Package 'xps'

April 15, 2017

Version 1.34.0

Title Processing and Analysis of Affymetrix Oligonucleotide Arrays including Exon Arrays, Whole Genome Arrays and Plate Arrays

Author Christian Stratowa, Vienna, Austria

Maintainer Christian Stratowa <cstrato@aon.at>

Depends R (>= 2.6.0), methods, utils

Suggests tools

Description The package handles pre-processing, normalization, filtering and analysis of Affymetrix GeneChip expression arrays, including exon arrays (Exon 1.0 ST: core, extended, full probesets), gene arrays (Gene 1.0 ST) and plate arrays on computers with 1 GB RAM only. It imports Affymetrix .CDF, .CLF, .PGF and .CEL as well as annotation files, and computes e.g. RMA, MAS5, FARMS, DFW, FIRMA, tRMA, 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, however, no knowledge of ROOT is required. ROOT is licensed under LGPL and can be downloaded from http://root.cern.ch.

License GPL (>= 2.0)

Collate utils.R TreeSetClasses.R methods.ProjectInfo.R methods.Filter.R methods.PreFilter.R methods.UniFilter.R methods.TreeSet.R methods.SchemeTreeSet.R methods.ProcesSet.R methods.DataTreeSet.R methods.ExprTreeSet.R methods.CallTreeSet.R methods.QualTreeSet.R methods.FilterTreeSet.R methods.AnalysisTreeSet.R Constructors.R import.R export.R root.files.R rma.R mas4.R mas5.R mas5.call.R dabg.call.R dfw.R farms.R ini.call.R firma.R trma.R bgcorrect.R normalize.R summarize.R express.R fitQC.R qualify.R AffyRNAdeg.R filter.R plot.R root.graphics.R xpsQAReport.R zzz.R

biocViews ExonArray, GeneExpression, Microarray, OneChannel, DataImport, Preprocessing, Transcription, DifferentialExpression

LazyLoad yes

SystemRequirements GNU make, root_v5.34.05 <http://root.cern.ch> - See README file for installation instructions.

NeedsCompilation yes

R topics documented:

`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>`

`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 `ProjectInfo`.
`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",sep="/"),
              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))
```

AffyRNAdeg *Functions to assess RNA Degradation.*

Description

Functions to detect possible RNA degradation.

Usage

```
AffyRNAdeg(xps.data, treename = "*", qualopt = "raw", log.it = TRUE)

summaryAffyRNAdeg(rna.deg, signif.digits=3)

plotAffyRNAdeg(rna.deg, transform = "shift.scale", col = NULL, summary = FALSE, add.legend = FALSE

xpsRNAdeg(object, ...)
```

Arguments

xps.data	object of class QualTreeSet.
treename	vector of tree names to export.
qualopt	option determining the data to which to apply qualification, one of 'raw', 'adjusted', 'normalized'.
log.it	logical, if TRUE, then probe data is log2 transformed.
rna.deg	list, output from AffyRNAdeg.
signif.digits	number of significant digits to show.
transform	transform data before plotting, one of "shift.scale", "shift.only", "none".
col	vector of colors for plot, length is number of samples.
summary	logical, if TRUE then the slope of summaryAffyRNAdeg will be plotted.
add.legend	logical or integer, if TRUE or larger than zero then a legend with the tree names will be drawn.
object	object of class QualTreeSet.
...	optional arguments to be passed to plotAffyRNAdeg.

Details

Since probes within a probeset are ordered directionally from the 5' end to the 3' end, it is possible to estimate the quality (degradation status) of the RNA.

Function AffyRNAdeg averages the probe intensities by location in the probeset, with the average taken over all probesets with identical number of probes.

Function summaryAffyRNAdeg produces a single summary statistic for each array.

Function plotAffyRNAdeg produces a side-by-side plot of the averaged intensities. Option transform = "none" shows the averaged intensities for each array while option "shift" staggers the plots for individual arrays vertically to make the display easier to read, and option "scale" normalizes the averaged intensities so that the standard deviation is equal to one.

Setting parameter add.legend = TRUE will add a legend containing all tree names to the plot, while setting e.g. add.legend = 6 will only show the first 6 tree names.

Value

AffyRNAdeg returns a list with following components:

N	number of probesets with identical number of probes
sample.names	names of samples, derived from affy batch object
mns	average intensity by probe position
ses	standard errors for probe position averages
slope	from linear regression of means.by.number
pvalue	from linear regression of means.by.number

Author(s)

Christian Stratowa, adapted from package affy

Examples

```
## Not run:
rnadeg <- xpsRNAdeg(rlm.all, treename="*", qualopt="raw")
plotAffyRNAdeg(rnadeg)

rnadeg <- AffyRNAdeg(rlm.all)
result <- summaryAffyRNAdeg(rnadeg)

## plot RNA degradation
plotAffyRNAdeg(rnadeg)

## plot slope of RNA degradation
plotAffyRNAdeg(rnadeg, summary = TRUE)

## End(Not run)
```

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 unitest 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 subdirectoy 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 subdirectoy setname.

treenames: Object of class "list" representing the names of the ROOT trees stored in subdirectoy 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 advisible 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

object	Object of class "CallTreeSet".
treenames	Object of class "list" representing the names of the ROOT 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 advisible 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.root",sep="/"))

## MAS5 detection call
call.mas5 <- mas5.call(data.test3,"tmp_Test3Call0",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()
```

---

attachData-methods *Attach/Remove Data*

Description

Attach/remove data from trees to/from ProcesSet.

Usage

```
attachData(object, treenames = character(0), varlist = character(0), outfile = "data.txt")
removeData(object)
```

Arguments

object	Object of class "ProcesSet".
treename	vector of tree names to export.
varlist	names of tree leaves to export
outfile	name of output file.

Details

attachData exports varlist from tree(s) with treenames and and saves the result as data.frame in slot data. Possible values of parameter varlist are described in export.

removeData removes data from slot data and replaces data.frame data with an empty data.frame of dim(0,0).

Value

A ProcesSet object.

Author(s)

Christian Stratowa

See Also

attachDataXY, attachInten

---

attachDataXY-methods *Attach/Remove (X,Y)-Coordinates*

Description

Attach/remove (x,y)-coordinates of raw CEL-files to/from DataTreeSet.

Usage

```
attachDataXY(object)
removeDataXY(object)
```

Arguments

object	Object of class "DataTreeSet".

Details

attachDataXY exports (x,y)-coordinates only from data tree of ROOT data file and and saves it as data.frame in slot data.

removeDataXY removes (x,y)-coordinates from slot data and replaces data.frame data with an empty data.frame of dim(0,0).

Value

A DataTreeSet object.

Author(s)

Christian Stratowa

See Also

attachInten, removeInten

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.root",sep="/"))

## attach (x,y)-coordinates
data.test3 <- attachDataXY(data.test3)

## get data.frame
xy <- treeData(data.test3)
head(xy)

## remove (x,y)-coordinates
data.test3 <- removeDataXY(data.test3)

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

---

attachExpr-methods *Attach/Remove Expression Measures*

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 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 advisible 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.root",sep="/"))

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

## 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 advisible 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.root",sep="/"))
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))
```

attachProbe-methods *Attach/Remove Probe Sequence and/or GC Content*

Description

Attach/remove probe sequence and/or GC content to/from SchemeTreeSet or to slot scheme of DataTreeSet.

Usage

```
attachProbe(object, varlist)
attachProbeContentGC(object)
attachProbeSequence(object)
removeProbe(object)
removeProbeContentGC(object)
removeProbeSequence(object)
```

Arguments

object	Object of class "SchemeTreeSet" or "DataTreeSet".
varlist	names of probe tree leaves to import to slot probe.

Details

Function attachProbe exports leaves from probe tree of ROOT scheme file and and saves the data as data.frame probe of slot scheme.

Following varlist parameters are valid:

fPosition:	probe interrogation position.
fSequence:	probe sequence.
fNumberGC:	number of G/C nucleotides in probe sequence.
fTm:	probe melting temperature dependent on G/C number.
fIsAntisense:	probe type (sense/antisense).

Function attachProbeContentGC saves fNumberGC in data.frame probe of SchemeTreeSet or in slot scheme of DataTreeSet.

Function attachProbeSequence saves fSequence in data.frame probe of SchemeTreeSet.

Function removeProbe removes probe data from SchemeTreeSet or from slot scheme of DataTreeSet and replaces data.frame probe with an empty data.frame of dim(0,0).

Value

A SchemeTreeSet object or DataTreeSet.

Note

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

Author(s)

Christian Stratowa

See Also

attachMask

Examples

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

scheme.test3 <- attachProbe(scheme.test3, varlist="fSequence:fNumberGC")
dim(chipProbe(scheme.test3))
head(chipProbe(scheme.test3))

scheme.test3 <- removeProbe(scheme.test3)
dim(chipProbe(scheme.test3))
```

---

attachUnitNames-methods

Attach/Remove Unit Names

Description

Attach/remove unit names, i.e. the Affymetrix probeset IDs to/from SchemeTreeSet or to slot scheme of DataTreeSet.

Usage

```
attachUnitNames(object, treetype = "idx")
removeUnitNames(object)
```

Arguments

object	Object of class "SchemeTreeSet" or "DataTreeSet".
treetype	the unit tree type, i.e. 'idx' or 'pbs'.

Details

attachUnitNames exports "UnitName" from unit tree of ROOT scheme file and and saves it as data.frame in slot unitname.

removeUnitNames removes unitname from slot unitname and replaces data.frame unitname with an empty data.frame of dim(0,0).

For treetype="idx" the internal "UNIT_ID" will be mapped to the Affymetrix probeset IDs of the expression arrays or to the transcript_cluster_ids of the exon arrays, respectively, as "UnitName".

For treetype="pbs" the internal "UNIT_ID" will be mapped to the Affymetrix probeset_ids of the exon arrays as "UnitName".

Value

A DataTreeSet object or SchemeTreeSet.

Note

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

Author(s)

Christian Stratowa

See Also

`attachMask`, `removeMask`

Examples

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

## attach unitname
scheme.test3 <- attachUnitNames(scheme.test3)

## get data.frame
unitnames <- unitNames(scheme.test3)
head(unitnames)

## remove unitname
scheme.test3 <- removeUnitNames(scheme.test3)

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

---

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, selec

bgcorrect.gc(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE, se

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, se

xpsBgCorrect(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.
update	logical. If TRUE the existing ROOT data file filename will be updated.
select	type of probes to select for background correction.
method	background method to use.
option	type of background correction to use.
exonlevel	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.root",sep="/"))

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

## 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.bg.mas4))))
}

## 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)
```

`borderplot-methods` *Plots of Border Elements*

Description

Produce box-and-whisker plot(s) of the positive and negative feature intensities.

Usage

```
borderplot(x,          type    = c("pos", "neg"),          qualopt = "raw",          transfo = log2,
```

Arguments

`x`	object of class `QualTreeSet`.
`type`	type of border elements to be used, one of "pos", "neg", or both.
`qualopt`	character string specifying whether to draw boxplots for "raw", "adjusted", or "normalized" border intensities.
`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.
`ylim`	the y limits of the plot.
`bmar`	optional `list` for bottom margin and axis label magnification `cex.axis`.
`las`	the style of axis labels.
`...`	optional arguments to be passed to `borderplot`.

Details

Creates a boxplot of the positive and negative feature intensities for an object of class QualTreeSet.

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

For bmar=NULL the default list bmar = list(b=6, cex.axis=1.0) will be used initially. However, both bottom margin b and axis label magnification cex.axis will be adjusted depending on the number of label characters and the number of samples.

Author(s)

Christian Stratowa

See Also

plotBorder, coiplot

Examples

```
## Not run:
 ## border intensities, created by e.g. rmaPLM()
 getTreeNames(rootFile(rlm.all), treetype="brd")
 borderplot(rlm.all)
 borderplot(rlm.all, type="pos")
 borderplot(rlm.all, type="neg")

## End(Not run)
```

---

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", bmar = NULL, ...)
```

Arguments

x	object of class DataTreeSet, ExprTreeSet or QualTreeSet.
which	type of probes to be used, for details see validData.
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.
bmar	optional list for bottom margin and axis label magnification cex.axis.
...	optional arguments to be passed to boxplot.

Details

Creates a boxplot for slot data for an object of class DataTreeSet, ExprTreeSet or QualTreeSet.

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.

For bmar=NULL the default list bmar = list(b=6, cex.axis=1.0) will be used initially. However, both bottom margin and axis label magnification will be adjusted depending on the number of label characters and the number of smaples.

Note

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

Alternatively it is possible to use the pre-calculated quantiles stored in the userinfo of the data trees by calling which="userinfo:varlist", where the varlist to call is described in method treeInfo.

Author(s)

Christian Stratowa

See Also

plotBoxplot, 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.root",sep="/"))

## need to attach scheme mask and probe intensities only if "userinfo" is not used
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)

## alternatively use the quantiles stored in userinfo of trees
if (interactive()) {
boxplot(data.test3, which="userinfo:fIntenQuant")
}

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:

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

Arguments

`x`	object of class `CallTreeSet`.
`beside`	logical. If `FALSE`, the columns of height are portrayed as stacked bars, and if `TRUE` the columns are portrayed as juxtaposed bars.
`names`	optional vector of sample names.
`col`	color for P/M/A bars
`legend`	legend for the plot, defaults to P/M/A.
`ylim`	the y limits of the plot.
`ylab`	a label for the y axis.
`las`	the style of axis labels.
`...`	optional arguments to be passed to `barplot`.

