreeSet</a> or <a href="#">ExprTreeSet</a> .
treename1	name of first tree, must be present in <code>rootfile</code> of object x.
treename2	name of second tree, must be present in <code>rootfile</code> of object x.
logbase	usually "log2", or "0", determines if leaf data should be converted to log.
option	ROOT TGraph::PaintGraph option, usually one of "P", "*", "L".
canvasname	name of ROOT canvas
save.as	graphics type for saving canvas, one of "ps", "eps", "pdf", "jpg", "gif", "png", "tiff"
w	the width of the canvas in pixels.
h	the height of the canvas in pixels.

**Details**

Creates a ROOT 2D-graph for trees `treename1` and `treename2` present in `rootfile`.

By selecting menu "File->Save->canvasname.xxx" you can save the figure as e.g. \*.gif, \*.jpg, \*.pdf, \*.ps or even as C++ macro.

Alternatively, you can save the plot by setting `save.as`. However, this will close the canvas immediately after opening it.

**Note**

Always select menu item "Quit ROOT" from menu "File" to close the ROOT canvas, otherwise you are in the CINT C/C++ interpreter from [ROOT](#). To exit CINT, you need to type ".q".

**Author(s)**

Christian Stratowa

**See Also**[root.graph1D](#), [root.mvaplplot](#)**Examples**

```
## Not run:
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

root.graph2D(data.test3, "TestA1.cel", "TestB1.cel")
## End(Not run)
```

root.hist1D

*ROOT 1D-Histogram***Description**

Creates a ROOT 1D-histogram for a ROOT tree.

**Usage**

```
root.hist1D(x, treename = character(0), logbase = "log2", type = "hist", option
```

**Arguments**

x	object of class <a href="#">DataTreeSet</a> or <a href="#">ExprTreeSet</a> .
treename	name of tree, must be present in <code>rootfile</code> of object x.
logbase	usually “log2”, or “0”, determines if leaf data should be converted to log.
type	ROOT 1D-hist or density, i.e. “hist” or “density”.
option	ROOT 1D-hist option only, usually one of “HIST”, “B”, “C”, “E”.
canvasname	name of ROOT canvas
save.as	graphics type for saving canvas, one of “ps”, “eps”, “pdf”, “jpg”, “gif”, “png”, “tiff”
w	the width of the canvas in pixels.
h	the height of the canvas in pixels.

**Details**Creates a ROOT 1D-histogram for tree `treename` present in `rootfile`.By selecting menu “File->Save->canvasname.xxx” you can save the figure as e.g. `*gif`, `*.jpg`, `*.pdf`, `*.ps` or even as C++ macro.Alternatively, you can save the plot by setting `save.as`. However, this will close the canvas immediately after opening it.



**Note**

Always select menu item “Quit ROOT” from menu “File” to close the ROOT canvas, otherwise you are in the CINT C/C++ interpreter from ROOT. To exit CINT, you need to type “.q”.

**Author(s)**

Christian Stratowa

**See Also**

[root.hist2D](#), [root.hist3D](#)

**Examples**

```
## Not run:
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

root.hist1D(data.test3, "TestA1.cel")
root.hist1D(data.test3, "TestA1.cel", type="density")
## End(Not run)
```

---

root.hist2D

*ROOT 2D-Histogram*

---

**Description**

Creates a ROOT 2D-histogram for a ROOT tree.

**Usage**

```
root.hist2D(x, treename1 = character(0), treename2 = character(0), logbase = "log
```

**Arguments**

x	object of class <a href="#">DataTreeSet</a> or <a href="#">ExprTreeSet</a> .
treename1	name of first tree, must be present in <code>rootfile</code> of object x.
treename2	name of second tree, must be present in <code>rootfile</code> of object x.
logbase	usually “log2”, or “0”, determines if leaf data should be converted to log.
option	ROOT hist TH2 option, usually one of “SCAT”, “COLZ”, “BOX”, “SURF2”, “SURF3”.
canvasname	name of ROOT canvas
save.as	graphics type for saving canvas, one of “ps”, “eps”, “pdf”, “jpg”, “gif”, “png”, “tiff”
w	the width of the canvas in pixels.
h	the height of the canvas in pixels.

**Details**

Creates a ROOT 2D-histogram for trees `treename1` and `treename2` present in `rootfile`.

By selecting menu “File->Save->canvasname.xxx” you can save the figure as e.g. \*.gif, \*.jpg, \*.pdf, \*.ps or even as C++ macro.

Alternatively, you can save the plot by setting `save.as`. However, this will close the canvas immediately after opening it.

**Note**

Always select menu item “Quit ROOT” from menu “File” to close the ROOT canvas, otherwise you are in the CINT C/C++ interpreter from ROOT. To exit CINT, you need to type “.q”.

**Author(s)**

Christian Stratowa

**See Also**

[root.hist1D](#), [root.hist3D](#)

**Examples**

```
## Not run:
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

root.hist2D(data.test3, "TestA1.cel", "TestB1.cel", option="COLZ")
## End(Not run)
```

---

root.hist3D

*ROOT 3D-Histogram*

---

**Description**

Creates a ROOT 3D-histogram for a ROOT tree.

**Usage**

```
root.hist3D(x, treename1 = character(0), treename2 = character(0), treename3 = c
```

**Arguments**

<code>x</code>	object of class <a href="#">DataTreeSet</a> or <a href="#">ExprTreeSet</a> .
<code>treename1</code>	name of first tree, must be present in <code>rootfile</code> of object <code>x</code> .
<code>treename2</code>	name of second tree, must be present in <code>rootfile</code> of object <code>x</code> .
<code>treename3</code>	name of third tree, must be present in <code>rootfile</code> of object <code>x</code> .
<code>logbase</code>	usually “log2”, or “0”, determines if leaf data should be converted to log.
<code>option</code>	ROOT hist TH3 option, usually one of “HIST”, “SCAT”, “BOX”.
<code>canvasname</code>	name of ROOT canvas

save.as	graphics type for saving canvas, one of “ps”, “eps”, “pdf”, “jpg”, “gif”, “png”, “tiff”
w	the width of the canvas in pixels.
h	the height of the canvas in pixels.

### Details

Creates a ROOT 3D-histogram for trees `treename1`, `treename2` and `treename3` present in `rootfile`. By selecting menu “File->Save->canvasname.xxx” you can save the figure as e.g. `*gif`, `*.jpg`, `*.pdf`, `*.ps` or even as C++ macro.

By moving the mouse into the middle of the canvas, the cursor changes and you can rotate the 3D-histogram. By selecting menu “View->View With->OpenGL” the OpenGL viewer opens, where you can rotate the 3D-histogram interactively.

Alternatively, you can save the plot by setting `save.as`. However, this will close the canvas immediately after opening it.

### Note

Always select menu item “Quit ROOT” from menu “File” to close the ROOT canvas, otherwise you are in the CINT C/C++ interpreter from ROOT. To exit CINT, you need to type “.q”.

### Author(s)

Christian Stratowa

### See Also

[root.hist1D](#), [root.hist2D](#)

### Examples

```
## Not run:
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

root.hist3D(data.test3, "TestA1.cel", "TestB2.cel", "TestB1.cel", option="BOX")
## End(Not run)
```

---

root.image

*ROOT Image*

---

### Description

Creates a ROOT image for a ROOT tree.

### Usage

```
root.image(x, treename = character(0), leafname = "fInten", logbase = "log2", op
```

**Arguments**

x	object of class <code>DataTreeSet</code> .
treename	name of tree, must be present in <code>rootfile</code> of object x.
leafname	leaf name of tree, usual “fInten” or “fBg”.
logbase	usually “log2”, or “0”, determines if leaf data should be converted to log.
option	ROOT graph option, usually. one of “COL”, “COLZ”.
zlim	size limits c(min,max) of leafname.
canvasname	name of ROOT canvas
save.as	graphics type for saving canvas, one of “ps”, “eps”, “pdf”, “jpg”, “gif”, “png”, “tiff”
w	the width of the device in pixels.
h	the height of the device in pixels.

**Details**

Creates a ROOT image for tree `treename` present in `rootfile`.

To zoom-in move the mouse cursor to the x-axis (y-axis) until it changes to a hand and click-drag to select an axis-range. To unzoom move the mouse cursor to the x-axis (y-axis) until it changes to a hand and right-click to select “Unzoom”.

By selecting menu “File->Save->canvasname.xxx” you can save the figure as e.g. \*.gif, \*.jpg, \*.pdf, \*.ps or even as C++ macro.

Alternatively, you can save the plot by setting `save.as`. However, this will close the canvas immediately after opening it.

**Note**

Always select menu item “Quit ROOT” from menu “File” to close the ROOT canvas, otherwise you are in the CINT C/C++ interpreter from ROOT. To exit CINT, you need to type “.q”.

**Author(s)**

Christian Stratowa

**See Also**

[image-methods](#), [image](#)

**Examples**

```
## Not run:
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

root.image(data.test3, "TestA1.cel")
root.image(data.test3, "TestA1.cel", save.as="png")
## End(Not run)
```

---

root.merge.data      *Create class DataTreeSet by merging ROOT data files*

---

### Description

Create class DataTreeSet by merging different ROOT data files.

### Usage

```
root.merge.data(xps.scheme, rootfiles = list(), celnames = "*")
```

### Arguments

xps.scheme      A [SchemeTreeSet](#) containing the correct scheme for the ROOT data file.  
rootfiles      list of ROOT data file(s), including full path.  
celnames      optional character vector of tree names to get only subset of trees.

### Details

This function allows to merge data trees from different existing ROOT data files.

An S4 class [DataTreeSet](#) will be created, serving as R wrapper to the existing [ROOT](#) data file(s) rootfiles.

If the [DataTreeSet](#) should only handle a subset of the trees stored in rootfiles, the tree names must be supplied as vector celnames.

To get the names of all trees stored in separate rootfiles you can call function [getTreeNames](#) first.

### Value

A DataTreeSet object.