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

`plotCall`, `pmplot`

`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 subdirectoy 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 subdirectoy `setname`.

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

validPVal `signature(object = "CallTreeSet")`: extracts a subset of columns from data.frame `data`.

Author(s)

Christian Stratowa

See Also

related classes `DataTreeSet`, `ExprTreeSet`.

Examples

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

`coiplot-methods` *Center-Of-Intensity QC Plots*

Description

Produce Center-Of-Intensity plot(s) of the positive and negative feature intensities.

Usage

```
coiplot(x,      type   = c("pos", "neg"),      qualopt = "raw",      radius  = 0.5,      linecc
```

Arguments

`x`	object of class `QualTreeSet`.
`type`	type of border elements to be used, one of "pos", "neg", or both.
`qualopt`	character string specifying whether to draw boxplots for "raw", "adjusted", or "normalized" border intensities.
`radius`	determines the radius within which the COI for each array should be located.
`linecol`	the color of the ablines and the circle to be drawn.
`visible`	logical, if TRUE then arrays outside the circle with `radius` will be flagged by labeling the data point with the array name.
`...`	optional arguments to be passed to `coiplot`.

Details

Produces Center-Of-Intensity (COI) plot(s) of the positive and negative feature intensities for an object of class QualTreeSet. This plot is useful for detecting spatial biases in intensities on an array.

Mean intensities for the left, right, top and bottom border elements are calculated, separated into positive and negative controls, and the "center of intensity" is calculated on a relative scale [-1,1]. Arrays with a COI outside a range with radius are considered to be outliers. If visible = TRUE then outlier arrays will be flagged by labeling the data point(s) with the array name(s).

Value

The names ot the outlier arrays, otherwise NULL.

Author(s)

Christian Stratowa

See Also

plotCOI, borderplot

Examples

```
## Not run:
## border intensities, created by e.g. rmaPLM()
coiplot(rlm.all)
coiplot(rlm.all, type="pos")
coiplot(rlm.all, type="neg", radius=0.1)

## End(Not run)
```

---

corplot-methods *Array-Array Expression Level Correlation Plot*

Description

A heat map of the array-array Spearman rank correlation coefficients.

Usage

```
corplot(x, which = "UnitName", transfo = log2, method = "spearman",
```

Arguments

x	object of class ExprTreeSet.
which	type of probes to be used, for details see validData.
transfo	a valid function to transform the data, usually "log2", or "0".
method	a character string indicating which correlation coefficient is to be computed.
col	vector of colors for plot, length is number of samples.
names	optional vector of sample names.
sort	logical, if TRUE the correlation matrix will be sorted decreasingly.

reverse	logical, if TRUE the correlation matrix will be replaced by 1 - cor().
bmar	optional list for bottom margin and axis label magnification cex.axis.
add.legend	logical, if TRUE then a color bar will be drawn.
...	optional arguments to be passed to plot.

Details

Produces a heat map of the array-array Spearman rank correlation coefficients 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 corplot.

For bmar=NULL the default list bmar = list(b=6, cex.axis=1.0) will be used initially. However, both bottom margin and axis label magnification will be adjusted depending on the number of label characters and the number of samples.

Note

Setting reverse = FALSE displays the correlation heat map as in package affyQCReport.

Author(s)

Christian Stratowa

See Also

plotCorr, madplot

---

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 = TRUE, verbose = TRUE)

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).
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 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:

transcript:	expression levels are computed for transcript clusters, i.e. probe sets containing the same 'transcript_cluste
exon:	expression levels are computed for exon clusters, i.e. probe sets containing the same 'exon_id', where each
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+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'.

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 pa-

rameters.

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.root",sep="/"))

## 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 subdirectoy 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 subdirectoy setname.

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

attachDataXY signature(object = "DataTreeSet"): exports (x,y)-coordinates from ROOT data file and and saves as data.frame data.

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 and saves as data.frame `mask` of slot `scheme`.

attachProbeContentGC `signature(object = "DataTreeSet")`: exports probe tree from ROOT scheme file and and saves `fNumberGC` as data.frame `probe`.

attachUnitNames `signature(object = "DataTreeSet")`: exports unit tree from ROOT scheme file and and saves as data.frame `unitname` 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`.

indexUnits `signature(object = "DataTreeSet")`: extracts (x,y)-coordinates and corresponding indices for all or selected unitIDs.

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

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

intensity2GCplot `signature(x = "DataTreeSet")`: creates a boxplot of probe intensities stratified by GC content.

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

mmindex `signature(object = "DataTreeSet")`: extracts (x,y)-coordinates and corresponding MM indices for all or selected unitIDs.

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.

pmindex `signature(object = "DataTreeSet")`: extracts (x,y)-coordinates and corresponding PM indices for all or selected unitIDs.

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

probesetID2unitID `signature(object = "DataTreeSet")`: extracts all or selected probesetIDs from data.frame `unitname` of slot `scheme` with UnitName, i.e. probeset ID, as (row)names.

probesetplot `signature(x = "DataTreeSet")`: creates a line plot of probe intensities for a probeset.

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).

removeDataXY `signature(object = "DataTreeSet")`: replaces data.frame `data` 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).

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

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

symbol2unitID `signature(object = "DataTreeSet")`: extracts internal UNIT_ID(s) for one or more gene symbols.

transcriptID2unitID `signature(object = "DataTreeSet")`: extracts all or selected transcriptIDs from data.frame `unitname` of slot `scheme` with UnitName, i.e. transcript ID, as (row)names.

unitID2probesetID `signature(object = "DataTreeSet")`: extracts all or selected unitIDs from data.frame `unitname` of slot `scheme` with UNIT_ID as (row)names.

symbol2unitID `signature(object = "DataTreeSet")`: extracts gene symbols for one or more internal UNIT_ID(s).

unitID2transcriptID `signature(object = "DataTreeSet")`: extracts all or selected unitIDs from data.frame `unitname` of slot `scheme` with UNIT_ID as (row)names.

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.

xpsFIRMA `signature(object = "DataTreeSet")`: computes FIRMA expression level and splice score.

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.

xpsQualify `signature(object = "DataTreeSet")`: applies quality control methods.

xpsQualityControl `signature(object = "DataTreeSet")`: applies quality control 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 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.
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_cluste
exon:	expression levels are computed for exon clusters, i.e. probe sets containing the same 'exon_id', where each
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.root",sep="/"))

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 >= cutoff)`

Usage

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

Arguments

`object`	object of class `PreFilter`.
`value`	numeric vector `c(cutoff, trim, epsilon)`.

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) >= 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 exlude 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` *Conversion of Parameter exonlevel to Integer*

Description

Conversion of parameter `exonlevel` to an integer vector.

Usage

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

Arguments

`exonlevel`	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
`chiptype`	chip tpye, one of 'GeneChip', 'GenomeChip', 'ExonChip'.
`as.sum`	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:

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

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

`core`:	`(=8192+1024)` probesets with category 'unique', 'similar' and 'mixed'.
`metacore`:	`(=8192)` probesets with category 'unique' only.
`affx`:	`(=60)` standard AFFX controls.
`all`:	`(=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 seperate 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:

`control->bgp->genomic:`	`(=32768)` genomic background probes.
`control->bgp->antigenomic:`	`(=65536)` antigenomic background probes.
`normgene->intron:`	`(=131072)` intronic controls.
`normgene->exon:`	`(=262144)` exronic controls.
`rescue->FLmRNA->unmapped:`	`(=524288)` unmapped mRNAs.

For whole genome arrays it is possible (but not recommended) to use all probesets by using `exonlevel=c(992316,99231`
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 *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 = character(0), sep = "\

export.data(xps.data, treename = "*", treetype = "cel", varlist = "*", outfile = character(0), sep

export.expr(xps.expr, treename = "*", treetype = character(0), varlist = "*", outfile = character(

export.call(xps.call, treename = "*", treetype = character(0), varlist = "*", outfile = character(

export(object, ...)
```

Arguments

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

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:

fInten:	intensities from e.g. tree.cel.
fStdev:	standard deviation from e.g. tree.cel.
fNPixels:	number of pixels from e.g. tree.cel.
fBg:	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:

fUnitName:	unit name (probeset ID).
fTranscriptID:	transcript_id (probeset ID).
fName:	gene name.
fSymbol:	gene symbol.

fAccession:	mRNA accession such as Refseq ID.
fEntrezID:	entrez ID.
fChromosome:	chromosome.
fStart:	start position.
fStop:	stop position.
fStrand:	strand on chromosome.
fCytoBand:	cytoband.

Following varlist parameters are valid for ExprTreeSet:

fLevel:	expression level.
fStdev:	standard deviation.
fNPairs:	number of pairs.

Following varlist parameters are valid for CallTreeSet:

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

Following varlist parameters are valid for QualTreeSet:

fLevel:	expression level.
fStderr:	standard error.
fNUSE:	normalized unscaled standard error.
fRLE:	relative log expression 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.root",sep="/"))

## 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=TRUE,verbose=FALSE)
head(ann)
data <- export.data(data.test3, outfile="Test3_cel.txt",as.dataframe=TRUE,verbose=FALSE)
head(data)
```

`export.filter` *Export filter data as text files*

Description

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

Usage

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

Arguments

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

`fUnitName`:	unit name (probeset ID).
`fFlag`:	mask.

For class `AnalysisTreeSet` `varlist` can contain annotation parameters and parameters of the resulting data.
Following `varlist` annotation parameters are valid:

`fUnitName`:	unit name (probeset ID).
`fTranscriptID`:	transcript_id (probeset ID).
`fName`:	gene name.
`fSymbol`:	gene symbol.
`fAccession`:	mRNA accession such as Refseq ID.
`fEntrezID`:	entrez ID.
`fChromosome`:	chromosome.

fStart:	start position.
fStop:	stop position.
fStrand:	strand on chromosome.
fCytoBand:	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.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 = character(0), treename =
```

Arguments

datafile	name of ROOT data file including full path
schemefile	name of ROOT scheme file including full path
treeset	name of subdirectory in ROOT file where trees are stored
treename	name of ROOT tree to export.

treetype	type of tree(s) to export, see validTreetype.
varlist	names of tree leaves to export.
outfile	name of output file.
sep	column separator
as.dataframe	if TRUE a data.frame will be returned.
verbose	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", as.dataframe = TRUE,
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(0),
  # misc.
    exonlevel  = "",
    xps.scheme = NULL,
    add.data   = TRUE,
    bufsize    = 32000,
    verbose    = TRUE)

xpsPreprocess(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.
`update`	logical. If `TRUE` the existing ROOT data file `filename` will be updated.
`bgcorrect.method`	background method to use.
`bgcorrect.select`	type of probes to select for background correction.
`bgcorrect.option`	type of background correction to use.
`bgcorrect.params`	vector of parameters for background method.
`normalize.method`	normalization method to use.
`normalize.select`	type of probes to select for normalization.
`normalize.option`	normalization option.
`normalize.logbase`	logarithm base as character, one of '0', 'log', 'log2', 'log10'.

`normalize.params`
: vector of parameters for normalization method.

`summarize.method`
: summarization method to use.

`summarize.select`
: type of probes to select for summarization.

`summarize.option`
: option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon arrays only.

`summarize.logbase`
: logarithm base as character, one of '0', 'log', 'log2', 'log10'.

`summarize.params`
: vector of parameters for summarization method.

`reference.index`
: index of reference tree to use, or 0.

`reference.method`
: for `refindex=0`, either trimmed mean or median of trees.

`reference.params`
: vector of parameters for reference method.

`exonlevel`
: exon annotation level determining which probes should be used for summarization; exon/genome arrays only.

`xps.scheme`
: optional alternative `SchemeSet`.

`add.data`
: logical. If `TRUE` expression data will be included as slot `data`.

`bufsize`
: integer which sets the buffer size of the tree branch baskets (default is 32000).

`verbose`
: logical, if `TRUE` print status information.

`object`
: object of class `DataTreeSet`.

`...`
: 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.

Please have a look at vignette "xpsPreprocess.pdf" for details on how to use function `express`.

`xpsPreprocess` is the `DataTreeSet` 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.root",sep="/"))

## 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:epanechnikov",bgcorrect.p
          normalize.method="quantile",normalize.select="pmonly",normalize.option="transcript:together:none"
          summarize.method="medianpolish",summarize.select="pmonly",summarize.option="transcript",summarize
            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="pmonly:epanechnikov",bgco
          normalize.method="quantile",normalize.select="pmonly",normalize.option="transcript:together:none"
          summarize.method="medianpolish",summarize.select="pmonly",summarize.option="transcript",summarize
            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

`object`	object of class `ExprTreeSet`.

treenames	character vector containing optional tree names to be used as subset.
value	data.frame 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.root",sep="/"))

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

## 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 exression 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 subdirectoy 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 subdirectoy setname.

treenames: Object of class "list" representing the names of the ROOT trees stored in subdirectoy 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 and saves as data.frame data.

corplot signature(x = "ExprTreeSet"): creates a correlation heat map.

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.

madplot `signature(x = "ExprTreeSet")`: creates a false color display of between arrays distances.

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

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

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

nuseplot `signature(x = "ExprTreeSet")`: creates a NUSE-plot.

pcaplot `signature(x = "ExprTreeSet")`: plots first two principal components of PCA.

rleplot `signature(x = "ExprTreeSet")`: creates a RLE-plot.

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.

validExpr `signature(object = "ExprTreeSet")`: extracts a subset of columns from data.frame `data`.

validSE `signature(object = "ExprTreeSet")`: extracts data columns from data.frame `se.exprs`.

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

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

xpsUniFilter `signature(object = "ExprTreeSet")`: applies unifiltering methods.

Author(s)

Christian Stratowa

See Also

related classes `DataTreeSet`, `CallTreeSet`, `QualTreeSet`.

Examples

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

`extenPart` *Get Extension of Tree Names*

Description

Get the extension(s) of (tree) names.

Usage