### Author(s)

Christian Stratowa

### See Also

[root.data](#), [DataTreeSet](#)

### Examples

```
## get scheme and import CEL-files from package
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- import.data(scheme.test3, "tmp_datatest3", celdir=paste(.path.package("xps"),

## get subset of CEL-files
subdataA <- root.data(scheme.test3, "tmp_datatest3_cel.root", celnames=c("TestA1.cel", "Tes
subdataB <- root.data(scheme.test3, "tmp_datatest3_cel.root", celnames=c("TestB1.cel", "Tes

## merge data
dataAB <- root.merge.data(scheme.test3, c(rootFile(subdataA), rootFile(subdataB)), celnames
```

---

root.mvaplot                    *ROOT M vs A Plot*

---

### Description

Creates a ROOT M vs A plot for a ROOT tree.

### Usage

```
root.mvaplot(x, treename1 = character(0), treename2 = character(0), logbase = "10")
```

### Arguments

x	object of class <a href="#">ExprTreeSet</a> or <a href="#">DataTreeSet</a> .
treename1	name of first tree, must be present in <code>rootfile</code> of object x.
treename2	name of second tree, must be present in <code>rootfile</code> of object x.
logbase	usually "log2", or "0", determines if leaf data should be converted to log.
option	ROOT TGraph::PaintGraph option, usually one of "P", "*".
canvasname	name of ROOT canvas
save.as	graphics type for saving canvas, one of "ps", "eps", "pdf", "jpg", "gif", "png", "tiff"
w	the width of the canvas in pixels.
h	the height of the canvas in pixels.

### Details

Creates a ROOT M vs A plot for trees `treename1` and `treename2` present in `rootfile`.

By selecting menu "File->Save->canvasname.xxx" you can save the figure as e.g. \*.gif, \*.jpg, \*.pdf, \*.ps or even as C++ macro.

Alternatively, you can save the plot by setting `save.as`. However, this will close the canvas immediately after opening it.

### Note

Always select menu item "Quit ROOT" from menu "File" to close the ROOT canvas, otherwise you are in the CINT C/C++ interpreter from [ROOT](#). To exit CINT, you need to type ".q".

### Author(s)

Christian Stratowa

### See Also

[root.graph1D](#)

**Examples**

```
## Not run:
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

# compute RMA
data.rma <- rma(data.test3, "Test3RMA", tmpdir="", background="pmonly", normalize=TRUE)

root.mvaplot(data.rma, "TestA1.mdp", "TestB1.mdp")
## End(Not run)
```

---

root.profile	<i>ROOT Profile Plot</i>
--------------	--------------------------

---

**Description**

Creates a ROOT profile plot, i.e. a plot of parallel coordinates

**Usage**

```
root.profile(x, treename = "*", varlist = NULL, as.log = TRUE, globalscale = TRUE)
```

**Arguments**

x	S4 object, usually of class <a href="#">DataTreeSet</a> or <a href="#">ExprTreeSet</a> .
treename	name of tree, usually all trees present in rootfile of object x.
varlist	leaf name of tree, usual "fInten" or "fLevel".
as.log	logical indicating if varlist should be drawn as logarithmic data.
globalscale	logical indicating if all axes should be drawn at the same scale.
boxes	logical indicating if box-and-whisker plots should be drawn.
ylim	size limits c(min,max) of varlist.
canvasname	name of ROOT canvas
save.as	graphics type for saving canvas, one of "ps", "eps", "pdf", "jpg", "gif", "png", "tiff"
w	the width of the device in pixels.
h	the height of the device in pixels.

**Details**

Creates a ROOT profile plot for all trees `treename="*" present in rootfile. In this case varlist must be the name of one tree leaf only; for varlist=NULL leaf "fInten" will be used for class DataTreeSet and leaf "fLevel" will be used for class ExprTreeSet. If treename is the name of one tree only then varlist can contain up to all leaves of the tree, separated by colons, e.g. varlist="fLevel:fStdev".`

For `boxes=TRUE` the profile plot draws box-and-whisker plots and can thus be considered the equivalent of the usual `boxplot`.

A ROOT profile plot, i.e. a plot of parallel coordinates, is drawn in a “TreeViewer”, a graphic user interface designed to handle ROOT trees. You can activate context menus by right-clicking on items or inside the right panel.

The “TreeViewer” is explained in <http://root.cern.ch/root/html/TTreeViewer.html>.

By selecting menu “File->Save->canvasname.xxx” you can save the figure as e.g. \*.gif, \*.jpg, \*.pdf, \*.ps or even as C++ macro.

Alternatively, you can save the plot by setting `save.as`. However, this will close the canvas immediately after opening it.

### Note

Always select menu item “Quit ROOT” from menu “File” to close the ROOT tree viewer, otherwise you are in the CINT C/C++ interpreter from ROOT. To exit CINT, you need to type “.q”.

### Author(s)

Christian Stratowa

### Examples

```
## Not run:
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

root.profile(data.test3)
## End(Not run)
```

---

ROOT

*ROOT An Object-Oriented Data Analysis Framework*

---

### Description

ROOT system overview

### Details

ROOT is a modular object-oriented framework aimed at solving the data analysis challenges of high-energy physics. The relevant features of ROOT are as follows:

**Architecture:** The ROOT architecture is a layered class hierarchy with over 500 classes divided into different categories. Most of the classes inherit from a common base class TObj, which provides the default behavior and protocol for all objects.

**ROOT Files:** Object input/output is handled by class TFile, which has a UNIX-like directory structure and provides a hierarchical sequential and direct access persistent object store. ROOT files store information in a machine independent format and support on-the-fly data compression. Furthermore, ROOT files are self-describing: for every object stored in TFile, a dictionary describing the corresponding class is written to the file. A dictionary generator, called ROOTCINT, parses the class header files and generates a dictionary. Note: TFile can be considered to be the ROOT analogon to an R environment.



**Data Trees:** Any object derived from TObject can be written to a file with an associated key TKey. However, each key has an overhead in the directory structure in memory. To reduce this overhead, a novel concept, called Trees (class TTree) has been developed. Trees are designed to support very large numbers of complex objects in a large number of files. A Tree consists of branches (TBranch) with each branch described by its leaves (TLeaf). Trees allow direct and random access to any entry of a selected subset of branches. Thus, Trees extend and replace the usual data tables. The concept of Tree friends allows the joining of many trees as one virtual tree. However, unlike table joins in an RDBMS, the processing time is independent of the number of tree friends. Note: TTree can be considered to be the ROOT analogon to an R data.frame.

**CINT:** CINT is an interactive C/C++ interpreter, which is aimed at processing C/C++ scripts, called macros. Currently, CINT covers 99% of ANSI C and 95% of ANSI C++. CINT offers a gdb-like debugger for interpreted programs and allows the automatic compilation of scripts using ACLiC, the automatic compiler of libraries for CINT. Although available as independent program, CINT is embedded in ROOT as command line interpreter and macro processor, as well as dictionary generator.

**User interaction:** The ROOT system can be accessed from the command line, by writing macros, or via a graphic user interface (e.g. RootBrowser). Furthermore, it is possible to write libraries and applications. The ROOT GUI classes allow the development of full-featured standalone applications. Note: A macro can be considered to be the ROOT analogon of an R script. The RootBrowser can be opened using function `root.browser`

**Platform independence:** The ROOT system is available for most platforms and operating systems, including Linux, MacOS X, and the major flavors of UNIX and Windows. ROOT and ROOT-derived applications can be compiled for any supported platform.

### Author(s)

The ROOT team <http://root.cern.ch/root/Authors.html>

### References

ROOT User Guide <http://root.cern.ch/root/doc/RootDoc.html>

ROOT publications <http://root.cern.ch/root/Publications.html>

Christian Stratowa (2003), Distributed Storage and Analysis of Microarray Data in the Terabyte Range: An Alternative to BioConductor <http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/Stratowa.pdf>

---

root.scheme

*Create class SchemeTreeSet accessing ROOT scheme file*

---

### Description

Create class SchemeTreeSet accessing ROOT scheme file.

### Usage

```
root.scheme(rootfile = character(0), add.mask = FALSE)
```

### Arguments

rootfile	name of ROOT scheme file, including full path.
add.mask	if TRUE mask information will be included as slot mask.

## Details

An S4 class `SchemeTreeSet` will be created, serving as R wrapper to the `ROOT` scheme file `rootfile`.

## Value

A `SchemeTreeSet` object.

## Note

Use this function to access the `ROOT` scheme file from new R sessions to avoid creating a new `ROOT` scheme file for every R session.

Do not set `add.mask=TRUE` for exon arrays unless you know that your computer has sufficient RAM.

## Author(s)

Christian Stratowa

## See Also

`import.expr.scheme`, `import.exon.scheme`, `SchemeTreeSet`

## Examples

```
## create class SchemeSet to access the ROOT scheme file for the Test3 GeneChip
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/",
str(scheme.test3)

## Not run:
## scheme set for existing human root exon scheme file
scheme.huex10stv2r2.na22 <- root.scheme("/my/path/schemes/Scheme_HuEx10stv2r2_na22.root")
## End(Not run)
```

---

`SchemeTreeSet-class`

*Class SchemeTreeSet*

---

## Description

This class provides the link to the `ROOT` scheme file and the `ROOT` trees contained therein. It extends class `TreeSet`.

## Objects from the Class

Objects can be created using the functions `import.expr.scheme`, `import.exon.scheme`, `import.genome.scheme` or `root.scheme`.

**Slots**

- chipname:** Object of class "character" representing the Affymetrix chip name.
- chiptype:** Object of class "character" representing the chip type, either 'GeneChip', 'GenomeChip' 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.
- setname:** Object of class "character" representing the name to the [ROOT](#) file subdirectory where the [ROOT](#) scheme trees are stored; it is identical to chipname.
- settype:** Object of class "character" describing the type of treeset stored in setname, i.e. 'scheme'.
- rootfile:** Object of class "character" representing the name of the [ROOT](#) scheme file, including 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 subdirectory setname.
- treenames:** Object of class "list" representing the names of the [ROOT](#) trees stored in subdirectory setname.