```
extenPart(names, as.unique=TRUE)
```

Arguments

names	vector of names.
as.unique	if TRUE return only unique extensions.

Details

Extracts the extension part of names, e.g.of tree names of treename.treetype stored in a ROOT file.

Value

A vector of (unique) extensions.

Author(s)

Christian Stratowa

See Also

namePart

Examples

```
names <- c("TestA1.int", "TestA2.int")
extenPart(names)
extenPart(names, as.unique=FALSE)
```

---

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

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.
normalize	logical. If TRUE normalize data using quantile normalization.
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.
weighted	logical, used only with version="1.3.1". Default is TRUE.
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 expression data will be included as slot data.
verbose	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_cluste
exon:	expression levels are computed for exon clusters, i.e. probe sets containing the same 'exon_id', where each
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 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.root",sep="/"))

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

`object`	object of class `UniFilter`.
`value`	numeric vector `c(cutoff, direction)`

Details

The method `fcFilter` initializes the following parameters:

`cutoff`:	the cutoff level for the filter.
`direction`:	`direction="both"` (default): select up and downregulated genes. `direction="up"`: select upregulated genes only. `direction="down"`: 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 subdirectoy 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 subdirectoy `setname`.

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

firma *Finding Isoforms using Robust Multichip Analysis*

Description

This function converts a DataTreeSet for exon arrays into an ExprTreeSet using the Finding Isoforms using Robust Multichip Analysis (FIRMA).

Usage

```
firma(xps.data,
      filename   = character(0),
      filedir    = getwd(),
      tmpdir     = "",
      background = "antigenomic",
      normalize  = TRUE,
      option     = "probeset",
      exonlevel  = "metacore",
      method     = "mdp",
      params     = list(16384, 0.0, 1.0, 10, 0.01, 1.0),
      xps.scheme = NULL,
      add.data   = TRUE,
      verbose    = TRUE)

xpsFIRMA(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.
background	probes used to compute background, one of 'genomic', 'antigenomic'
normalize	logical. If TRUE normalize data using quantile normalization.
option	option determining the grouping of probes for summarization, one of 'exon', 'probeset'.
exonlevel	exon annotation level determining which probes should be used for summarization.
method	method to be used for summarization, currently 'mdp'.
params	list of (default) parameters for rma.
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.
object	object of class DataTreeSet.
...	the arguments described above.

Details

This function computes FIRMA (Finding Isoforms using Robust Multichip Analysis) for detecting differential alternative splicing for exon arrays, as described in Purdom et al.

Following `options` are valid for exon arrays:

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

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 (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".

Method `xpsFIRMA` is the `DataTreeSet` method called by function `firma`, containing the same parameters.

Value

An `ExprTreeSet`

Note

In contrary to other implementations of (FI)RMA the expression measure of FIRMA is given in linear scale, analogously to the expression measures computed with `mas5` and `mas4`.

Please note that the current implementation of FIRMA is based on median-polish only, see: `http://www.aroma-project.org/node/81`

Please note that the default settings of `params` gives results which are identical to the results obtained with APT (Affymetrix Power Tools) and with package `affy_1.14.2` or earlier. If you want to obtain results which are identical to the results obtained with `affy_1.16.0` or later then you need to set `params = list(16384, 0.0, 0.4, 10, 0.01, 1.0)`.

By setting parameter `background="none"` it is possible to skip background correction .

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 seperate 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

Purdom, E., Simpson K.M., Robinson M.D., Conboy J.G., Lapuk A.V. and Speed, T.P. (2008), FIRMA: a method for detection of alternative splicing from exon array data. Bioinformatics 24(15):1707-1714

Examples

```
## Not run:
## load ROOT scheme file
scmdir <- "/Volumes/GigaDrive/CRAN/Workspaces/Schemes"
scheme.exon <- root.scheme(paste(scmdir,"Scheme_HuEx10stv2r2_na27.root",sep="/"))

## load subset of ROOT data file
datdir <- "/Volumes/GigaDrive/CRAN/Workspaces/ROOTData"
subnames <- c("HeartA","HeartB","HeartC", "MuscleA","MuscleB","MuscleC")
sub.exon <- root.data(scheme.exon, rootFile(data.exon), celnames=subnames)

## firma
outdir <- getwd()
sub.firma.ps <- firma(sub.exon,"HeartMuscleFIRMAcorePS",filedir=outdir,tmpdir="",background="antigenomic",
                      normalize=TRUE,option="probeset",exonlevel="core")

## get transcript expression levels for all transcripts or transcript=2429277
expr.firma <- firma.expr(sub.firma.ps, probeset=NULL, option="transcript")
expr.firma <- firma.expr(sub.firma.ps, probeset=2429277, option="transcript")

## get probeset expression levels for all probeset or probeset=2429278 or transcript=2429277
expr.firma <- firma.expr(sub.firma.ps, probeset=NULL, option="probeset")
expr.firma <- firma.expr(sub.firma.ps, probeset=2429278, option="probeset")
expr.firma <- firma.expr(sub.firma.ps, probeset=2429277, option="probeset")

## get probeset splice scores for all probeset or probeset=2429278 or transcript=2429277
score.firma <- firma.score(sub.firma.ps, probeset=NULL, option="probeset")
score.firma <- firma.score(sub.firma.ps, probeset=2429278, option="probeset")
score.firma <- firma.score(sub.firma.ps, probeset=2429277, option="probeset")

## different plots
boxplot(sub.firma.ps, which="UnitName:LEVEL_PS")
boxplot(sub.firma.ps, which="UnitName:LEVEL_TS")

hist(sub.firma.ps, which="UnitName:LEVEL_PS")
hist(sub.firma.ps, which="UnitName:LEVEL_TS")
```

```
rleplot(sub.firma.ps, which="UnitName:LEVEL_PS")
rleplot(sub.firma.ps, which="UnitName:LEVEL_TS")

nuseplot(sub.firma.ps, which="UnitName:STDEV_PS")
nuseplot(sub.firma.ps, which="UnitName:STDEV_TS")

## End(Not run)
```

`firma.expr` *Get Expression Levels from FIRMA*

Description

Extracts FIRMA expression levels from data.frame `data`.

Usage

```
firma.expr(xps.data,
           probeset = NULL,
           option   = "probeset")
```

Arguments

`xps.data`	object of class `ExprTreeSet`.
`probeset`	transcriptID or probesetID or NULL.
`option`	option determining the `probeset` type for which to extract expression levels, one of 'transcript', 'probeset', 'exon'.

Details

Function `firma.expr` returns the expression levels from slot `data` for a given `probeset`, or for all probesets or transcripts in case of `probeset=NULL`. Row names will be the Affymetrix transcriptIDs, probesetIDs or exonIDs, respectively, dependent on the selected `option`.

Value

A `data.frame`.

Note

For `option="probeset"` parameter `probeset` should usually be the transcriptID in order to get the expression levels for all probesetIDs of the corresponding transcriptID.

Author(s)

Christian Stratowa

See Also

`firma`

Examples

```
## Not run:
## get transcript expression levels for all transcripts or for transcript=2429277
expr.firma <- firma.expr(sub.firma.ps, probeset=NULL, option="transcript")
expr.firma <- firma.expr(sub.firma.ps, probeset=2429277, option="transcript")

## get probeset expression levels for all probeset or for probeset=2429278
expr.firma <- firma.expr(sub.firma.ps, probeset=NULL, option="probeset")
expr.firma <- firma.expr(sub.firma.ps, probeset=2429278, option="probeset")

## get probeset expression levels for all probesets corresponding to transcript=2429277
expr.firma <- firma.expr(sub.firma.ps, probeset=2429277, option="probeset")

## End(Not run)
```

`firma.score` *Get Splice Score from FIRMA*

Description

Extracts the FIRMA splice score from data.frame `data`.

Usage

```
firma.score(xps.data,
            probeset = NULL,
            option   = "probeset")
```

Arguments

`xps.data`	object of class `ExprTreeSet`.
`probeset`	probesetID or NULL.
`option`	option determining the `probeset` type for which to extract expression levels, one of 'probeset', 'exon'.

Details

Function `firma.score` returns the FIRMA splice score described in Purdom et al. from slot `data` for a given `probeset`, or for all probesets in case of `probeset=NULL`. Row names will be the Affymetrix probesetIDs or exonIDs, respectively, dependent on the selected `option`.

Value

A `data.frame`.

Note

For `option="probeset"` parameter `probeset` should usually be the transcriptID in order to get the splice scores for all probesetIDs of the corresponding transcriptID.

Author(s)

Christian Stratowa

References

Purdom, E., Simpson K.M., Robinson M.D., Conboy J.G., Lapuk A.V. and Speed, T.P. (2008), FIRMA: a method for detection of alternative splicing from exon array data. Bioinformatics 24(15):1707-1714

See Also

firma

Examples

```
## Not run:
## get probeset splice scores for all probeset or for probeset=2429278
score.firma <- firma.score(sub.firma.ps, probeset=NULL, option="probeset")
score.firma <- firma.score(sub.firma.ps, probeset=2429278, option="probeset")

## get probeset splice scores for all probesets corresponding to transcript=2429277
score.firma <- firma.score(sub.firma.ps, probeset=2429277, option="probeset")

## End(Not run)
```

---

fitQC *Functions for fitting probe-level models*

Description

This function allows to combine different algorithms to compute background correction, normalization and fit a multichip model for summarization.

Usage

```
fitQC(xps.data,
      filename = character(),
      filedir = getwd(),
      tmpdir = "",
      update = FALSE,
    # background correction
      bgcorrect.method  = "rma",
      bgcorrect.select  = "none",
      bgcorrect.option  = "pmonly:epanechnikov",
      bgcorrect.params  = c(16384),
    # normalization
      normalize.method  = "quantile",
      normalize.select  = "pmonly",
      normalize.option  = "transcript:together:none",
      normalize.logbase = "0",
      normalize.params  = c(0.0),
    # quality control
      qualify.method    = "rlm",
      qualify.select    = "pmonly",
      qualify.qualopt   = "all",
```

```
      qualify.option    = "transcript",
      qualify.estimator = "huber",
      qualify.logbase   = "log2",
      qualify.params    = list(10, 0.01, 1.0),
    # reference values
      reference.index   = 0,
      reference.method  = "mean",
      reference.params  = list(0.0),
    # misc.
      exonlevel  = "",
      xps.scheme = NULL,
      add.data   = FALSE,
      bufsize    = 32000,
      verbose    = TRUE)

xpsQualityControl(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.
update	logical. If TRUE the existing ROOT data file filename will be updated.
bgcorrect.method	background method to use.
bgcorrect.select	type of probes to select for background correction.
bgcorrect.option	type of background correction to use.
bgcorrect.params	vector of parameters for background method.
normalize.method	normalization method to use.
normalize.select	type of probes to select for normalization.
normalize.option	normalization option.
normalize.logbase	logarithm base as character, one of '0', 'log', 'log2', 'log10'.
normalize.params	vector of parameters for normalization method.
qualify.method	qualification method to use, currently rlm.
qualify.select	type of probes to select for qualification.
qualify.qualopt	option determining the data to which to apply qualification, one of 'raw', 'adjusted', 'normalized', 'all'.
qualify.option	option determining the grouping of probes for qualification, one of 'transcript', 'exon', 'probeset'; exon arrays only.

qualify.estimator
: option determining the M-estimator to use, one of 'huber', 'fair', 'cauchy', 'geman mcclure', 'welsch', 'tukey', 'andrew'.

qualify.logbase
: logarithm base as character, one of '0', 'log', 'log2', 'log10'.

qualify.params
: vector of parameters for qualification method.

reference.index
: index of reference tree to use, or 0.

reference.method
: for refindex=0, either trimmed mean or median of trees.

reference.params
: vector of parameters for reference method.

exonlevel
: exon annotation level determining which probes should be used for summarization; exon/genome arrays only.

xps.scheme
: optional alternative SchemeSet.

add.data
: logical. If TRUE expression data will be included as slot data.

bufsize
: integer which sets the buffer size of the tree branch baskets (default is 32000).

verbose
: logical, if TRUE print status information.

object
: object of class DataTreeSet.

...
: the arguments described above.

Details

This function allows to combine different algorithms to compute background correction, normalization and fit a multichip model for summarization.

xpsQualityControl is the DataTreeSet method called by function fitQC, containing the same parameters.

Value

An object of type QualTreeSet.

Author(s)

Christian Stratowa

See Also

fitRLM, qualify, express

Examples

```
## Not run:
## 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.root",sep="/"))

## qualification - rlm
rlm.all <- fitQC(data.test3, "tmp_Test3RLMall", filedir=getwd(), tmpdir="",
             qualify.method="rlm", qualify.qualopt="all", qualify.option="transcript", add.data=FALSE)
```

```
## get expression data.frame
expr.rlm.all <- validData(rlm.all)

## get borders
brd.rlm.all <- borders(rlm.all)

## get residuals
res.rlm.all <- residuals(rlm.all)

## get weights
w.rlm.all <- weights(rlm.all)

## plot expression levels
if (interactive()) {
coiplot(rlm.all)
borderplot(rlm.all)
nuseplot(rlm.all)
rleplot(rlm.all)
image(rlm.all, type="resids")
}

## End(Not run)
```

`fitRLM` *Functions for fitting RMA as probe-level model*

Description

Convert Affymetrix probe level data to expression levels by fitting RMA as multichip model.

Usage

```
fitRLM(xps.data,
       filename   = character(),
       filedir    = getwd(),
       tmpdir     = "",
       background = "pmonly",
       normalize  = TRUE,
       qualopt    = "all",
       option     = "transcript",
       exonlevel  = "",
       params     = list(16384, 0.0, 1.0, 10, 0.01, 1),
       xps.scheme = NULL,
       add.data   = FALSE,
       bufsize    = 32000,
       verbose    = TRUE)

rmaPLM(xps.data,
       filename   = character(),
       filedir    = getwd(),
       tmpdir     = "",
       background = "pmonly",
       normalize  = TRUE,
```

```
qualopt    = "all",
option     = "transcript",
exonlevel  = "",
params     = list(16384, 0.0, 1.0, 10, 0.01, 1),
xps.scheme = NULL,
add.data   = FALSE,
bufsize    = 32000,
verbose    = TRUE)
```

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.
background	probes used to compute background, one of 'pmonly', 'mmonly', 'both'; for genome/exon arrays one of 'genomic', 'antigenomic'
normalize	logical. If TRUE normalize data using quantile normalization.
qualopt	option determining the data to which to apply qualification, one of 'raw', 'adjusted', 'normalized', 'all'.
option	option determining the grouping of probes for qualification, one of 'transcript', 'exon', 'probeset'; exon arrays only.
exonlevel	exon annotation level determining which probes should be used for summarization; exon/genome arrays only.
params	list of (default) parameters for rma.
xps.scheme	optional alternative SchemeSet.
add.data	logical. If TRUE expression data will be included as slot data.
bufsize	integer which sets the buffer size of the tree branch baskets (default is 32000).
verbose	logical, if TRUE print status information.

Details

Convert Affymetrix probe level data to expression levels by fitting RMA as multichip model.

Value

An object of type QualTreeSet.

Author(s)

Christian Stratowa

See Also

fitQC, qualify, express

Examples

```
## Not run:
## 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.root",sep="/"))

## qualification - rlm
rlm.all <- rmaPLM(data.test3, "tmp_Test3RLMall", filedir=getwd(), tmpdir="", qualopt="all", option="transcri

## get borders
brd.rlm.all <- borders(rlm.all)

## get residuals
res.rlm.all <- residuals(rlm.all)

## get weights
w.rlm.all <- weights(rlm.all)

## plot expression levels
if (interactive()) {
coiplot(rlm.all)
borderplot(rlm.all)
nuseplot(rlm.all)
rleplot(rlm.all)
image(rlm.all, type="resids")
}

## End(Not run)
```

`gapFilter-methods` *Gap Filter*

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

`object`	object of class `PreFilter`.
`value`	numeric vector `c(cutoff, window, trim, epsilon)`.

Details

The method `gapFilter` initializes the following parameters:

`cutoff`: the cutoff level for the filter.

window:	trim value for the ordered expression levels (default is window=0.05).
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: (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 *Get Chip Name*

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 *Get Chip Type*

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 *Get Chip Name and Type*

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:

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

object	Object of class "ProcesSet".
treetype	type of tree to export, see validTreetype
varlist	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 *Get Tree Names*

Description

Get tree names stored in a ROOT file.

Usage

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

Arguments

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

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

Details

The method `highFilter` initializes the following parameters:

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

---

hist-methods *Plot Density Estimate*

Description

Plot the density estimates for each sample.

Usage

```
hist(x, which = "", size = 0, transfo = log2, xlab = "log intensity",
```

Arguments

x	object of class DataTreeSet or ExprTreeSet.
which	type of probes to be used, for details see validData.
size	length of sequence to be generated as subset.
transfo	a valid function to transform the data, usually "log2", or "0".
xlab	a title for the x axis.
ylab	a title for the y axis.
names	optional vector of sample names.
type	type for the plot.
col	colors to use for the different arrays.
lty	line types to use for the different arrays.
add.legend	logical, if TRUE then a legend will be drawn.
verbose	logical, if TRUE print status information.
...	optional arguments to be passed to plot.

Details

Plots the non-parametric density estimates for each sample.

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.

Note

For objects of class DataTreeSet it is no longer necessary to attachInten since each data tree will be imported separately.

Author(s)

Christian Stratowa

See Also

plotDensity

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.root",sep="/"))

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

---

image-methods *Display an Image*

Description

Creates an image of intensities or residuals, respectively, for each sample.

Usage

```
image(x,          bg      = FALSE,          transfo   = log2,          col      = NULL,
image(x,          type     = c("resids", "pos.resids", "neg.resids", "sign.resids", "weights"),
```

Arguments

x	object of class ProcesSet.
bg	logical. If FALSE, intensities from slot data will be used; if TRUE, background intensities from slot bgrd will be used.
type	character string specifying the type of residual image.
qualopt	character string specifying whether to draw residual image for "raw", "adjusted", or "normalized" intensities.
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 label for the x axis.
ylab	a label for the y axis.
add.legend	logical, if TRUE then a color bar will be drawn.
...	optional arguments to be passed to image.

Details

Creates an image of intensities or residuals, respectively, for each array, i.e. 'pseudo chip images'.

If x belongs to class DataTreeSet then images of raw intensities will be drawn.

If x belongs to class ExprTreeSet and bg=FALSE then images of background corrected intensities will be drawn.

If x belongs to class ExprTreeSet and bg=TRUE the distribution of the 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.

If x belongs to class QualTreeSet then images of the residuals or the probe weights, respectively, will be drawn. For col=NULL the same colors will be used as described in vignette "QualityAssess.pdf" of package affyPLM, using internally function pseudoPalette described in affyPLM.

For names=NULL full tree names will be displayed while for names="namepart" column names will be displayed without name extension. If names is a vector of tree names then data from these trees only will displayed as image(s).

Author(s)

Christian Stratowa

See Also

plotImage

Examples

```
## Not run:
## images of raw intensities as imported using import.data()
unlist(treeNames(data.test3)) # show available tree names
image(data.test3, names="TestA2.cel")
image(data.test3)

## images of background adjusted or background intensities, created by e.g. rma()
getTreeNames(rootFile(data.rma))
image(data.rma, names="TestA2.int")
image(data.rma, names="TestA2.rbg", bg=TRUE)

## residual images, created by e.g. rmaPLM()
getTreeNames(rootFile(rlm.all), treetype="res")
image(rlm.all, type="resids")
image(rlm.all, type="resids", names="TestA2_raw.res", add.legend=TRUE)
image(rlm.all, type="pos.resids", names="TestA2_raw.res", add.legend=TRUE)
image(rlm.all, type="neg.resids", names="TestA2_raw.res", add.legend=TRUE)
image(rlm.all, type="sign.resids", names="TestA2_raw.res", add.legend=TRUE)
image(rlm.all, type="weights", names="TestA2_raw.res", add.legend=TRUE)
image(rlm.all, type="resids", qualopt="adjusted", names="TestA2_adjusted.res", add.legend=TRUE)

## End(Not run)
```

`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

`xps.scheme`	a `SchemeTreeSet` containing the correct scheme for the CEL-files
`filename`	file name of ROOT data file.
`filedir`	system directory where ROOT data file should be stored.
`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 `ProjectInfo`.
`verbose`	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`.

The optional parameter `celnames` allows you to replace the original CEL-file names with names of your choice, otherwise the names of the CEL-files will be used as `celnames`.

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, colons, parenthesis, etc. 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"),"raw",sep="/"))
unlist(treeNames(data.test3))

## import only subset of CEL-files
subdata.test3 <- import.data(scheme.test3,"tmpdt_data_test3",celdir=paste(path.package("xps"),"raw",sep="/"
                 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 September 2011. 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. but use filename="Scheme_HuEx10stv2r2_na32" 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.na32 <- import.exon.scheme("Scheme_HuEx10stv2r2_na32", filedir=scmdir,
                      layoutfile=file.path(libdir, "HuEx-1_0-st-v2_libraryfile", "HuEx-1_0-st-r2/HuEx-1_0-s
                      schemefile=file.path(libdir, "HuEx-1_0-st-v2_libraryfile", "HuEx-1_0-st-r2/HuEx-1_0-s
                      probeset=file.path(anndir, "HuEx-1_0-st-v2.na32.hg19.probeset.csv"),
                      transcript=file.path(anndir, "HuEx-1_0-st-v2.na32.hg19.transcript.csv"))

## access ROOT scheme file from new R session
scheme.exon <- root.scheme(paste(scmdir,"Scheme_HuEx10stv2r2_na32.root",sep="/"))


## create scheme for HuGene-1_0-st-v1.r4 as exon array
scheme.hugene10stv1r4.na32 <- import.exon.scheme("Scheme_HuGene10stv1r4_na32",filedir=scmdir,
                        layoutfile=file.path(libdir, "HuGene-1_0-st-v1.r4.analysis-lib-files", "HuGene-1_0-
                        schemefile=file.path(libdir, "HuGene-1_0-st-v1.r4.analysis-lib-files", "HuGene-1_0-
                        probeset=file.path(anndir, "HuGene-1_0-st-v1.na32.hg19.probeset.csv"),
                        transcript=file.path(anndir, "HuGene-1_0-st-v1.na32.hg19.transcript.csv"))

## access ROOT scheme file from new R session
scheme.gene <- root.scheme(file.path(scmdir, "Scheme_HuGene10stv1r4_na32.root"))


## 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=file.path(libdir, "HuEx-1_0-st-v2_libraryfile", "HuEx-1_0-st-r2", "HuEx-1_0
                     schemefile=file.path(libdir, "HuEx-1_0-st-v2_libraryfile", "HuEx-1_0-st-r2", "HuEx-1_0
                          probeset=file.path(anndir, "HuEx-1_0-st-probeset-annot.csv"),
                       transcript=file.path(anndir, "HuEx-1_0-st-transcript-annot.csv"),
                     control=file.path(libdir, "HuEx-1_0-st-v2_libraryfile", "HuEx-1_0-st-r2", "HuEx-1_0-st

## 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 Afymetrix 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.na32" but use filename="Scheme_Test3_na32" or simply filename="SchemeTest3na32" 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
scmdir <- "/common/path/schemes"
libdir <- "/my/path/Affy/libraryfiles"
anndir <- "/my/path/Affy/Annotation"

## create scheme for Test3 GeneChip
scheme.test3.na32 <- import.expr.scheme("Scheme_Test3_na32", filedir=scmdir,
                     schemefile=file.path(libdir, "Test3.CDF"),
                     probefile=file.path(libdir, "Test3_probe.tab"),
                     annotfile=file.path(anndir, "Test3.na32.annot.csv"))

## access ROOT scheme file from new R session
scheme.test3 <- root.scheme(file.path(scmdir, "Scheme_Test3_na32.root"))


## create scheme for HG-U133_Plus_2 GeneChip
scheme.hgu133p2.na32 <- import.expr.scheme("Scheme_HGU133p2_na32", filedir=scmdir,
                        schemefile=file.path(libdir, "HG-U133_Plus_2.cdf"),
                        probefile=file.path(libdir, "HG-U133-PLUS_probe.tab"),
                        annotfile=file.path(anndir, "HG-U133_Plus_2.na32.annot.csv"))

## access ROOT scheme file from new R session
scheme.hgu133p2 <- root.scheme(file.path(scmdir, "Scheme_HGU133p2_na32.root"))

## 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 Afymetrix 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.HuGene10stv1. but use filename="Scheme_HuGene10stv1_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
scmdir <- "/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.hugene10stv1r3.na27 <- import.genome.scheme("Scheme_HuEx10stv1r3_na27", filedir=scmdir,
                        layoutfile=file.path(libdir, "HuGene-1_0-st-v1.r3.analysis_libraryfile", "HuGene-1_
                        schemefile=file.path(libdir, "HuGene-1_0-st-v1.r3.analysis_libraryfile", "HuGene-1_
                        transcript=file.path(anndir, "HuGene-1_0-st-v1.na27.hg18.transcript.csv"))

## access ROOT scheme file from new R session
scheme.hugene10stv1r3 <- root.scheme(file.path(scmdir, "Scheme_HuEx10stv1r3_na27.root"))

## End(Not run)
```

---

indexUnits-methods *Unit Locations*

Description

Returns a data.frame or list with locations of the probes in each probe set.