**Extends**

Class "[TreeSet](#)", directly.

**Methods**

- attachMask** signature(object = "SchemeTreeSet"): exports scheme tree from [ROOT](#) scheme file and saves as data.frame mask.
- chipMask** signature(object = "SchemeTreeSet"): extracts data.frame mask.
- chipMask<-** signature(object = "SchemeTreeSet", value = "data.frame"): replaces data.frame mask.
- chipName** signature(object = "SchemeTreeSet"): extracts slot chipname.
- chipType** signature(object = "SchemeTreeSet"): extracts slot chiptype.
- chipType<-** signature(object = "SchemeTreeSet", value = "character"): replaces slot chiptype.
- export** signature(object = "SchemeTreeSet"): exports [ROOT](#) trees as text file, see [export-methods](#).
- ncols** signature(object = "SchemeTreeSet"): extracts the physical number of array columns from slot probeinfo.
- nrows** signature(object = "SchemeTreeSet"): extracts the physical number of array rows from slot probeinfo.
- probeInfo** signature(object = "SchemeTreeSet"): extracts slot probeinfo.
- removeMask** signature(object = "SchemeTreeSet"): replaces data.frame mask with an empty data.frame of dim(0,0).

**Author(s)**

Christian Stratowa

**Examples**

```
showClass("SchemeTreeSet")
```

---

```
summarize
```

*Probe Set Summarizing Functions*

---

**Description**

Converts Affymetrix probe level data to expression levels by summarizing the probe set values into one expression measure and a standard error for this summary.

**Usage**

```
summarize(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE, select = "all", method = "rma", option = "transcript", logbase = "log", exonlevel = 1, params = list(), xps.scheme = "standard", add.data = FALSE, verbose = FALSE, object, ...)
```

```
summarize.mas4(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE, select = "all", method = "rma", option = "transcript", logbase = "log", exonlevel = 1, params = list(), xps.scheme = "standard", add.data = FALSE, verbose = FALSE, object, ...)
```

```
summarize.mas5(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE, select = "all", method = "rma", option = "transcript", logbase = "log", exonlevel = 1, params = list(), xps.scheme = "standard", add.data = FALSE, verbose = FALSE, object, ...)
```

```
summarize.rma(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE, select = "all", method = "rma", option = "transcript", logbase = "log", exonlevel = 1, params = list(), xps.scheme = "standard", add.data = FALSE, verbose = FALSE, object, ...)
```

```
xpsSummarize(object, ...)
```

**Arguments**

<code>xps.data</code>	object of class <code>DataTreeSet</code> .
<code>filename</code>	file name of ROOT data file.
<code>filedir</code>	system directory where ROOT data file should be stored.
<code>tmpdir</code>	optional temporary directory where temporary ROOT files should be stored.
<code>update</code>	logical. If <code>TRUE</code> the existing ROOT data file <code>filename</code> will be updated.
<code>select</code>	type of probes to select for summarization.
<code>method</code>	summarization method to use.
<code>option</code>	option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon arrays only.
<code>logbase</code>	logarithm base as character, one of '0', 'log', 'log2', 'log10'.
<code>exonlevel</code>	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
<code>params</code>	vector of parameters for summarization method.
<code>xps.scheme</code>	optional alternative <code>SchemeTreeSet</code> .
<code>add.data</code>	logical. If <code>TRUE</code> expression data will be included as slot data.
<code>verbose</code>	logical, if <code>TRUE</code> print status information.
<code>object</code>	object of class <code>DataTreeSet</code> .
<code>...</code>	the arguments described above.

## Details

Converts Affymetrix probe level data to expression levels by summarizing the probe set values into one expression measure and a standard error for this summary.

`xpsSummarize` is the `DataTreeSet` method called by function `summarize`, containing the same parameters.

## Value

An `ExprTreeSet`.

## Author(s)

Christian Stratowa

## See Also

[express](#)

## Examples

```
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## RMA background
data.bg.rma <- bgcorrect.rma(data.test3, "tmp_Test3RMA", filedir=getwd(), tmpdir="", verbose=
## normalize quantiles
data.qu.rma <- normalize.quantiles(data.bg.rma, "tmp_Test3RMA", filedir=getwd(), tmpdir="", u
## summarize medianpolish
data.mp.rma <- summarize.rma(data.qu.rma, "tmp_Test3RMA", filedir=getwd(), tmpdir="", update=

## get expression data.frame
expr.rma <- exprs(data.mp.rma)
head(expr.rma)

## plot expression levels
if (interactive()) {
  boxplot(data.mp.rma)
  boxplot(log2(expr.rma[,3:6]))
}
```

---

TreeSet-class

*Class TreeSet*

---

## Description

This is the virtual base class for all other classes providing the link to a [ROOT](#) file and the [ROOT](#) trees contained therein.

## Objects from the Class

A virtual Class: No objects may be created from it.

**Slots**

**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 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 subdirectory setname.

**treenames:** Object of class "list" representing the names of the [ROOT](#) trees stored in subdirectory setname.

**Methods**

**export** signature(object = "TreeSet"): exports [ROOT](#) trees as text file, see [export-methods](#).

**fileDir** signature(object = "TreeSet"): extracts slot filedir.

**fileDir<-** signature(object = "TreeSet", value = "character"): replaces slot filedir.

**root.browser** signature(object = "TreeSet"): opens the [ROOT](#) file browser.

**rootFile** signature(object = "TreeSet"): extracts slot rootfile.

**rootFile<-** signature(object = "TreeSet", value = "character"): replaces slot rootfile.

**setName** signature(object = "TreeSet"): extracts slot setname.

**setName<-** signature(object = "TreeSet", value = "character"): replaces slot setname.

**setType** signature(object = "TreeSet"): extracts slot settype.

**setType<-** signature(object = "TreeSet", value = "character"): replaces slot settype.

**treeNames** signature(object = "TreeSet"): extracts slot treenames.

**Author(s)**

Christian Stratowa

**See Also**

derived classes [SchemeTreeSet](#), [DataTreeSet](#), [ExprTreeSet](#), [CallTreeSet](#).

**Examples**

```
showClass("TreeSet")
```

---

`type2Exten`*Convert Method Type to Tree Extension*

---

**Description**

Convert Method Type to Tree Extension.

**Usage**

```
type2Exten(type, datatype)
```

**Arguments**

<code>type</code>	method type.
<code>datatype</code>	data type.

**Details**

For every `datatype` different methods, i.e. algorithms exist which can be applied. Valid `datatypes` are 'preprocess' and 'normation'.

For `datatype` 'preprocess' the following methods can be applied:

<code>mean:</code>	trimmed mean
<code>median:</code>	median
<code>quantile:</code>	quantile
<code>tukeybiweight:</code>	tukey biweight
<code>medianpolish:</code>	median polish

For `datatype` 'normation' the following methods can be applied:

<code>mean:</code>	trimmed mean
<code>median:</code>	median
<code>quantile:</code>	quantile
<code>lowess:</code>	lowess
<code>supsmu:</code>	supsmu

The tree extensions are described in [validTreetype](#).

**Value**

A character with the correct tree extension.

**Author(s)**

Christian Stratowa

**See Also**

[getDatatype](#), [validTreetype](#)

**Examples**

```
type2Exten("quantile", "preprocess")
type2Exten("medianpolish", "preprocess")
type2Exten("supsmu", "normation")
```

---

UniFilter-class      *Class UniFilter*

---

**Description**

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

**Objects from the Class**

Objects can be created by calls of the form `new("UniFilter", ...)`. Alternatively, the constructor `UniFilter` can be used.

**Slots**

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

**Extends**

Class "`Filter`", directly.

**Methods**

**callFilter** signature(object = "UniFilter"): extracts slot `prescall`.  
**callFilter<-** signature(object = "UniFilter", value = "character"): replaces slot `prescall` with character vector `c(cutoff, samples, condition)`.  
**fcFilter** signature(object = "UniFilter"): extracts slot `foldchange`.  
**fcFilter<-** signature(object = "UniFilter", value = "numeric"): replaces slot `foldchange` with numeric vector `c(cutoff, direction)`.  
**uniTest** signature(object = "UniFilter"): extracts slot `unitest`.  
**uniTest<-** signature(object = "UniFilter", value = "character"): replaces slot `unitest` with character vector `c(type, alternative, correction, numperm, mu, paired, conflevel, varequ)`.  
**unitestFilter** signature(object = "UniFilter"): extracts slot `unifilter`.  
**unitestFilter<-** signature(object = "UniFilter", value = "character"): replaces slot `unifilter` with character vector `c(cutoff, variable)`.

**Author(s)**

Christian Stratowa



**See Also**

related classes [Filter](#), [PreFilter](#).

**Examples**

```
unifltr <- new("UniFilter", unitest=list("t.test"))
fcFilter(unifltr) <- c(1.5, "both")
unitestFilter(unifltr) <- c(0.01, "pval")
str(unifltr)
```

---

UniFilter-constructor

*Constructor for Class UniFilter*

---

**Description**

Constructor for class UniFilter allows to apply different unitest filters to class [ExprTreeSet](#), i.e. to the expression level data.frame data.

**Usage**

```
UniFilter(unitest      = "t.test",
          foldchange   = character(),
          prescall     = character(),
          unifilter    = character())
```

**Arguments**

unitest        "character" vector describing parameters for [uniTest](#).  
foldchange    "character" vector describing parameters for [fcFilter](#).  
prescall      "character" vector describing parameters for [callFilter](#).  
unifilter     "character" vector describing parameters for [unitestFilter](#).

**Details**

The UniFilter constructor allows to apply the following unitest filters to class [ExprTreeSet](#):

unitest:        character vector c(type,alternative,correction.numperm,mu,paired,conflevel,varequ).  
foldchange:    character vector c(cutoff,direction).  
prescall:      character vector c(cutoff,samples,condition).  
unifilter:     character vector c(cutoff,variable).