Usage

```
indexUnits(object, which = "", unitID = NULL, unittype = "transcript", as.list = TRUE, data = NULL)
pmindex(object, unitID = NULL, as.list = TRUE)
mmindex(object, unitID = NULL, as.list = TRUE)
```

Arguments

object	Object of class "DataTreeSet".
which	type of probes to be used, for details see validData.
unitID	optional vector of UNIT_IDs.
unittype	character vector, “transcript” or “probeset”.
as.list	if TRUE a list will be returned (default is data.frame).
data	optional data.frame containing (x,y)-coordinates.

Details

Function indexUnits returns a data.frame or list with locations of the probes in each probe set.

By default a data.frame for selected unitIDs or all unitIDs (unitID="*") will be returned with columns <UNIT_ID, X, Y, XY>. Here "XY" are the selected rows of slot data.

For as.list=TRUE a list of unitIDs will be returned containing the selected rows "XY". The names of the elements in the list returned are the UNIT_IDs.

For unitID=NULL a vector of data rows "XY" will be returned.

For expression arrays which can be one of "pm", "mm", or "both". Alternatively, functions pmindex and mmindex can be used for PM probes or MM probes, respectively.

For exon arrays which is described in validData. However, in this case slot data must contain the (x,y)-coordinates of the probesetIDs.

Value

A list or data.frame.

Author(s)

Christian Stratowa

See Also

unitID2transcriptID, unitID2probesetID

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.root",sep="/"))

## dataXY not attached
id <- indexUnits(data.test3, which="pm", unitID=c(34,36,122))
id

## dataXY attached (only necessary for whole genome and exon arrays)
data.test3 <- attachDataXY(data.test3)
xy <- treeData(data.test3)
id <- indexUnits(data.test3, which="pm", unitID=c(34,36,122), data=xy)
id
id <- indexUnits(data.test3, which="", unitID=c(34,36,122), data=xy)
id
id <- indexUnits(data.test3, which="", unitID=34, as.list=FALSE, data=xy)
id
data.test3 <- removeDataXY(data.test3)

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

---

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_cluste
`exon`:	expression levels are computed for exon clusters, i.e. probe sets containing the same 'exon_id', where each
`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

Talloen, W., Clevert D.-A., Hochreiter, S., Amaratunga, D., Bijnens, 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.root",sep="/"))

## 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

`object`	object of class `DataTreeSet`.
`filename`	`character` vector containing optional ROOT file name.
`verbose`	logical, if `TRUE` print status information.
`value`	`data.frame` 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 preferable 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.root",sep="/"))

## 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,verbose=FALSE)

## End(Not run)
```

---

intensity2GCplot-methods

Boxlot of Probe Intensities Stratified by GC Content.

Description

Creates a boxplot of probe intensities stratified by GC content.

Usage

```
intensity2GCplot(x,            treename,            which  = "",            transfo = log2
```

Arguments

x	object of class DataTreeSet.
treename	character vector, tree name containing intensities.
which	type of probes to be used, for details see validData.
transfo	a valid function to transform the data, usually "log2", or "0".
range	determines how far the plot whiskers extend out from the box.
col	color pair to be used by function colorRampPalette.
...	optional arguments to be passed to intensity2GCplot.

Details

Creates a boxplot of probe intensities for treename stratified by GC content for an object of class DataTreeSet.

Note

G/C content must first be attached to class DataTreeSet using method attachProbeContentGC. It is also recommended to attach the probe mask using method attachMask.

Author(s)

Christian Stratowa

See Also

plotIntensity2GC

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.root",sep="/"))

## need to attach probe G/C content and optionally mask
data.test3 <- attachProbeContentGC(data.test3)
data.test3 <- attachMask(data.test3)

if (interactive()) {
intensity2GCplot(data.test3, treename = "TestA1.cel", which="mm")
}

## optionally remove probe G/C content and mask to free memory
data.test3 <- removeMask(data.test3)
data.test3 <- removeProbeContentGC(data.test3)
```

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:

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

object	object of class PreFilter.
value	numeric vector c(cutoff, epsilon).

Details

The method madFilter initializes the following parameters:

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

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)
```

`madplot-methods` *Array-Array Expression Level Distance Plot*

Description

A false color display of between arrays distances, computed as the MAD of the M-values of each pair of arrays.

Usage

```
madplot(x, which = "UnitName", transfo = log2, col = NULL, name
```

Arguments

`x`	object of class `ExprTreeSet`.
`which`	type of probes to be used, for details see `validData`.
`transfo`	a valid function to transform the data, usually "log2", or "0".
`col`	vector of colors for plot, length is number of samples.
`names`	optional vector of sample names.
`sort`	logical, if TRUE the correlation matrix will be sorted decreasingly.
`bmar`	optional `list` for bottom margin and axis label magnification `cex.axis`.
`add.legend`	logical, if TRUE then a color bar will be drawn.
`...`	optional arguments to be passed to `plot`.

Details

Produces a false color display, i.e. heatmap, of between array distances for slot `data` for an object of class `ExprTreeSet`, computed as the MAD of the M-values of each pair of arrays.

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 mdaplot.

For `bmar=NULL` the default list `bmar = list(b=6, cex.axis=1.0)` will be used initially. However, both bottom margin and axis label magnification will be adjusted depending on the number of label characters and the number of smaples.

Author(s)

Christian Stratowa

See Also

`plotMAD`, `corplot`

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

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.
normalize	logical. If TRUE scale normalization is used after an ExprTreeSet is obtained.
sc	value at which all arrays will be scaled to.
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.
update	logical. If TRUE the existing ROOT data file filename will be updated.
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.
object	object of class DataTreeSet.
...	arguments filename,filedir,tmpdir,option,exonlevel,xps.scheme.

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.root",sep="/"))

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

## 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 *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

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.
normalize	logical. If TRUE scale normalization is used after an ExprTreeSet is obtained.
sc	value at which all arrays will be scaled to.
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.
update	logical. If TRUE the existing ROOT data file filename will be updated.
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.
object	object of class DataTreeSet.
...	arguments filename,filedir,tmpdir,option,exonlevel,xps.scheme.

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:

`transcript`:	expression levels are computed for transcript clusters, i.e. probe sets containing the same 'transcript_cluste
`exon`:	expression levels are computed for exon clusters, i.e. probe sets containing the same 'exon_id', where each
`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 (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+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'.

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

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.root",sep="/"))

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

## 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()
```

---

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, bgcorrect.option = "nor
        option = "transcript", exonlevel = "", xps.scheme = NULL, add.data = TRUE, verbose = TRUE)

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.
bgcorrect.option	bgcorrect option determining wether to subtract background first, one of 'none' or 'correctbg'.
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 Ri = (PMi - MMi) / (PMi + MMi) 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.Furthermore, in these cases it may be better to use bgcorrect.option = "correctbg" to get reasonable present calls. Note that 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

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.root",sep="/"))

## 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()
```

---

mboxplot-methods *Box Plots of Relative M Values*

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), outl
```

Arguments

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

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

mvaplot, 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.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="metacore")
```

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-methods` *M vs A Plot*

Description

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

Usage

```
mvaplot(x, which = "UnitName", transfo = log2, method = "median", names
```

Arguments

`x`	object of class `ExprTreeSet`.
`which`	type of probes to be used, for details see `validData`.
`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 label for the x axis.
`ylab`	a label for the y axis.
`pch`	an integer specifying a symbol or a character to be used as the default in plotting points.
`las`	the style of axis labels.
`...`	optional arguments to be passed to `plot`.

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

`plotMA`

namePart *Get Tree Names w/o Extension*

Description

Get (tree) names w/o their extension.

Usage

```
namePart(names)
```

Arguments

names	vector of names.

Details

Extracts the name part of names, e.g.of tree names of treename.treetype stored in a ROOT file.

Value

A vector of tree names w/o its extension.

Author(s)

Christian Stratowa

See Also

extenPart

Examples

```
names <- c("TestA1.int", "TestA2.int")
namePart(names)
```

normalize *Normalization on Affymetrix Probe Level Data or on Expression Levels*

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, selec

normalize.constant(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FAL

normalize.lowess(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE

normalize.quantiles(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FA

normalize.supsmu(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE

xpsNormalize(object, ...)
```

Arguments

xps.data	object of class DataTreeSet or ExprTreeSet.
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 filename 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 refindex=0, 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 DataTreeSet or ExprTreeSet.
...	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").

Please have a look at vignette "xpsPreprocess.pdf" for details on how to use function normalize.

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.root",sep="/"))

## RMA background
data.bg.rma <- bgcorrect.rma(data.test3,"tmp_Test3NormRMA",filedir=getwd(),tmpdir="",verbose=FALSE)
## normalize quantiles
data.qu.rma <- normalize.quantiles(data.bg.rma,"tmp_Test3NormRMA",filedir=getwd(),tmpdir="",update=TRUE,ver
## summarize medianpolish
data.mp.rma <- summarize.rma(data.qu.rma,"tmp_Test3NormRMA",filedir=getwd(),tmpdir="",update=TRUE,verbose=F
```

NUSE-methods *Normalized Unscaled Standard Errors (NUSE)*

Description

Produce boxplot of Normalized Unscaled Standard Errors (NUSE) for the set of arrays. Alternatively, summary statistics or NUSE values can be extracted as data.frame.

Usage

```
NUSE(x,    treename = "*",    type    = c("plot", "stats", "values"),    qualopt  = NULL,
```

Arguments

x	object of class QualTreeSet.
treename	vector of tree names to export.
type	type of output, plot, stats or values.
qualopt	quality control option, i.e. 'raw', 'adjusted', 'normalized' or 'all'.
...	optional arguments to be passed to nuseplot.

Details

Create boxplots of Normalized Unscaled Standard Errors (NUSE) for the set of arrays.

Alternatively it is possible to extract either the summary statistics as data.frame (type="stats") or all NUSE values as data.frame (type="values").

If an object of class QualTreeSet was created by fitting a probe level model with qualopt="all" then NUSE will plot or extract NUSE for "all" quality options. If you want to plot or extract NUSE for a certain quality option only, e.g. "normalized" data only, then you can use parameter qualopt with qualopt="<qualopt>".

Author(s)

Christian Stratowa

See Also

plotNUSE, nuseplot

Examples

```
## Not run:
## 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.root",sep="/"))

## qualification - rlm
rlm.all <- rmaPLM(data.test3, "tmp_Test3RLMall", filedir=getwd(), tmpdir="", qualopt="all", option="transcri

## plot expression levels
if (interactive()) {
NUSE(rlm.all)
NUSE(rlm.all, qualopt="normalized")
qcNUSE <- NUSE(rlm.all, type="stats")
qcNUSE <- NUSE(rlm.all, type="values")
qcNUSE <- NUSE(rlm.all, treename="TestA1_normalized.rlm", type="stats")
qcNUSE <- NUSE(rlm.all, treename="TestA1_normalized.rlm", type="values")
}

## End(Not run)
```

---

nuseplot-methods *Box Plots of Normalized Unscaled Standard Errors (NUSE)*

Description

Produce boxplot of Normalized Unscaled Standard Errors (NUSE) for the set of arrays.

Usage

```
nuseplot(x,       which    = "UnitName",       size    = 0,       range    = 0,       names    = "
```

Arguments

x	object of class or QualTreeSet.
which	type of probes to be used, for details see validData (only ExprTreeSet).
size	length of sequence to be generated as subset (only ExprTreeSet).
range	determines how far the plot whiskers extend out from the box.
names	optional vector of sample names.
main	the main title for the plot.
ylim	range for the plotted y values.
las	the style of axis labels.
add.line	logical, if TRUE a horizontal line is drawn.
outline	if outline is not true, the outliers are not drawn (only ExprTreeSet).
...	optional arguments to be passed to boxplot.

Details

Create boxplots of Normalized Unscaled Standard Errors (NUSE) for the set of arrays.

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.

If an object of class `QualTreeSet` was created by fitting a probe level model with `qualopt="all"` then `nuseplot` will plot NUSE for "all" quality options. If you want to plot NUSE for a certain quality option only, e.g. "normalized" data only, then you can use parameter `names` with `names="namepart:<qualopt>"`, e.g. `names="namepart:normalized"`.

Author(s)

Christian Stratowa

See Also

`NUSE`, `plotNUSE`, `rleplot`

Examples

```
# load existing ROOT scheme file and ROOT expression file for rma
scheme.test3 <- root.scheme(paste(path.package("xps"),"schemes/SchemeTest3.root",sep="/"))
data.rma <- root.expr(scheme.test3, paste(path.package("xps"),"rootdata/tmp_Test3RMA.root",sep="/"), "mdp")

if (interactive()) {
nuseplot(data.rma)
}
```

`pcaplot-methods` *PCA Plot*

Description

This function produces a PCA plot of the first two principle components.

Usage

```
pcaplot(x, which = "UnitName", transfo = log2, method = "none", g
```

Arguments

`x`	object of class `ExprTreeSet`.
`which`	type of probes to be used, for details see `validData`.
`transfo`	a valid function to transform the data, usually “log2”, or “0”.
`method`	a character string indicating which correlation coefficient is to be computed. One of “pearson”, “spearman”, “kendall”, or “none”.
`groups`	character `vector` listing the group names in order of the `names`.
`screeplot`	logical, if TRUE plot a `screeplot` instead of a PCA plot.
`squarepca`	logical, if TRUE make the y-axis of the PCA plot comparable to the x-axis.
`pcs`	a character `vector` of length two indicating which principal components to plot.

add.labels	logical, if TRUE then name labels will be added to the points.
add.legend	logical, if TRUE and groups are supplied then a legend indicating the groups will be drawn. Optionally, a character indicating the position of the legend, default is "topleft".
col	vector of colors for plot, length is number of samples.
names	optional vector of sample names.
as.list	logical, if TRUE then a list will be returned in addition to the plot.
...	optional arguments to be passed to plot.

Details

Function pcaplot produces a PCA plot of the first two principle components for slot data or the correlations between the columns of slot data, respectively, of an object of class ExprTreeSet.

For method="none" function [stats]prcomp will be applied to slot data directly, otherwise prcomp will be applied to (1 - cor(data)) with the respective method.

For screeplot=TRUE a screeplot will be plotted instead of a PCA plot.

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.

Value

None by default.

Optionally, for as.list=TRUE a list will be returned with the components sdev and rotation, see [stats]prcomp.

Author(s)

Christian Stratowa, partly adapted from function plotPCA() of package affycoretools

See Also

plotPCA, corplot madplot

---

plotBorder *Plots of Border Elements for Device*

Description

Produce box-and-whisker plot(s) of the positive and negative feature intensities for the selected device.

Usage

```
plotBorder(x,
           type    = c("pos", "neg"),
           qualopt = "raw",
           transfo = log2,
           range   = 0,
           names   = "namepart",
           ylim    = NULL,
           bmar    = NULL,
           las     = 2,
           dev     = "screen",
           outfile = "BorderPlot",
           w       = 800,
           h       = 540,
           ...)
```

Arguments

x	object of class QualTreeSet.
type	type of border elements to be used, one of "pos", "neg", or both.
qualopt	character string specifying whether to draw boxplots for "raw", "adjusted", or "normalized" border intensities.
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.
ylim	the y limits of the plot.
bmar	optional list for bottom margin and axis label magnification cex.axis.
las	the 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 borderplot.

Details

Creates a boxplot of the positive and negative feature intensities for an object of class QualTreeSet.

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

For bmar=NULL the default list bmar = list(b=6, cex.axis=1.0) will be used initially. However, both bottom margin b and axis label magnification cex.axis will be adjusted depending on the number of label characters and the number of samples.

Author(s)

Christian Stratowa

See Also

borderplot

---

plotBoxplot *Box Plots for Device*

Description

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

Usage

```
plotBoxplot(x,
            which   = "",
            size    = 0,
            transfo = log2,
            range   = 0,
            names   = "namepart",
            mar     = NULL,
            las     = 2,
            cex     = 1.0,
            dev     = "screen",
            outfile = "BoxPlot",
            w       = 800,
            h       = 540,
            ...)
```

Arguments

x	object of class DataTreeSet or ExprTreeSet.
which	type of probes to be used, for details see validData.
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.
mar	plot margin.
las	style of axis labels.
cex	amount by which plotting text and symbols should be magnified.
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 boxplot.

Details

Produces a boxplot for slot data for an object of class DataTreeSet, ExprTreeSet or QualTreeSet for the selected graphics device.

Author(s)

Christian Stratowa

See Also

boxplot, plotBorder, plotNUSE, plotRLE

---

plotCall *Barplot of Percent Present and Absent Calls for Device*

Description

Creates a barplot of percent Present/Marginal/Absent calls for the selected device.

Usage

```
plotCall(x,
         beside  = TRUE,
         names   = "namepart",
         col     = c("red","green","blue"),
         legend  = c("P","M","A"),
         ylim    = c(0,100),
         ylab    = "detection call [%]",
         las     = 2,
         dev     = "screen",
         outfile = "CallPlot",
         w       = 800,
         h       = 540,
         ...)
```

Arguments

x	object of class CallTreeSet.
beside	logical. If FALSE, the columns of height are portrayed as stacked bars, and if TRUE the columns are portrayed as juxtaposed bars.
names	optional vector of sample names.
col	color for P/M/A bars
legend	legend for the plot, defaults to P/M/A.
ylim	the y limits of the plot.
ylab	a label for the y axis.
las	the 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 barplot.

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

`callplot`

`plotCOI` *Center-Of-Intensity QC Plots for Device*

Description

Produce Center-Of-Intensity plot(s) of the positive and negative feature intensities for the selected device.

Usage

```
plotCOI(x,
        type    = c("pos", "neg"),
        qualopt = "raw",
        radius  = 0.5,
        linecol = "gray70",
        visible = TRUE,
        dev     = "screen",
        outfile = "CenterOfIntensityPlot",
        w       = 540,
        h       = 540,
        ...)
```

Arguments

`x`	object of class `QualTreeSet`.
`type`	type of border elements to be used, one of “pos”, “neg”, or both.
`qualopt`	character string specifying whether to draw boxplots for “raw”, “adjusted”, or “normalized” border intensities.
`radius`	determines the radius within which the COI for each array should be located.
`linecol`	the color of the ablines and the circle to be drawn.
`visible`	logical, if TRUE then arrays outside the circle with `radius` will be flagged by labeling the data point with the array name.
`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 `coiplot`.

Details

Produces Center-Of-Intensity (COI) plot(s) of the positive and negative feature intensities for an object of class QualTreeSet. This plot is useful for detecting spatial biases in intensities on an array.

Mean intensities for the left, right, top and bottom border elements are calculated, separated into positive and negative controls, and the "center of intensity" is calculated on a relative scale [-1,1]. Arrays with a COI outside a range with radius are considered to be outliers. If visible = TRUE then outlier arrays will be flagged by labeling the data point(s) with the array name(s).

Author(s)

Christian Stratowa

See Also

coiplot

---

plotCorr *Array-Array Expression Level Correlation Plot for Device*

Description

A heat map of the array-array Spearman rank correlation coefficients for the selected device.

Usage

```
plotCorr(x,
         which      = "UnitName",
         transfo    = log2,
         method     = "spearman",
         col        = NULL,
         names      = "namepart",
         sort       = FALSE,
         reverse    = TRUE,
         bmar       = NULL,
         add.legend = FALSE,
         dev        = "screen",
         outfile    = "CorrelationPlot",
         w          = 540,
         h          = 540,
         ...)
```

Arguments

x	object of class ExprTreeSet.
which	type of probes to be used, for details see validData.
transfo	a valid function to transform the data, usually "log2", or "0".
method	a character string indicating which correlation coefficient is to be computed.
col	vector of colors for plot, length is number of samples.

names	optional vector of sample names.
sort	logical, if TRUE the correlation matrix will be sorted decreasingly.
reverse	logical, if TRUE the correlation matrix will be replaced by 1 - cor().
bmar	optional list for bottom margin and axis label magnification cex.axis.
add.legend	logical, if TRUE then a color bar will be drawn.
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 plot.

Details

Produces a heat map of the array-array Spearman rank correlation coefficients 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 corplot.

For bmar=NULL the default list bmar = list(b=6, cex.axis=1.0) will be used initially. However, both bottom margin and axis label magnification will be adjusted depending on the number of label characters and the number of samples.

Author(s)

Christian Stratowa

See Also

corplot

---

plotDensity *Plot Density Estimate for Device*

Description

Plot the density estimates for each sample for the selected device.

Usage

```
plotDensity(x,
            which      = "",
            size       = 0,
            transfo    = log2,
            ylab       = "density",
            xlab       = "log intensity",
            names      = "namepart",
            type       = "l",
            col        = 1:6,
```

```
lty        = 1:5,
add.legend = FALSE,
dev        = "screen",
outfile    = "DensityPlot",
w          = 540,
h          = 540,
verbose    = TRUE,
...)
```

Arguments

x	object of class DataTreeSet or ExprTreeSet.
which	type of probes to be used, for details see validData.
size	length of sequence to be generated as subset.
transfo	a valid function to transform the data, usually "log2", or "0".
xlab	a title for the x axis.
ylab	a title for the y axis.
names	optional vector of sample names.
type	type for the plot.
col	colors to use for the different arrays.
lty	line types to use for the different arrays.
add.legend	logical, if TRUE then a legend will be drawn.
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.
verbose	logical, if TRUE print status information.
...	optional arguments to be passed to plot.

Details

Plots the non-parametric density estimates for each sample.

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

hist

plotImage *Plot Image(s) for Device*

Description

Creates an image for each sample for the selected device.

Usage

```
plotImage(x,
          type     = character(),
          qualopt  = c("raw", "adjusted", "normalized"),
          transfo  = log2,
          col      = NULL,
          names    = character(),
          dev      = "screen",
          outfile  = "Image",
          w        = 800,
          h        = 800,
          verbose  = TRUE,
          ...)
```

Arguments

x	object of class DataTreeSet or QualTreeSet.
type	character string specifying the type of image.
qualopt	character string specifying whether to draw residual image for "raw", "adjusted", or "normalized" intensities.
transfo	a valid function to transform the data, usually "log2", or "0".
col	color range for intensities.
names	vector of sample names.
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.
verbose	logical, if TRUE print status information.
...	optional arguments to be passed to image.

Details

Creates intensity image(s) or residual image(s), respectively, for each array for the selected graphics device, see image for more details.

For intensity image(s) type must be one of "intensity".

For residual image(s) type must be one of "resids","pos.resids", "neg.resids", "sign.resids", or "weights". Furthermore, qualopt determines if images should be drawn for "raw", "adjusted", or "normalized" data.

For names="*" names of all samples will be displayed as images. If names is a vector of column names, only these samples will displayed as image(s).

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",sep="/"))

## qualification - rlm
rlm.all <- rmaPLM(data.test3, "tmp_Test3RLMall", filedir=getwd(), tmpdir="", qualopt="all", option="transcri

if (interactive()) {
## image(s) of raw data
plotImage(data.test3, type="intensity", names="*")
plotImage(data.test3, type="intensity", names="TestA2.cel")

## image(s) of residuals/weights
plotImage(rlm.all, type="weights", names="*")
plotImage(rlm.all, type="weights", qualopt="adjusted", names="*")
plotImage(rlm.all, type="resids", names="TestA2_raw.res")
}

## function image.dev() will be deprecated since it needs attachInten!!
## 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 comsumption of R remove data:
data.test3 <- removeInten(data.test3)
data.test3 <- removeMask(data.test3)

## End(Not run)
```

---

plotIntensity2GC *Boxlot of Probe Intensities Stratified by GC Content for Device.*

Description

Creates a boxplot of probe intensities stratified by GC content for the selected device.

Usage

```
plotIntensity2GC(x,
                 treename,
                 which   = "",
```

```
transfo = log2,
range   = 0,
col     = c("lightblue", "darkblue"),
dev     = "screen",
outfile = "Intensity2GCPlot",
w       = 540,
h       = 540,
...)
```

Arguments

x	object of class DataTreeSet.
treename	character vector, tree name used for intensities.
which	type of probes to be used, for details see validData.
transfo	a valid function to transform the data, usually "log2", or "0".
range	determines how far the plot whiskers extend out from the box.
col	color pair to be used by function colorRampPalette.
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 plotIntensity2GC.

Details

Creates a boxplot of probe intensities for treename stratified by GC content for an object of class DataTreeSet.

Note

G/C content must first be attached to class DataTreeSet using method attachProbeContentGC. It is also recommended to attach the probe mask using method attachMask.

Author(s)

Christian Stratowa

See Also

intensity2GCplot

plotMA *MvA Scatter Plot for Device*

Description

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

Usage

```
plotMA(x,
       transfo = log2,
       method  = "median",
       names   = "namepart",
       ylim    = c(-6, 6),
       xlab    = "A",
       ylab    = "M",
       pch     = ".",
       mar     = c(3, 3, 2, 1),
       dev     = "screen",
       outfile = "MvAPlot",
       w       = 540,
       h       = 540,
       ...)
```

Arguments

x	object of class ExprTreeSet.
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 plot.

Details

Produces M vs A plots 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 M vs A plot.

Author(s)

Christian Stratowa

See Also

mvaplot

---

plotMAD *Array-Array Expression Level Distance Plot for Device*

Description

A false color display of between arrays distances, computed as the MAD of the M-values of each pair of arrays for the selected device.

Usage

```
plotMAD(x,
        which      = "UnitName",
        transfo    = log2,
        col        = NULL,
        names      = "namepart",
        sort       = FALSE,
        bmar       = NULL,
        add.legend = FALSE,
        dev        = "screen",
        outfile    = "MADPlot",
        w          = 540,
        h          = 540,
        ...)
```

Arguments

x	object of class ExprTreeSet.
which	type of probes to be used, for details see validData.
transfo	a valid function to transform the data, usually “log2”, or “0”.
col	vector of colors for plot, length is number of samples.
names	optional vector of sample names.
sort	logical, if TRUE the correlation matrix will be sorted decreasingly.
bmar	optional list for bottom margin and axis label magnification cex.axis.
add.legend	logical, if TRUE then a color bar will be drawn.
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 plot.

Details

Produces a false color display, i.e. heatmap, of between array distances for slot `data` for an object of class `ExprTreeSet`, computed as the MAD of the M-values of each pair of arrays.

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 mdaplot.