**Value**

An object of type "[UniFilter](#)"

**Note**

Function `UniFilter` is used as constructor for class `UniFilter` so that the user need not know details for creating S4 classes.

**Author(s)**

Christian Stratowa

**See Also**[UniFilter](#), [PreFilter](#)**Examples**

```
## fill character vectors within constructor
unifltr <- UniFilter(unitest=c("t.test", "two.sided", "none", 0, 0.0, FALSE, 0.95, TRUE),
                    foldchange=c(1.3, "both"), unifilter=c(0.1, "pval"))
str(unifltr)

## alternatively add character vectors as methods after creation of constructor
unifltr <- UniFilter()
fcFilter(unifltr) <- c(1.5, "both")
unitestFilter(unifltr) <- c(0.01, "pval")
str(unifltr)
```

unifilter

*Function for Applying an UniFilter to an ExprTreeSet***Description**

This function applies an [UniFilter](#) to an [ExprTreeSet](#).

**Usage**

```
unifilter(xps.expr,
          filename = character(0),
          filedir  = getwd(),
          filter   = NULL,
          minfilters = 999,
          logbase  = "log2",
          group    = character(0),
          treename = "UniTest",
          xps.fltr = NULL,
          xps.call = NULL,
          update   = FALSE,
          verbose  = TRUE)

xpsUniFilter(object, ...)
```

**Arguments**

<code>xps.expr</code>	object of class <code>ExprTreeSet</code> .
<code>filename</code>	file name of ROOT filter file.
<code>filedir</code>	system directory where ROOT filter file should be stored.
<code>filter</code>	object of class <code>UniFilter</code> .
<code>minfilters</code>	minimum number of initialized filter methods to satisfy (default is all filters).

logbase	convert data to logarithm of base: "0", "log", "log2" (default), "log10"
group	a character vector assigning the trees of <code>xps.expr</code> to one of two groups.
treename	tree name to be used in ROOT filter file.
<code>xps.fltr</code>	optional object of class <code>FilterTreeSet</code> .
<code>xps.call</code>	optional object of class <code>CallTreeSet</code> .
update	logical. If TRUE the existing ROOT filter file <code>filename</code> will be updated.
verbose	logical, if TRUE print status information.
object	object of class <code>ExprTreeSet</code> .
...	same arguments as function <code>unifilter</code> .

### Details

This function applies the different filters initialized with constructor `UniFilter` to the `ExprTreeSet` `xps.expr`.

Slot `minfilters` determines the minimum number of initialized filters, which must be satisfied so that the mask is set to `flag=1`. For `minfilters=1` at least one filter must be satisfied, equivalent to logical 'OR'; for `minfilters=999` all filters must be satisfied, equivalent to logical 'AND'.

If pre-filtering should be done before applying function `unifilter` then a `FilterTreeSet` `xps.fltr` must be supplied, created with function `prefilter`.

If method `callFilter` was initialized with constructor `UniFilter` then `CallTreeSet` `xps.call` must be supplied, usually created with function `mas5.call`.

### Value

An `AnalysisTreeSet`

### Note

Internally, slot `group` will be converted to integer values using `as.integer(as.factor(group))`, thus `group=c("GrpA", "GrpA", "GrpB", "GrpB")` will result in a fold-change of `fc=mean(GrpB)/mean(GrpA)`.

### Author(s)

Christian Stratowa

### See Also

`UniFilter`, `prefilter`

### Examples

```
## Not run:
## first, load ROOT scheme file and ROOT data file
scheme.test3 <- root.scheme(paste(.path.package("xps"), "schemes/SchemeTest3.root", sep="/")
data.test3 <- root.data(scheme.test3, paste(.path.package("xps"), "rootdata/DataTest3_cel.

## second, create an ExprTreeSet
data.rma <- rma(data.test3, "tmp_Test3_RMA", tmpdir="", background="pmonly", normalize=TRUE, v
## note: do not copy/paste this code, it is necessary only because R CMD check fails sinc
data.rma@rootfile <- paste(.path.package("xps"), "rootdata/tmp_Test3RMA.root", sep="/")
data.rma@filedir <- paste(.path.package("xps"), "rootdata", sep="/")
```

```
## third, construct an UniFilter
unifltr <- UniFilter(unitest=c("t.test", "two.sided", "none", 0, 0.0, FALSE, 0.95, TRUE), foldcha

## finally, create an AnalysisTreeSet
rma.ufr <- unifilter(data.rma, "tmp_Test3Unifilter", getwd(), unifltr, group=c("GrpA", "GrpA",
str(rma.ufr)
## End(Not run)
```

---

unitestFilter-methods

*Unitest Filter*

---

## Description

This method initializes the Unitest Filter.

Applying an unitest such as the `t.test` to two groups returns the p-value for the test and the value of the t-statistic. The Unitest Filter allows to select only rows satisfying e.g. a certain p-value as cutoff.

The Unitest Filter flags all rows with: `flag = (variable <= cutoff)`

### Usage

```
unitestFilter(object)
unitestFilter(object, value)<-
```

## Arguments

<code>object</code>	object of class <code>UniFilter</code> .
<code>value</code>	character vector <code>c(cutoff, variable)</code> .

## Details

The method `unitestFilter` initializes the following parameters:

<code>cutoff</code> :	the cutoff level for the filter.
<code>variable</code> :	<code>variable="pval"</code> (default): p-value.
	<code>variable="stat"</code> : univariate statistic.
	<code>variable="padj"</code> : optional adjusted p-value.
	<code>variable="pcha"</code> : optional p-value obtained by permutations.

## Value

An initialized `UniFilter` object.

## Author(s)

Christian Stratowa

## Examples

```
unifltr <- UniFilter()
unitestFilter(unifltr) <- c(0.01, "pval")
str(unifltr)
```

---

uniTest-methods      *A Two-Group Unittest*


---

## Description

Unittest performs a a two group uni-test such as the `t.test` on each row of the expression dataframe. The Unittest returns a dataframe containing the results of the test.

### Usage

```
uniTest(object)
uniTest(object, value)<-
```

## Arguments

<code>object</code>	object of class <code>UniFilter</code> .
<code>value</code>	character vector <code>c(type, alternative, correction, numperm, mu, paired, conflevel, varequ)</code>

## Details

The method `uniTest` initializes the following parameters:

<code>type</code> :	a character string specifying the type of test: currently <code>"t.test"</code> (default) or <code>"normal.test"</code> .
<code>alternative</code> :	a character string specifying the alternative hypothesis, must be one of <code>"two.sided"</code> (default), <code>"s.paired"</code> , <code>"s.upper"</code> or <code>"s.lower"</code> .
<code>correction</code> :	a correction to adjust p-values for multiple comparisons: <code>correction="none"</code> : no correction (default). <code>correction="bonferroni"</code> : Bonferroni correction. <code>correction="BH" or "fdr"</code> : correction for false discovery rate (Benjamini & Hochberg). <code>correction="BY"</code> : correction for false discovery rate (Benjamini & Yekutieli). <code>correction="hochberg"</code> : Hochberg correction. <code>correction="holm"</code> : Holm correction. <code>correction="wy"</code> : Westfall-Young step-down adjusted p-chance (E.Manduchi).
<code>numperm</code> :	optional number of permutations used to determine p-chance (default is 0).
<code>mu</code> :	a number indicating the true value of the difference in means for a two sample test (default is 0).
<code>paired</code> :	a logical indicating whether you want a paired uni-test (default is FALSE).
<code>conflevel</code> :	confidence level of the interval (default is 0.95).
<code>varequ</code> :	a logical variable indicating whether to treat the two variances as being equal. If TRUE then the pooled variance is used.

## Value

An initialized `UniFilter` object.

## Author(s)

Christian Stratowa

## References

Benjamini, Y., and Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series B*, **57**, 289–300.

Benjamini, Y., and Yekutieli, D. (2001). The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics* **29**, 1165–1188.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, **6**, 65–70.

Westfall P.H. and Young S.S. (1993) Resampling-based multiple testing: examples and methods for p-value adjustment. *Wiley series in probability and mathematical statistics*; Wiley.

Dudoit S., Yang Y.H., Callow M.J., Speed T.P. (2000) Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Technical report 578*; UC Berkeley.

Manduchi E. (2000) Software: tpWY, see: <http://www.cbil.upenn.edu/tpWY/>

## Examples

```
unifltr <- UniFilter()
uniTest(unifltr) <- c("t.test", "two.sided", "none", 0, 0.0, FALSE, 0.98, TRUE)
str(unifltr)
```

---

validData-methods *Extract Subset of Data*

---

## Description

Extracts a subset of valid data from data.frame data.

### Usage

```
validData(object, which = "")
```

## Arguments

object            object of class DataTreeSet.  
which            type of probes to be returned.

## Details

For expression arrays, validData returns all the perfect match or mismatch probes on the arrays the object represents as data.frame, i.e. which can have the following values:

pm:            perfect match probes.  
mm:            mismatch probes.  
both:          both perfect match and mismatch probes.

For exon arrays, validData returns the probes of the different exon levels as data.frame, i.e. which can have one of the following values:

core:            probesets supported by RefSeq and full-length GenBank transcripts.  
metacore:        core meta-probesets.  
extended:        probesets with other cDNA support.  
metaextended:    extended meta-probesets.  
full:            probesets supported by gene predictions only.  
metafull:        full meta-probesets.

affx: standard AFX controls.  
 all: combination of above.  
 genomic: genomic background probes.  
 antigenomic: antigenomic background probes.

**Value**

A `data.frame`.

**Author(s)**

Christian Stratowa

**See Also**

[pm](#), [mm](#)

---

validTreetype	<i>Validate Tree Type</i>
---------------	---------------------------

---

**Description**

Validate tree type for corresponding data type.

**Usage**

```
validTreetype(treetype, datatype)
```

**Arguments**

treetype      tree type.  
 datatype      data type.

**Details**

Every `ROOT` tree has an extension, which describes the type of data stored in this tree. For example, 'TestA1.cel' is the tree name that stores the CEL-file data for 'TestA1.CEL'.