For `bmar=NULL` the default list `bmar = list(b=6, cex.axis=1.0)` will be used initially. However, both bottom margin and axis label magnification will be adjusted depending on the number of label characters and the number of smaples.

Author(s)

Christian Stratowa

See Also

`madplot`

`plotNUSE` *Box Plots of Normalized Unscaled Standard Errors (NUSE) for Device*

Description

Produce boxplot of Normalized Unscaled Standard Errors (NUSE) for the set of arrays and the selected device.

Usage

```
plotNUSE(x,
         which    = "UnitName",
         size     = 0,
         range    = 0,
         names    = "namepart",
         main     = "NUSE Plot",
         ylim     = c(0.8,1.2),
         las      = 2,
         add.line = TRUE,
         outline  = FALSE,
         dev      = "screen",
         outfile  = "NUSEPlot",
         w        = 800,
         h        = 540,
         ...)
```

Arguments

`x`	object of class `ExprTreeSet` or `QualTreeSet`.
`which`	type of probes to be used, for details see `validData`.
`size`	length of sequence to be generated as subset.

range	determines how far the plot whiskers extend out from the box.
names	optional vector of sample names.
main	the main title for the plot.
ylim	range for the plotted y values.
las	the style of axis labels.
add.line	logical, if TRUE a horizontal line is drawn.
outline	if outline is not true, the outliers are not drawn.
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 boxplot.

Details

Create boxplots of Normalized Unscaled Standard Errors (NUSE) for the set of arrays.

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.

Author(s)

Christian Stratowa

See Also

nuseplot

---

plotPCA *PCA Plot for Device*

Description

This function produces a PCA plot of the first two principle components for the selected device.

Usage

```
plotPCA(x,
        which      = "UnitName",
        transfo    = log2,
        method     = "none",
        groups     = NULL,
        screeplot  = FALSE,
        squarepca  = FALSE,
        pcs        = c(1,2),
        add.labels = FALSE,
        add.legend = FALSE,
        col        = NULL,
```

```
        names      = "namepart",
        as.list    = FALSE,
        dev        = "screen",
        outfile    = "PCAPlot",
        w          = 540,
        h          = 540,
        ...)
```

Arguments

x	object of class ExprTreeSet.
which	type of probes to be used, for details see validData.
transfo	a valid function to transform the data, usually "log2", or "0".
method	a character string indicating which correlation coefficient is to be computed. One of "pearson", "spearman", "kendall", or "none".
groups	character vector listing the group names in order of the names.
screeplot	logical, if TRUE plot a screeplot instead of a PCA plot.
squarepca	logical, if TRUE make the y-axis of the PCA plot comparable to the x-axis.
pcs	a character vector of length two indicating which principal components to plot.
add.labels	logical, if TRUE then name labels will be added to the points.
add.legend	logical, if TRUE and groups are supplied then a legend indicating the groups will be drawn. Optionally, a character indicating the position of the legend, default is "topleft".
col	vector of colors for plot, length is number of samples.
names	optional vector of sample names.
as.list	logical, if TRUE then a list will be returned in addition to the plot.
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 plot.

Details

Function plotPCA produces a PCA plot of the first two principle components for slot data or the correlations between the columns of slot data, respectively, of an object of class ExprTreeSet.

For method="none" function [stats]prcomp will be applied to slot data directly, otherwise prcomp will be applied to (1 - cor(data)) with the respective method.

For screeplot=TRUE a screeplot will be plotted instead of a PCA plot.

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

pcaplot

---

plotPM *Barplot of PM and MM Intensities for Device*

Description

Creates a barplot of mean perfect match and mismatch intensities for the selected device.

Usage

```
plotPM(x,
       which   = "",
       size    = 0,
       transfo = NULL,
       method  = mean,
       names   = "namepart",
       beside  = TRUE,
       col     = c("red", "blue"),
       legend  = c("PM","MM"),
       las     = 2,
       ylab    = "mean intensities",
       dev     = "screen",
       outfile = "PMPlot",
       w       = 540,
       h       = 540,
       ...)
```

Arguments

x	object of class DataTreeSet.
which	type of probes to be used, for details see validData.
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.
las	the style of axis labels.
ylab	a title for the y axis.
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 barplot.

Details

Produces barplots of mean perfect match and mismatch intensities for slot `data` for an object of class `DataTreeSet`.

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.

Author(s)

Christian Stratowa

See Also

`pmplot`

`plotProbeset` *Plot of Probe Intensities for a Probeset for Device.*

Description

Creates a line plot of probe intensities for a probeset for the selected device.

Usage

```
plotProbeset(x,
             unitID,
             unittype   = "transcript",
             which      = "pm",
             transfo    = log2,
             names      = "namepart",
             ylim       = NULL,
             col        = 1:6,
             lty        = 1:5,
             add.legend = FALSE,
             dev        = "screen",
             outfile    = "ProbesetPlot",
             w          = 540,
             h          = 540,
             ...)
```

Arguments

`x`	object of class `DataTreeSet`.
`unitID`	unit ID of probeset with type of ID determined by parameter `unittype`.
`unittype`	`character` vector, one of "unit", "transcript", "probeset".
`which`	type of probes to be used, for details see `validData`.
`transfo`	a valid function to transform the data, usually "log2", or "0".
`names`	optional vector of sample names.
`ylim`	range for the plotted y values.

col	color to use for the different samples.
lty	line types to use for the different samples.
add.legend	logical, if TRUE a legend of sample names will be drawn. Optionally, a character indicating the position of the legend, default is "topleft".
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 plotProbeset.

Details

Produces line plots of the probe intensities for probeset unitID. Probe intensities are taken from slot data.

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, line plots of probe intensities will only be drawn for these columns.

Note

Data must first be attached to class DataTreeSet using method attachInten. Furthermore, unit names must be attached using method attachUnitNames.

Author(s)

Christian Stratowa

See Also

probesetplot

---

plotRLE *Box Plots of Relative Log Expression (RLE) for Device*

Description

Produce boxplot of Relative Log Expression (RLE) for the set of arrays and the selected device.

Usage

```
plotRLE(x,
        which    = "UnitName",
        size     = 0,
        range    = 0,
        names    = "namepart",
        main     = "RLE Plot",
        ylim     = c(-1.0, 1.0),
        las      = 2,
        add.line = TRUE,
```

```
outline  = FALSE,
dev      = "screen",
outfile  = "RLEPlot",
w        = 800,
h        = 540,
verbose  = TRUE,
...)
```

Arguments

x	object of class ExprTreeSet or QualTreeSet.
which	type of probes to be used, for details see validData.
size	length of sequence to be generated as subset.
range	determines how far the plot whiskers extend out from the box.
names	optional vector of sample names.
main	the main title for the plot.
ylim	range for the plotted y values.
las	the style of axis labels.
add.line	logical, if TRUE a horizontal line is drawn.
outline	if outline is not true, the outliers are not drawn.
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.
verbose	logical, if TRUE print status information.
...	optional arguments to be passed to boxplot.

Details

Create boxplots of Relative Log Expression (RLE) values for the set of arrays, i.e. of M plots, where M is determined relative to a pseudo-median 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.

Author(s)

Christian Stratowa

See Also

rleplot

plotVolcano *Volcano Plot*

Description

Produce a scatter plot of fold-change values vs p-values, called volcano plot.

Usage

```
plotVolcano(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         = '.',
            dev         = "screen",
            outfile     = "VolcanoPlot",
            w           = 540,
            h           = 540,
            ...)
```

Arguments

x	object of class AnalysisTreeSet.
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.
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 barplot.

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

See Also

`volcanoplot`

`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", unitID = NULL, unittype = "transcript")
mm(object, which = "mm", unitID = NULL, unittype = "transcript")
```

Arguments

`object`	object of class `DataTreeSet`.
`which`	type of perfect match or mismatch probes to be returned.
`unitID`	optional vector of UNIT_IDs.
`unittype`	`character` vector, "transcript" or "probeset".

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:

`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.

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:

core:	probesets with category 'unique' and 'mixed'.
metacore:	probesets with category 'unique' only.
affx:	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.root",sep="/"))

## 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)

## need to convert Affy ID to UNIT_ID first
id <- transcriptID2unitID(schemeSet(data.test3), transcriptID="100084_at", as.list=FALSE)
pm <- pm(data.test3, unitID=id)
mm <- mm(data.test3, unitID=id)
head(pm)
head(mm)

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

pmplot-methods *Barplot of PM and MM Intensities.*

Description

Creates a barplot of mean perfect match and mismatch intensities.

Usage

```
pmplot(x, which = "", size = 0, transfo = NULL, method = "mean", name
```

Arguments

x	object of class DataTreeSet.
which	type of probes to be used, for details see validData.
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.
las	the style of axis labels.
ylab	a title for the y axis.
...	optional arguments to be passed to barplot.

Details

Produces barplots of mean perfect match and mismatch intensities for slot data for an object of class DataTreeSet.

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

plotPM, 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.root",sep="/"))

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

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

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

---

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

xps.expr	object of class ExprTreeSet.
filename	file name of ROOT filter file.
filedir	system directory where ROOT filter file should be stored.
filter	object of class PreFilter.
minfilters	minimum number of initialized filter methods to satisfy (default is all filters).
logbase	convert data to logarithm of base: "0", "log", "log2" (default), "log10"
treename	tree name to be used in ROOT filter file.
xps.call	optional object of class CallTreeSet.

verbose	logical, if TRUE print status information.
object	object of class ExprTreeSet.
...	same arguments as function prefilter.

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.root",sep="/"))

## second, create an ExprTreeSet
data.rma <- rma(data.test3,"tmp_TestRMA",tmpdir="",background="pmonly",normalize=TRUE,verbose=FALSE)
## note: do not copy/paste this code, it is necessary only because R CMD check fails since it does not find tmp
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(10.5,80.0,"percent"))

## finally, create a FilterTreeSet
rma.pfr <- prefilter(data.rma,"tmp_Test3Prefilter",getwd(),prefltr,2,verbose=FALSE)
str(rma.pfr)

## End(Not run)
```

PreFilter-class *Class PreFilter*

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 contructor 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 madFilter.
cv	"character" vector describing parameters for cvFilter.
variance	"character" vector describing parameters for varFilter.
difference	"character" vector describing parameters for diffFilter.
ratio	"character" vector describing parameters for ratioFilter.
gap	"character" vector describing parameters for gapFilter.
lothreshold	"character" vector describing parameters for lowFilter.
hithreshold	"character" vector describing parameters for highFilter.
quantile	"character" vector describing parameters for quantileFilter.
prescall	"character" vector describing parameters for callFilter.

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)
```

---

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

object	object of class CallTreeSet.
treenames	character vector containing optional tree names to be used as subset.
value	data.frame 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.root",sep="/"))

## 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)
```

---

probeContentGC-methods

Get G/C Content for Probes

Description

Get G/C content for all or selected UNIT_IDs.

Usage

```
probeContentGC(object, which = "", unitID = NULL, unittype = "transcript")
```

Arguments

object	Object of class "SchemeTreeSet" or "DataTreeSet".
which	type of probes to be used, for details see validData.
unitID	optional vector of UNIT_IDs.
unittype	character vector, one of "transcript", "probeset".

Details

Function probeContentGC returns a data.frame containing columns "Mask" and "ContentGC" for all or selected the UNIT_ID(s).

For exon arrays the type of UNIT_ID(s) depends on unittype.

Value

A data.frame.

Author(s)

Christian Stratowa

See Also

probeSequence

Examples

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

## get UNIT_ID for probeset IDs
id <- probesetID2unitID(scheme.test3, c("PA1178_oprH_at","AFFX-Bt_eIF-4E_3_at","100084_at"))

## get GC content
gc <- probeContentGC(scheme.test3, unitID=id)
head(gc)

scheme.test3 <- removeProbeContentGC(scheme.test3)

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

---

probeSequence-methods *Get Probe Sequence*

Description

Get probe sequences for all or selected UNIT_IDs.

Usage

```
probeSequence(object, unitID = NULL)
```

Arguments

object	Object of class "SchemeTreeSet" or "DataTreeSet".
unitID	optional vector of UNIT_IDs.

Details

Function probeSequence returns a data.frame containing column "ProbeSequence" for all or selected the UNIT_ID(s).

Value

A data.frame.

Author(s)

Christian Stratowa

See Also

probeContentGC

Examples

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

## get UNIT_ID for probeset ID
id <- probesetID2unitID(scheme.test3, "100084_at")

## get GC content
seq <- probeSequence(scheme.test3, unitID=id)
head(seq)

scheme.test3 <- removeProbeSequence(scheme.test3)

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

---

probesetID2unitID-methods

Conversion between Probeset IDs and UnitIDs

Description

Convert probeset IDs and transcript IDs to internal UNIT_IDs and vice verse.

Usage

```
probesetID2unitID(object, probesetID = NULL, as.list = TRUE)
transcriptID2unitID(object, transcriptID = NULL, as.list = TRUE)
unitID2probesetID(object, unitID = NULL, as.list = TRUE)
unitID2transcriptID(object, unitID = NULL, as.list = TRUE)
```

Arguments

object	Object of class "SchemeTreeSet" or "DataTreeSet".
probesetID	optional vector of probeset IDs.
transcriptID	optional vector of transcript IDs.
unitID	optional vector of UNIT_IDs.
as.list	if TRUE a list will be returned (default is data.frame).