Trees with `datatype="scheme"` have the following extensions:

`scm`: scheme tree containing (x,y)-coordinates and mask for `UNIT_ID`.

`idx`: unit tree containing `UnitName` (i.e. probeset id), `NumCells`, `NumAtoms`, `UnitType`, for `UNIT_ID`.

`prb`: probe tree containing probe sequences.

`ann`: transcript annotation tree.

`anx`: exon annotation tree; exon arrays only.

`anp`: probeset annotation tree; exon arrays only.

`cxy`: coordinate tree containing CLF-file information; exon arrays only.

`exn`: exon tree; exon arrays only.

`pbs`: probeset tree; exon arrays only.

Trees with `datatype="rawdata"` have the following extensions:

`cel`: data tree containing CEL-file data.

Trees with `datatype="preprocess"` have the following extensions:

`int`: intensity tree containing background-corrected intensities.  
`sbg`: background tree containing MAS4 sector background levels.  
`wbg`: background tree containing MAS5 weighted sector background levels.  
`rbg`: background tree containing RMA background levels.  
`gbg`: background tree containing GC-content background levels.  
`cmn`: cell tree containing preprocessed intensities using algorithm 'mean'.  
`cmd`: cell tree containing preprocessed intensities using algorithm 'median'.  
`clw`: cell tree containing preprocessed intensities using algorithm 'lowess'.  
`css`: cell tree containing preprocessed intensities using algorithm 'supsmu'.  
`cqu`: cell tree containing preprocessed intensities using algorithm 'quantile'.  
`dc5`: detection tree containing MAS5 detection call and p-value.  
`dab`: detection tree containing DABG detection call and p-value.  
`amn`: expression tree containing expression levels computed with 'arithmetic mean'.  
`gmn`: expression tree containing expression levels computed with 'geometric mean'.  
`wmn`: expression tree containing expression levels computed with 'weighted mean'.  
`wdf`: expression tree containing expression levels computed with 'weighted difference'.  
`adf`: expression tree containing expression levels computed with 'average difference'.  
`tbw`: expression tree containing expression levels computed with 'tukey biweight'.  
`mdp`: expression tree containing expression levels computed with 'median polish'.

Trees with `datatype="normalization"` have the following extensions:

`tmn`: expression tree after normalization using algorithm 'trimmed mean'.  
`med`: expression tree after normalization using algorithm 'median'.  
`ksm`: expression tree after normalization using algorithm 'kernel smoother'.  
`low`: expression tree after normalization using algorithm 'lowess'.  
`sup`: expression tree after normalization using algorithm 'supsmu'.  
`qua`: expression tree after normalization using algorithm 'quantile'.  
`mdp`: expression tree after normalization using algorithm 'median polish'.

## Value

Returns the valid `treetype`, otherwise an error message is returned.

## Note

Not all tree types are used in the current package.

## Author(s)

Christian Stratowa

## See Also

[getDatatype](#), [type2Exten](#)

## Examples

```
validTreetype("prb", "scheme")
validTreetype("cel", "rawdata")
validTreetype("tbw", "preprocess")
```



---

varFilter-methods *Variance Filter*

---

### Description

This method initializes the Variance Filter.

The Variance Filter flags all rows with: `flag = (var/mean >= cutoff)`

#### Usage

```
varFilter(object)
varFilter(object, value)<-
```

### Arguments

`object`            object of class `PreFilter`.  
`value`            numeric vector `c(cutoff, trim, epsilon)`.

### Details

The method `varFilter` initializes the following parameters:

```
cutoff:    the cutoff level for the filter.
trim:      the trim value for trimmed mean (default is trim=0).
epsilon:   value to replace mean (default is epsilon=0.01):
           epsilon > 0: replace mean=0 with epsilon.
           epsilon = 0: always set mean=1.
```

Note, that for `epsilon = 0` the filter flags all rows with: `variance >= cutoff`

### Value

An initialized `PreFilter` object.

### Author(s)

Christian Stratowa

### Examples

```
prefltr <- PreFilter()
varFilter(prefltr) <- c(0.6,0.02,0.01)
str(prefltr)
```

---

volcanoplot-methods

*Volcano Plot*

---

**Description**

Produce a scatter plot of fold-change values vs p-values, called volcano plot.

*Usage*

```
volcanoplot(x, labels = "", p.value = "pval", mask = FALSE, show.cutoff
= TRUE, cex.text = 0.7, col.text = "blue", col.cutoff = "grey", xlim
= NULL, xlab = "Log2(Fold-Change)", ylab = "-Log10(P-Value)", pch =
'.' , ...)
```

**Arguments**

x	object of class <a href="#">AnalysisTreeSet</a> .
labels	optional transcript labels to be drawn at plotting points.
p.value	type of p-value, 'pval' for p-value, 'padj' for adjusted p-value, or 'pcha' for p-chance.
mask	logical, if TRUE draw only points for transcripts satisfying the univariate test.
show.cutoff	logical, if TRUE draw lines indicating cutoff.
cex.text	magnification to be used for optional labels.
col.text	color to be used for optional labels.
col.cutoff	color to be used for lines indicating cutoff, if show.cutoff=TRUE.
xlim	optional range for the plotted fold-change values.
xlab	label of x-axis.
ylab	label of y-axis.
pch	either an integer specifying a symbol or a single character to be used as the default in plotting points.
...	optional arguments to be passed to plot.

**Details**

Produces a volcano plot for slot `data` for an object of class [AnalysisTreeSet](#).

It is possible to label the points of the volcano plot, whereby the following `labels` parameters are valid:

fUnitName:	unit name (probeset ID).
fName:	gene name.
fSymbol:	gene symbol.
fChromosome:	chromosome.
fCytoBand:	cytoband.

**Author(s)**

Christian Stratowa

**Description**

Options for xps

**Usage**

```
xpsOptions (debug=FALSE)
```

**Arguments**

debug                   logical, if TRUE, print debug information.

**Details**

Currently only used to set debug to FALSE or TRUE.

**Value**

A global variable debug.xps can be set to TRUE.

**Author(s)**

Christian Stratowa

---

xps-package

*xps Package Overview*

---

**Description**

xps Package Overview

**Details**

Important data classes: [SchemeTreeSet](#), [DataTreeSet](#), [ExprTreeSet](#), [CallTreeSet](#), [FilterTreeSet](#), [AnalysisTreeSet](#). Full help on methods and associated functions is available from within class help pages.

Additional data classes: [ProjectInfo](#), [PreFilter](#), [UniFilter](#).

The package handles pre-processing, normalization, filtering and analysis of Affymetrix GeneChip expression arrays, including exon array systems (Exon 1.0 ST: core, extended, full probesets), gene array systems (Gene 1.0 ST) and plate array systems on computers with 1 GB RAM only. It imports Affymetrix .CDF, .CLF, .PGF and .CEL as well as Affymetrix annotation files, and computes e.g. RMA, MAS5, FARMS, DFW, MAS5-calls, DABG-calls, I/NI-calls. It is an R wrapper to XPS (eXpression Profiling System), which is based on ROOT, an object-oriented framework developed at CERN. Thus, the prior installation of ROOT is a prerequisite for the usage of this package, see the README file. However, no knowledge of ROOT is required. ROOT is licensed under LGPL and can be downloaded from <http://root.cern.ch>.

**Author(s)**

Christian Stratowa (cstrato@aon.at)

# Index

## \*Topic classes

- AnalysisTreeSet-class, 2
- CallTreeSet-class, 15
- DataTreeSet-class, 20
- ExprTreeSet-class, 37
- Filter-class, 42
- FilterTreeSet-class, 42
- PreFilter-class, 85
- ProcesSet-class, 92
- ProjectInfo-class, 93
- SchemeTreeSet-class, 120
- TreeSet-class, 123
- UniFilter-class, 126

## \*Topic device

- boxplot.dev, 11
- image.dev, 53
- mvaplot.dev, 79
- root.density, 106
- root.graph1D, 108
- root.graph2D, 109
- root.hist1D, 110
- root.hist2D, 111
- root.hist3D, 112
- root.image, 113
- root.mvaplot, 116
- root.profile, 117

## \*Topic manip

- bgcorrect, 9
- dabg.call, 17
- dfw, 22
- existsROOTFile, 25
- exonLevel, 26
- export, 29
- export.filter, 28
- export.root, 31
- express, 32
- farms, 38
- getChipName, 45
- getChipType, 45
- getDatatype, 46
- getNameType, 47
- getNumberTrees, 48
- getProbeInfo, 48

- getTreeNames, 50
- import.data, 55
- import.exon.scheme, 57
- import.expr.scheme, 59
- import.genome.scheme, 61
- ini.call, 63
- isROOTFile, 67
- mas4, 70
- mas5, 74
- mas5.call, 72
- metaProbesets, 78
- normalize, 81
- prefilter, 89
- PreFilter-constructor, 87
- ProjectInfo-constructor, 95
- rma, 100
- root.call, 104
- root.data, 105
- root.expr, 107
- root.merge.data, 115
- root.scheme, 119
- summarize, 122
- type2Exten, 125
- unifilter, 128
- UniFilter-constructor, 127
- validTreetype, 133
- xpsOptions, 136

## \*Topic methods

- addData-methods, 1
- attachBgrd-methods, 3
- attachCall-methods, 4
- attachExpr-methods, 5
- attachInten-methods, 7
- attachMask-methods, 8
- boxplot-methods, 12
- callFilter-methods, 13
- callplot-methods, 14
- cvFilter-methods, 16
- diffFilter-methods, 24
- exprs-methods, 35
- fcFilter-methods, 41
- gapFilter-methods, 44
- getTreeData-methods, 49

- highFilter-methods, 51
- hist-methods, 52
- image-methods, 54
- initialize-methods, 66
- intensity-methods, 66
- lowFilter-methods, 68
- madFilter-methods, 69
- mboxplot-methods, 77
- mvaplot-methods, 80
- pm-methods, 82
- pmplot-methods, 84
- presCall-methods, 90
- quantileFilter-methods, 97
- ratioFilter-methods, 98
- rawCELName-methods, 99
- root.browser-methods, 103
- uniTest-methods, 131
- unittestFilter-methods, 130
- validData-methods, 132
- varFilter-methods, 135
- volcanoplot-methods, 135
- \*Topic misc**
  - ROOT, 118
- \*Topic package**
  - xps-package, 137
- addData (*addData-methods*), 1
- addData, DataTreeSet-method (*DataTreeSet-class*), 20
- addData-methods, 1
- AnalysisTreeSet, 28, 43, 129, 136, 137
- AnalysisTreeSet (*AnalysisTreeSet-class*), 2
- AnalysisTreeSet-class, 2
- arrayInfo (*ProjectInfo-class*), 93
- arrayInfo, ProjectInfo-method (*ProjectInfo-class*), 93
- arrayInfo<- (*ProjectInfo-class*), 93
- arrayInfo<-, ProjectInfo, character-method (*ProjectInfo-class*), 93
- attachBgrd, 8
- attachBgrd (*attachBgrd-methods*), 3
- attachBgrd, DataTreeSet-method (*DataTreeSet-class*), 20
- attachBgrd-methods, 3
- attachCall, 6
- attachCall (*attachCall-methods*), 4
- attachCall, CallTreeSet-method (*CallTreeSet-class*), 15
- attachCall-methods, 4
- attachExpr, 5
- attachExpr (*attachExpr-methods*), 5
- attachExpr, ExprTreeSet-method (*ExprTreeSet-class*), 37
- attachExpr-methods, 5
- attachInten, 4, 11, 12, 52, 53, 55, 77, 84
- attachInten (*attachInten-methods*), 7
- attachInten, DataTreeSet-method (*DataTreeSet-class*), 20
- attachInten-methods, 7
- attachMask (*attachMask-methods*), 8
- attachMask, DataTreeSet-method (*DataTreeSet-class*), 20
- attachMask, SchemeTreeSet-method (*SchemeTreeSet-class*), 120
- attachMask-methods, 8
- attachPVal (*attachCall-methods*), 4
- attachPVal, CallTreeSet-method (*CallTreeSet-class*), 15
- attachPVal-methods (*attachCall-methods*), 4
- authorInfo (*ProjectInfo-class*), 93
- authorInfo, ProjectInfo-method (*ProjectInfo-class*), 93
- authorInfo<- (*ProjectInfo-class*), 93
- authorInfo<-, ProjectInfo, character-method (*ProjectInfo-class*), 93
- background (*DataTreeSet-class*), 20
- background, DataTreeSet-method (*DataTreeSet-class*), 20
- background<- (*DataTreeSet-class*), 20
- background<-, DataTreeSet, data.frame-method (*DataTreeSet-class*), 20
- barplot, 84
- bgcorrect, 3, 9, 21, 35, 53, 55
- bgcorrect.mas4, 53, 55
- bgtreeNames (*DataTreeSet-class*), 20
- bgtreeNames, DataTreeSet-method (*DataTreeSet-class*), 20
- biopsyInfo (*ProjectInfo-class*), 93
- biopsyInfo, ProjectInfo-method (*ProjectInfo-class*), 93
- biopsyInfo<- (*ProjectInfo-class*), 93
- biopsyInfo<-, ProjectInfo, character-method (*ProjectInfo-class*), 93
- boxplot, 12, 77, 84, 92
- boxplot (*boxplot-methods*), 12
- boxplot, ProceSet-method (*ProceSet-class*), 92

- boxplot-methods, 12
- boxplot.dev, 11, 12, 77, 84
- callFilter, 87, 89, 127, 129
- callFilter (*callFilter-methods*), 13
- callFilter, *PreFilter*-method (*PreFilter-class*), 85
- callFilter, *UniFilter*-method (*UniFilter-class*), 126
- callFilter-methods, 13
- callFilter<- (*callFilter-methods*), 13
- callFilter<-, *PreFilter*, character-method (*PreFilter-class*), 85
- callFilter<-, *UniFilter*, character-method (*UniFilter-class*), 126
- callplot (*callplot-methods*), 14
- callplot, *CallTreeSet*-method (*CallTreeSet-class*), 15
- callplot-methods, 14
- CallTreeSet*, 4, 5, 14, 19, 22, 29, 30, 38, 43, 65, 73, 89, 90, 92, 93, 103, 104, 124, 129, 137
- CallTreeSet* (*CallTreeSet-class*), 15
- callTreetset (*FilterTreeSet-class*), 42
- callTreetset, *FilterTreeSet*-method (*FilterTreeSet-class*), 42
- CallTreeSet*-class, 15
- celllineInfo (*ProjectInfo-class*), 93
- celllineInfo, *ProjectInfo*-method (*ProjectInfo-class*), 93
- celllineInfo<- (*ProjectInfo-class*), 93
- celllineInfo<- , *ProjectInfo*, character-method (*ProjectInfo-class*), 93
- chipMask (*SchemeTreeSet-class*), 120
- chipMask, *SchemeTreeSet*-method (*SchemeTreeSet-class*), 120
- chipMask<- (*SchemeTreeSet-class*), 120
- chipMask<- , *SchemeTreeSet*, data.frame-method (*SchemeTreeSet-class*), 120
- chipName (*SchemeTreeSet-class*), 120
- chipName, *ProcesSet*-method (*ProcesSet-class*), 92
- chipName, *SchemeTreeSet*-method (*SchemeTreeSet-class*), 120
- chipType (*SchemeTreeSet-class*), 120
- chipType, *ProcesSet*-method (*ProcesSet-class*), 92
- chipType, *SchemeTreeSet*-method (*SchemeTreeSet-class*), 120
- chipType<- (*SchemeTreeSet-class*), 120
- chipType<- , *SchemeTreeSet*, character-method (*SchemeTreeSet-class*), 120
- cvFilter, 87
- cvFilter (*cvFilter-methods*), 16
- cvFilter, *PreFilter*-method (*PreFilter-class*), 85
- cvFilter-methods, 16
- cvFilter<- (*cvFilter-methods*), 16
- cvFilter<- , *PreFilter*, numeric-method (*PreFilter-class*), 85
- dabg.call, 15, 17, 73
- data.frame, 50, 83, 133
- datasetInfo (*ProjectInfo-class*), 93
- datasetInfo, *ProjectInfo*-method (*ProjectInfo-class*), 93
- datasetInfo<- (*ProjectInfo-class*), 93
- datasetInfo<- , *ProjectInfo*, character-method (*ProjectInfo-class*), 93
- DataTreeSet*, 3, 4, 7–12, 16, 22, 29, 30, 34, 38, 39, 52–56, 66, 70, 74, 77, 82, 84, 92, 93, 95, 96, 100, 103, 105, 106, 108–112, 114–117, 124, 137
- DataTreeSet* (*DataTreeSet-class*), 20
- DataTreeSet*-class, 20
- debug.xps (*xpsOptions*), 136
- dfw, 22
- diffFilter, 87
- diffFilter (*diffFilter-methods*), 24
- diffFilter, *PreFilter*-method (*PreFilter-class*), 85
- diffFilter-methods, 24
- diffFilter<- (*diffFilter-methods*), 24
- diffFilter<- , *PreFilter*, numeric-method (*PreFilter-class*), 85
- existsROOTFile, 25, 68
- exonLevel, 26, 102
- export, 29, 32, 50

- export, *ProcesSet*-method  
(*ProcesSet*-class), 92
- export, *SchemeTreeSet*-method  
(*SchemeTreeSet*-class), 120
- export, *TreeSet*-method  
(*TreeSet*-class), 123
- export-methods, 29, 31, 32
- export-methods, 92, 121, 124
- export-methods (*export*), 29
- export.call (*export*), 29
- export.data (*export*), 29
- export.expr (*export*), 29
- export.filter, 28
- export.root, 31
- export.scheme (*export*), 29
- express, 10, 24, 32, 37, 41, 71, 75, 76, 82,  
102, 123
- exprs, 91
- exprs (*exprs*-methods), 35
- exprs, *ExprTreeSet*-method  
(*ExprTreeSet*-class), 37
- exprs-methods, 35
- exprs<- (*exprs*-methods), 35
- exprs<-, *ExprTreeSet*, *data.frame*-method  
(*ExprTreeSet*-class), 37
- ExprTreeSet*, 2, 6, 11, 12, 16, 22, 23, 29,  
30, 34, 36, 38, 40, 43, 52, 70, 71, 74,  
76, 77, 79–82, 84, 85, 87–89, 92, 93,  
100, 101, 103, 106–112, 116, 117,  
123, 124, 126–129, 137
- ExprTreeSet* (*ExprTreeSet*-class),  
37
- exprTreeset  
(*FilterTreeSet*-class), 42
- exprTreeset, *FilterTreeSet*-method  
(*FilterTreeSet*-class), 42
- ExprTreeSet*-class, 37
- exprType (*ExprTreeSet*-class), 37
- exprType, *ExprTreeSet*-method  
(*ExprTreeSet*-class), 37
- exprType<- (*ExprTreeSet*-class), 37
- exprType<-, *ExprTreeSet*, character-method  
(*ExprTreeSet*-class), 37
- farms, 38, 65
- fcFilter, 127
- fcFilter (*fcFilter*-methods), 41
- fcFilter, *UniFilter*-method  
(*UniFilter*-class), 126
- fcFilter-methods, 41
- fcFilter<- (*fcFilter*-methods), 41
- fcFilter<-, *UniFilter*, character-method  
(*UniFilter*-class), 126
- fileDir (*TreeSet*-class), 123
- fileDir, *TreeSet*-method  
(*TreeSet*-class), 123
- fileDir<- (*TreeSet*-class), 123
- fileDir<-, *TreeSet*, character-method  
(*TreeSet*-class), 123
- Filter, 85, 86, 88, 126, 127
- Filter-class, 42
- FilterTreeSet, 3, 28, 89, 129, 137
- FilterTreeSet  
(*FilterTreeSet*-class), 42
- filterTreeset  
(*AnalysisTreeSet*-class), 2
- filterTreeset, *AnalysisTreeSet*-method  
(*AnalysisTreeSet*-class), 2
- FilterTreeSet-class, 42
- gapFilter, 87
- gapFilter (*gapFilter*-methods), 44
- gapFilter, *PreFilter*-method  
(*PreFilter*-class), 85
- gapFilter-methods, 44
- gapFilter<- (*gapFilter*-methods),  
44
- gapFilter<-, *PreFilter*, numeric-method  
(*PreFilter*-class), 85
- getChipName, 45, 46, 47
- getChipType, 45, 45, 47
- getDatatype, 46, 125, 134
- getNameType, 45, 46, 47
- getNumberTrees, 48
- getProbeInfo, 48
- getTreeData  
(*getTreeData*-methods), 49
- getTreeData, *AnalysisTreeSet*-method  
(*AnalysisTreeSet*-class), 2
- getTreeData, *FilterTreeSet*-method  
(*FilterTreeSet*-class), 42
- getTreeData, *ProcesSet*-method  
(*ProcesSet*-class), 92
- getTreeData-methods, 49
- getTreeNames, 50, 104, 105, 107, 115
- highFilter, 87
- highFilter (*highFilter*-methods),  
51
- highFilter, *PreFilter*-method  
(*PreFilter*-class), 85
- highFilter-methods, 51
- highFilter<-  
(*highFilter*-methods), 51
- highFilter<-, *PreFilter*, character-method  
(*PreFilter*-class), 85

- hist (*hist-methods*), 52
- hist, *ProcesSet*-method  
(*ProcesSet-class*), 92
- hist-methods, 52
- hybridizInfo (*ProjectInfo-class*),  
93
- hybridizInfo, *ProjectInfo*-method  
(*ProjectInfo-class*), 93
- hybridizInfo<-  
(*ProjectInfo-class*), 93
- hybridizInfo<-, *ProjectInfo*, character-method  
(*ProjectInfo-class*), 93
  
- image, 54, 55, 114
- image (*image-methods*), 54
- image, *DataTreeSet*-method  
(*DataTreeSet-class*), 20
- image-methods, 54, 114
- image-methods, 54
- image.dev, 53, 55
- import.data, 2, 7, 20, 55, 67, 99, 105
- import.exon.scheme, 9, 57, 60, 62, 120
- import.expr.scheme, 9, 58, 59, 120
- import.genome.scheme, 60, 61, 120
- ini.call, 63
- initialize (*initialize-methods*),  
66
- initialize, *AnalysisTreeSet*-method  
(*initialize-methods*), 66
- initialize, *CallTreeSet*-method  
(*initialize-methods*), 66
- initialize, *DataTreeSet*-method  
(*initialize-methods*), 66
- initialize, *ExprTreeSet*-method  
(*initialize-methods*), 66
- initialize, *Filter*-method  
(*initialize-methods*), 66
- initialize, *FilterTreeSet*-method  
(*initialize-methods*), 66
- initialize, *PreFilter*-method  
(*initialize-methods*), 66
- initialize, *ProcesSet*-method  
(*initialize-methods*), 66
- initialize, *ProjectInfo*-method  
(*initialize-methods*), 66
- initialize, *SchemeTreeSet*-method  
(*initialize-methods*), 66
- initialize, *TreeSet*-method  
(*initialize-methods*), 66
- initialize, *UniFilter*-method  
(*initialize-methods*), 66
- initialize-methods, 66
- intensity (*intensity-methods*), 66
- intensity, *DataTreeSet*-method  
(*DataTreeSet-class*), 20
- intensity-methods, 66
- intensity<- (*intensity-methods*),  
66
- intensity<-, *DataTreeSet*, *data.frame*-method  
(*DataTreeSet-class*), 20
- isROOTFile, 26, 67
  
- lowFilter, 87
- lowFilter (*lowFilter-methods*), 68
- lowFilter, *PreFilter*-method  
(*PreFilter-class*), 85
- lowFilter-methods, 68
- lowFilter<- (*lowFilter-methods*),  
68
- lowFilter<-, *PreFilter*, character-method  
(*PreFilter-class*), 85
  
- madFilter, 87
- madFilter (*madFilter-methods*), 69
- madFilter, *PreFilter*-method  
(*PreFilter-class*), 85
- madFilter-methods, 69
- madFilter<- (*madFilter-methods*),  
69
- madFilter<-, *PreFilter*, numeric-method  
(*PreFilter-class*), 85
  
- mas4, 37, 70, 101
- mas5, 23, 27, 37, 40, 71, 74, 101
- mas5.call, 4, 15, 19, 64, 65, 72, 89, 129
- mboxplot (*mboxplot-methods*), 77
- mboxplot, *ProcesSet*-method  
(*ProcesSet-class*), 92
- mboxplot-methods, 77
- metaProbesets, 78
- mm, 133
- mm (*pm-methods*), 82
- mm, *DataTreeSet*-method  
(*DataTreeSet-class*), 20
- mm-methods (*pm-methods*), 82
- mvaplot, 80
- mvaplot (*mvaplot-methods*), 80
- mvaplot, *ExprTreeSet*-method  
(*ExprTreeSet-class*), 37
- mvaplot-methods, 80
- mvaplot.dev, 79, 80
  
- ncols (*SchemeTreeSet-class*), 120
- ncols, *DataTreeSet*-method  
(*DataTreeSet-class*), 20
- ncols, *SchemeTreeSet*-method  
(*SchemeTreeSet-class*), 120



- normalize, 35, 37, 81
- normType (*ExprTreeSet-class*), 37
- normType, *ExprTreeSet*-method  
(*ExprTreeSet-class*), 37
- normType<- (*ExprTreeSet-class*), 37
- normType<-, *ExprTreeSet*, character-method  
(*ExprTreeSet-class*), 37
- nrows (*SchemeTreeSet-class*), 120
- nrows, *DataTreeSet*-method  
(*DataTreeSet-class*), 20
- nrows, *SchemeTreeSet*-method  
(*SchemeTreeSet-class*), 120
- numberFilters (*Filter-class*), 42
- numberFilters, *Filter*-method  
(*Filter-class*), 42
- pm, 133
- pm (*pm-methods*), 82
- pm, *DataTreeSet*-method  
(*DataTreeSet-class*), 20
- pm-methods, 82
- pmplot, 15
- pmplot (*pmplot-methods*), 84
- pmplot, *DataTreeSet*-method  
(*DataTreeSet-class*), 20
- pmplot-methods, 84
- PreFilter, 14, 17, 25, 42, 43, 51, 69, 85,  
88–90, 98, 127, 128, 135, 137
- PreFilter  
(*PreFilter-constructor*), 87
- prefilter, 42, 89, 129
- PreFilter-class, 85
- PreFilter-constructor, 87
- presCall, 36
- presCall (*presCall-methods*), 90
- presCall, *CallTreeSet*-method  
(*CallTreeSet-class*), 15
- presCall-methods, 90
- presCall<- (*presCall-methods*), 90
- presCall<-, *CallTreeSet*, data.frame-method  
(*CallTreeSet-class*), 15
- primcellInfo (*ProjectInfo-class*),  
93
- primcellInfo, *ProjectInfo*-method  
(*ProjectInfo-class*), 93
- primcellInfo<-  
(*ProjectInfo-class*), 93
- primcellInfo<-, *ProjectInfo*, character-method  
(*ProjectInfo-class*), 93
- probeInfo (*SchemeTreeSet-class*),  
120
- probeInfo, *SchemeTreeSet*-method  
(*SchemeTreeSet-class*), 120
- ProcesSet, 2, 3, 15, 20, 37, 42, 43
- ProcesSet (*ProcesSet-class*), 92
- ProcesSet-class, 92
- ProjectInfo, 1, 55, 93, 97, 137
- ProjectInfo  
(*ProjectInfo-constructor*),  
95
- projectInfo (*ProjectInfo-class*),  
93
- projectInfo, *DataTreeSet*-method  
(*DataTreeSet-class*), 20
- projectInfo, *ProjectInfo*-method  
(*ProjectInfo-class*), 93
- ProjectInfo-class, 93
- ProjectInfo-constructor, 95
- projectInfo<-  
(*ProjectInfo-class*), 93
- projectInfo<-, *DataTreeSet*, *ProjectInfo*-method  
(*DataTreeSet-class*), 20
- projectInfo<-, *ProjectInfo*, character-method  
(*ProjectInfo-class*), 93
- pvalData, 36
- pvalData (*presCall-methods*), 90
- pvalData, *CallTreeSet*-method  
(*CallTreeSet-class*), 15
- pvalData-methods  
(*presCall-methods*), 90
- pvalData<- (*presCall-methods*), 90
- pvalData<-, *CallTreeSet*, data.frame-method  
(*CallTreeSet-class*), 15
- quantileFilter, 87
- quantileFilter  
(*quantileFilter-methods*),  
97
- quantileFilter, *PreFilter*-method  
(*PreFilter-class*), 85
- quantileFilter-methods, 97
- quantileFilter<-  
(*quantileFilter-methods*),  
97
- quantileFilter<-, *PreFilter*, numeric-method  
(*PreFilter-class*), 85
- ratioFilter, 87
- ratioFilter  
(*ratioFilter-methods*), 98
- ratioFilter, *PreFilter*-method  
(*PreFilter-class*), 85
- ratioFilter-methods, 98
- ratioFilter<-  
(*ratioFilter-methods*), 98

- ratioFilter<-, PreFilter, numeric-method (PreFilter-class), 85
- rawCELName (rawCELName-methods), 99
- rawCELName, DataTreeSet-method (DataTreeSet-class), 20
- rawCELName-methods, 99
- removeBgrd, 8
- removeBgrd (attachBgrd-methods), 3
- removeBgrd, DataTreeSet-method (DataTreeSet-class), 20
- removeBgrd-methods (attachBgrd-methods), 3
- removeCall, 6
- removeCall (attachCall-methods), 4
- removeCall, CallTreeSet-method (CallTreeSet-class), 15
- removeCall-methods (attachCall-methods), 4
- removeExpr, 5
- removeExpr (attachExpr-methods), 5
- removeExpr, ExprTreeSet-method (ExprTreeSet-class), 37
- removeExpr-methods (attachExpr-methods), 5
- removeInten, 4
- removeInten (attachInten-methods), 7
- removeInten, DataTreeSet-method (DataTreeSet-class), 20
- removeInten-methods (attachInten-methods), 7
- removeMask (attachMask-methods), 8
- removeMask, DataTreeSet-method (DataTreeSet-class), 20
- removeMask, SchemeTreeSet-method (SchemeTreeSet-class), 120
- removeMask-methods (attachMask-methods), 8
- removePVal (attachCall-methods), 4
- removePVal, CallTreeSet-method (CallTreeSet-class), 15
- removePVal-methods (attachCall-methods), 4
- rma, 23, 27, 37, 40, 100
- ROOT, 1-4, 6-8, 15, 16, 20, 21, 25, 31, 32, 37, 42, 43, 45-50, 56-62, 68, 75, 92, 93, 95, 96, 103-109, 111-116, 118, 118, 120, 121, 123, 124, 133
- root.browser, 119
- root.browser (root.browser-methods), 103
- root.browser, TreeSet-method (TreeSet-class), 123
- root.browser-methods, 103
- root.call, 104, 107
- root.data, 2, 20, 56, 104, 105, 105, 107, 115
- root.density, 52, 106
- root.expr, 104, 107
- root.graph1D, 108, 110, 116
- root.graph2D, 109, 109
- root.hist1D, 106, 110, 112, 113
- root.hist2D, 111, 111, 113
- root.hist3D, 111, 112, 112
- root.image, 113
- root.merge.data, 115
- root.mvaplot, 110, 116
- root.profile, 117
- root.scheme, 57, 58, 60-62, 119, 120
- rootFile (TreeSet-class), 123
- rootFile, TreeSet-method (TreeSet-class), 123
- rootFile<- (TreeSet-class), 123
- rootFile<-, TreeSet, character-method (TreeSet-class), 123
- sampleInfo (ProjectInfo-class), 93
- sampleInfo, ProjectInfo-method (ProjectInfo-class), 93
- sampleInfo<- (ProjectInfo-class), 93
- sampleInfo<-, ProjectInfo, character-method (ProjectInfo-class), 93
- schemeFile (ProcesSet-class), 92
- schemeFile, ProcesSet-method (ProcesSet-class), 92
- schemeFile<- (ProcesSet-class), 92
- schemeFile<-, ProcesSet, character-method (ProcesSet-class), 92
- schemeSet (ProcesSet-class), 92
- schemeSet, ProcesSet-method (ProcesSet-class), 92
- schemeSet<- (ProcesSet-class), 92
- schemeSet<-, ProcesSet, SchemeTreeSet-method (ProcesSet-class), 92
- SchemeTreeSet, 8, 19, 23, 29, 30, 40, 55, 57-62, 65, 73, 75, 92, 101, 103-105, 107, 115, 120, 124, 137
- SchemeTreeSet (SchemeTreeSet-class), 120
- SchemeTreeSet-class, 120
- se.exprs (ExprTreeSet-class), 37
- se.exprs, ExprTreeSet-method (ExprTreeSet-class), 37

- setName (*TreeSet-class*), 123
- setName, *TreeSet*-method
  - (*TreeSet-class*), 123
- setName<- (*TreeSet-class*), 123
- setName<- , *TreeSet*, character-method
  - (*TreeSet-class*), 123
- setType (*TreeSet-class*), 123
- setType, *TreeSet*-method
  - (*TreeSet-class*), 123
- setType<- (*TreeSet-class*), 123
- setType<- , *TreeSet*, character-method
  - (*TreeSet-class*), 123
- show, *ProjectInfo*-method
  - (*ProjectInfo-class*), 93
- sourceInfo (*ProjectInfo-class*), 93
- sourceInfo, *ProjectInfo*-method
  - (*ProjectInfo-class*), 93
- sourceInfo<- (*ProjectInfo-class*), 93
- sourceInfo<- , *ProjectInfo*, character-method
  - (*ProjectInfo-class*), 93
- summarize, 35, 37, 122
  
- tissueInfo (*ProjectInfo-class*), 93
- tissueInfo, *ProjectInfo*-method
  - (*ProjectInfo-class*), 93
- tissueInfo<- (*ProjectInfo-class*), 93
- tissueInfo<- , *ProjectInfo*, character-method
  - (*ProjectInfo-class*), 93
- treatmentInfo
  - (*ProjectInfo-class*), 93
- treatmentInfo, *ProjectInfo*-method
  - (*ProjectInfo-class*), 93
- treatmentInfo<-
  - (*ProjectInfo-class*), 93
- treatmentInfo<- , *ProjectInfo*, character-method
  - (*ProjectInfo-class*), 93
- treeNames (*TreeSet-class*), 123
- treeNames, *TreeSet*-method
  - (*TreeSet-class*), 123
- TreeSet*, 3, 15, 20, 37, 43, 92, 120, 121
- TreeSet* (*TreeSet-class*), 123
- TreeSet*-class, 123
- type2Exten, 47, 125, 134
  
- UniFilter*, 2, 14, 41, 42, 86, 88, 126–131, 137
- UniFilter*
  - (*UniFilter-constructor*), 127
- unifilter, 2, 90, 128
- UniFilter*-class, 126
  
- UniFilter*-constructor, 127
- uniTest, 127
- uniTest (*uniTest-methods*), 131
- uniTest, *UniFilter*-method
  - (*UniFilter-class*), 126
- uniTest-methods, 131
- uniTest<- (*uniTest-methods*), 131
- uniTest<- , *UniFilter*, character-method
  - (*UniFilter-class*), 126
- unitestFilter, 127
- unitestFilter
  - (*unitestFilter-methods*), 130
- unitestFilter, *UniFilter*-method
  - (*UniFilter-class*), 126
- unitestFilter-methods, 130
- unitestFilter<-
  - (*unitestFilter-methods*), 130
- unitestFilter<- , *UniFilter*, character-method
  - (*UniFilter-class*), 126
  
- validBgrd (*DataTreeSet-class*), 20
- validBgrd, *DataTreeSet*-method
  - (*DataTreeSet-class*), 20
- validCall (*CallTreeSet-class*), 15
- validCall, *CallTreeSet*-method
  - (*CallTreeSet-class*), 15
- validData, 11, 12, 52, 67, 77, 83, 84
- validData (*validData-methods*), 132
- validData, *AnalysisTreeSet*-method
  - (*AnalysisTreeSet-class*), 2
- validData, *DataTreeSet*-method
  - (*DataTreeSet-class*), 20
- validData, *FilterTreeSet*-method
  - (*FilterTreeSet-class*), 42
- validData, *ProcesSet*-method
  - (*ProcesSet-class*), 92
- validData-methods, 132
- validFilter
  - (*AnalysisTreeSet-class*), 2
- validFilter, *AnalysisTreeSet*-method
  - (*AnalysisTreeSet-class*), 2
- validTreetype, 30, 32, 46–50, 104, 107, 125, 133
- varFilter, 87
- varFilter (*varFilter-methods*), 135
- varFilter, *PreFilter*-method
  - (*PreFilter-class*), 85
- varFilter-methods, 135
- varFilter<- (*varFilter-methods*), 135

- varFilter<-, PreFilter, numeric-method  
(PreFilter-class), 85
- volcanoplot  
(volcanoplot-methods), 135
- volcanoplot, AnalysisTreeSet-method  
(AnalysisTreeSet-class), 2
- volcanoplot-methods, 135
  
- xps (xps-package), 137
- xps-package, 137
- xpsBgCorrect (bgcorrect), 9
- xpsBgCorrect, DataTreeSet-method  
(DataTreeSet-class), 20
- xpsBgCorrect-methods (bgcorrect),  
9
- xpsDABGCall (dabg.call), 17
- xpsDABGCall, DataTreeSet-method  
(DataTreeSet-class), 20
- xpsDABGCall-methods (dabg.call),  
17
- xpsINICall (ini.call), 63
- xpsINICall, DataTreeSet-method  
(DataTreeSet-class), 20
- xpsINICall-methods (ini.call), 63
- xpsMAS4, 71
- xpsMAS4 (mas4), 70
- xpsMAS4, DataTreeSet-method  
(DataTreeSet-class), 20
- xpsMAS4-methods (mas4), 70
- xpsMAS5 (mas5), 74
- xpsMAS5, DataTreeSet-method  
(DataTreeSet-class), 20
- xpsMAS5-methods (mas5), 74
- xpsMAS5Call (mas5.call), 72
- xpsMAS5Call, DataTreeSet-method  
(DataTreeSet-class), 20
- xpsMAS5Call-methods (mas5.call),  
72
- xpsNormalize (normalize), 81
- xpsNormalize, DataTreeSet-method  
(DataTreeSet-class), 20
- xpsNormalize, ExprTreeSet-method  
(ExprTreeSet-class), 37
- xpsNormalize-methods (normalize),  
81
- xpsOptions, 136
- xpsPreFilter (prefilter), 89
- xpsPreFilter, ExprTreeSet-method  
(ExprTreeSet-class), 37
- xpsPreFilter-methods (prefilter),  
89
- xpsPreprocess (express), 32
- xpsPreprocess, DataTreeSet-method  
(DataTreeSet-class), 20
- xpsPreprocess-methods (express),  
32
- xpsRMA (rma), 100
- xpsRMA, DataTreeSet-method  
(DataTreeSet-class), 20
- xpsRMA-methods (rma), 100
- xpsSummarize (summarize), 122
- xpsSummarize, DataTreeSet-method  
(DataTreeSet-class), 20
- xpsSummarize-methods (summarize),  
122
- xpsUniFilter (unifilter), 128
- xpsUniFilter, ExprTreeSet-method  
(ExprTreeSet-class), 37
- xpsUniFilter-methods (unifilter),  
128