Details

Functions probesetID2unitID and transcriptID2unitID return the UNIT_ID(s) for all or selected probeset IDs or transcript IDs, respectively.

Conversely, functions unitID2probesetID and unitID2transcriptID return the probeset IDs or transcript IDs, respectively, for all or selected UNIT_IDs. . For expression arrays the functions for probeset IDs and transcript IDs return identical IDs. . For exon arrays the functions for probeset IDs and transcript IDs return the probeset_id(s) or transcript_cluster_id(s), respectivley.

By default a list is returned, however for as.list=FALSE a character vector of IDs is returned.

Value

A list or character vector.

Author(s)

Christian Stratowa

See Also

unitID2transcriptID, unitID2probesetID

Examples

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

## unitNames not attached
id <- unitID2probesetID(scheme.test3, c(2,34,229))
id

## unitNames attached
scheme.test3 <- attachUnitNames(scheme.test3)
id <- probesetID2unitID(scheme.test3, c("PA1178_oprH_at","AFFX-Bt_eIF-4E_3_at","100084_at"))
id
scheme.test3 <- removeUnitNames(scheme.test3)

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

probesetplot-methods *Plot of Probe Intensities for a Probeset.*

Description

Creates a line plot of probe intensities for a probeset.

Usage

```
probesetplot(x, unitID, unittype = "transcript", which = "pm",
```

Arguments

x	object of class DataTreeSet.
unitID	unit ID of probeset with type of ID determined by parameter unittype.
unittype	character vector, one of "unit", "transcript", "probeset".
which	type of probes to be used, for details see validData.
transfo	a valid function to transform the data, usually "log2", or "0".
names	optional vector of sample names.
ylim	range for the plotted y values.
col	color to use for the different samples.
lty	line types to use for the different samples.
add.legend	logical, if TRUE a legend of sample names will be drawn. Optionally, a character indicating the position of the legend, default is "topleft".
...	optional arguments to be passed to probesetplot.

Details

Produces line plots of the probe intensities for probeset unitID. Probe intensities are taken from slot data.

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, line plots of probe intensities will only be drawn for these columns.

Note

Data must first be attached to class DataTreeSet using method attachInten. Furthermore, unit names must be attached using method attachUnitNames.

Author(s)

Christian Stratowa

See Also

plotPM, 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.root",sep="/"))

## need to attach probe intensities and optionally unit names
data.test3 <- attachUnitNames(data.test3)
data.test3 <- attachInten(data.test3)

if (interactive()) {
probesetplot(data.test3, unitID="100084_at", unittype="transcript", add.legend=TRUE)
}

## optionally remove unit names and data to free memory
data.test3 <- removeInten(data.test3)
data.test3 <- removeUnitNames(data.test3)
```

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 subdirectoy 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 subdirectoy setname.

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

Extends

Class "TreeSet", directly.

Methods

attachData `signature(object = "ProcesSet")`: exports data from ROOT data file and and saves as data.frame `data`.

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`.

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

mboxplot `signature(x = "ProcesSet")`: creates an M-boxplot of the data from data.frame `data`.

removeData `signature(object = "ProcesSet")`: replaces data.frame `data` with an empty data.frame of dim(0,0).

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`.

treeData `signature(object = "ProcesSet")`: extracts all columns from data.frame `data`.

validData `signature(object = "ProcesSet")`: extracts a subset of columns from data.frame `data`.

Author(s)

Christian Stratowa

See Also

derived classes `DataTreeSet`, `ExprTreeSet`, `CallTreeSet`, `QualTreeSet`.

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, isxenograft,xenostrain,xenosex,xeno

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,isxenograf

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,isxenograft,xenostrain,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,phe 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,isxenograft,xenostrain,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,preparation,pro

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,concentrationunit,tim

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 testing","my comment
authorInfo(project)    <- c("Stratowa","Christian","Project Leader","Company","Dept","cstrato.at.aon.at","+
datasetInfo(project)   <- c("Test3Set","MC","Tissue","Stratowa","20060106","description","my comment")
sourceInfo(project)    <- c("Unknown","source type","Homo sapiens","caucasian","description","my comment")
primcellInfo(project)  <- c("Mel31","primary cell",20071123,"extracted from patient","male","my pheno","my
arrayInfo(project)      <- c("Test3","GeneChip","description","my comment")
hybridizInfo(project)  <- c(c("TestA1","hyb type","TestA1.CEL",20071117,"my prep1","standard protocol","A1"
                       c("TestA2","hyb type","TestA2.CEL",20071117,"my prep2","standard protocol","A2",1,"m
                       c("TestB1","hyb type","TestB1.CEL",20071117,"my prep1","standard protocol","B1",2,"m
                       c("TestB2","hyb type","TestB2.CEL",20071117,"my prep2","standard protocol","B2",2,"m
treatmentInfo(project) <- c(c("TestA1","DMSO",4.3,"mM",1.0,"hours","intravenous","my comment"),
```

```
                        c("TestA2","DMSO",4.3,"mM",8.0,"hours","intravenous","my comment"),
                        c("TestB1","DrugA2",4.3,"mM",1.0,"hours","intravenous","my comment"),
                        c("TestB2","DrugA2",4.3,"mM",8.0,"hours","intravenous","my comment"))
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,xenostrain,xenosex,x
celline:	character vector c(name,type,parent,atcc,modification,sex,phenotype, genotype,extraction,isxenogr
primarycell:	character vector c(name,type,date,description,sex,phenotype, genotype,extraction,isxenograft,xeno
tissue:	character vector c(name,type,development,morphology,disease,stage, donorage,ageunit,status,sex,p
biopsy:	character vector c(name,type,morphology,disease,stage,donorage,ageunit, status,sex,phenotype,ge
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,time

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 for testing","my commen
                  hybridizations=c(c("TestA1","hyb type","TestA1.CEL",20071117,"my prep1","standard protoco
                                   c("TestA2","hyb type","TestA2.CEL",20071117,"my prep2","standard protocol",
                                   c("TestB1","hyb type","TestB1.CEL",20071117,"my prep1","standard protocol",
                                   c("TestB2","hyb type","TestB2.CEL",20071117,"my prep2","standard protocol",
str(project)
```

```
## alternatively add character vectors as methods after creation of constructor
authorInfo(project)   <- c("Stratowa","Christian","Project Leader","Company","Dept","cstrato.at.aon.at","++
datasetInfo(project)  <- c("Test3Set","MC","Tissue","Stratowa","20060106","description","my comment")
treatmentInfo(project) <- c(c("TestA1","DMSO",4.3,"mM",1.0,"hours","intravenous","my comment"),
                      c("TestA2","DMSO",4.3,"mM",8.0,"hours","intravenous","my comment"),
                      c("TestB1","DrugA2",4.3,"mM",1.0,"hours","intravenous","my comment"),
                      c("TestB2","DrugA2",4.3,"mM",8.0,"hours","intravenous","my comment"))
str(project)
```

---

qualify *Probe Set Quality Control Functions*

Description

Converts Affymetrix probe level data to expression levels by fitting a multichip model.

Usage

```
qualify(xps.data,
        filename   = character(0),
        filedir    = getwd(),
        tmpdir     = "",
        update     = FALSE,
        select     = "none",
        method     = character(),
        option     = "transcript",
        logbase    = "log2",
        exonlevel  = "",
        params     = list(),
        xps.scheme = NULL,
        add.data   = TRUE,
        verbose    = TRUE)

qualify.rlm(xps.data,
            filename   = character(0),
            filedir    = getwd(),
            tmpdir     = "",
            update     = FALSE,
            option     = "transcript",
            exonlevel  = "",
            xps.scheme = NULL,
            add.data   = TRUE,
            verbose    = TRUE)

xpsQualify(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.
update	logical. If TRUE the existing ROOT data file filename will be updated.
select	type of probes to select for summarization.
method	qualification method to use, currently rlm.
option	option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon/genome arrays only.
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.
params	vector of parameters for summarization method.
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.
object	object of class DataTreeSet.
...	the arguments described above.

Details

Converts Affymetrix probe level data to expression levels by fitting a multichip model.

This function stores three types of ROOT trees in filename:
- quality trees containing expression levels, normalized unscaled standard errors (NUSE), relative log expressions (RLE)
- residual trees containing the residual SE and the model fit weights
- border trees containing the border intensities, mean border intensities and center of intensities (COI)

xpsQualify is the DataTreeSet method called by function qualify, containing the same parameters.

Value

An QualTreeSet.

Note

This function takes any DataTreeSet and computes expression levels by summarizing the probe set values into one expression measure. It does NOT do any further preprocessing such as background correction or (quantile) normalization. If you want to do background correction and/or normalization first then you need to use function fitQC.

Author(s)

Christian Stratowa

See Also

fitQC

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",sep="/"))

## compute RMA stepwise

## background correction
data.bg.rma <- bgcorrect.rma(data.test3,"tmp_Test3RMABgrd",filedir=getwd())

## normalize quantiles
data.qu.rma <- normalize.quantiles(data.bg.rma,"tmp_Test3RMANorm",filedir=getwd())

## summarize medianpolish
data.mp.rma <- summarize.rma(data.qu.rma,"tmp_Test3RMAExpr",filedir=getwd(),tmpdir="")

## qualification - rlm

## fit model on raw data
data.raw.rlm <- qualify.rlm(data.test3, "tmp_Test3RawQual", filedir=getwd(), tmpdir="", option="transcript",

## fit model on background adjusted data
data.adj.rlm <- qualify.rlm(data.bg.rma, "tmp_Test3AdjQual", filedir=getwd(), tmpdir="", option="transcript'

## fit model on normalized data
data.nrm.rlm <- qualify.rlm(data.qu.rma, "tmp_Test3NormQual", filedir=getwd(), tmpdir="", option="transcript

## get expression levels
expr.raw.rlm <- validData(data.raw.rlm)
expr.adj.rlm <- validData(data.adj.rlm)
expr.nrm.rlm <- validData(data.nrm.rlm)

## get borders
brd.raw <- borders(data.raw.rlm)
brd.adj <- borders(data.adj.rlm)

## get residuals
res.raw <- residuals(data.raw.rlm)
res.adj <- residuals(data.adj.rlm)

## get weights
w.raw <- weights(data.raw.rlm)
w.adj <- weights(data.adj.rlm)

## End(Not run)
```

QualTreeSet-class *Class QualTreeSet*

Description

This class provides the link to the ROOT quality control file and the ROOT trees contained therein. It extends class ProcesSet.

Objects from the Class

Objects are created using functions qualify, fitQC, or the specialized functions qualify.rlm, fitRLM or rmaPLM.

Slots

qualopt: Object of class "character" representing the quality control option, i.e. 'raw', 'adjusted', 'normalized' or 'all'.

qualtype: Object of class "character" representing the quality control type, i.e. 'rlm'.

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 subdirectoy 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 subdirectoy setname.

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

Extends

Class "ProcesSet", directly. Class "TreeSet", by class "ProcesSet", distance 2.

Methods

borderplot signature(x = "QualTreeSet"): creates a boxplot of positive and negative border elements.

borders signature(object = "QualTreeSet"): exports border trees from ROOT quality control file as data.frame data.

coiplot signature(x = "QualTreeSet"): creates a Center-of-Intensity-plot for positive and negative feature intensities.

image signature(x = "QualTreeSet"): creates a pseudo image for each quality control tree, i.e. residual images.

NUSE signature(x = "QualTreeSet"): plot Normalized Unscaled Standard Errors, or return stats, values.

nuseplot signature(x = "QualTreeSet"): creates a NUSE-plot.

qualOption signature(object = "QualTreeSet"): extracts slot qualopt.

qualOption<- signature(object = "QualTreeSet", value = "character"): replaces slot qualopt.

qualType signature(object = "QualTreeSet"): extracts slot qualtype.

qualType<- `signature(object = "QualTreeSet", value = "character")`: replaces slot `qualtype`.

residuals `signature(object = "QualTreeSet")`: exports residuals from the residuals trees of the ROOT quality control file as data.frame `data`.

RLE `signature(x = "QualTreeSet")`: plot Relative Log Expression, or return stats, values.

rleplot `signature(x = "QualTreeSet")`: creates a RLE-plot.

weights `signature(object = "QualTreeSet")`: exports weights from the residuals trees of the ROOT quality control file as data.frame `data`.

xpsRNAdeg `signature(x = "QualTreeSet")`: list with parameters for RNA degradation.

Author(s)

Christian Stratowa

See Also

related classes `DataTreeSet`, `CallTreeSet`, `ExprTreeSet`.

Examples

```
showClass("QualTreeSet")
```

---

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.root",sep="/"))

rawCELName(data.test3)
rawCELName(data.test3, treename = "TestA2.cel", fullpath = FALSE)
```

RLE-methods *Relative Log Expression (RLE)*

Description

Produce boxplots of Relative Log Expression (RLE) values for the set of arrays. Alternatively, summary statistics or RLE values can be extracted as data.frame.

Usage

```
RLE(x, treename = "*", type = c("plot", "stats", "values"), qualopt = NULL, ...)
```

Arguments

x	object of class QualTreeSet.
treename	vector of tree names to export.
type	type of output, plot, stats or values.
qualopt	quality control option, i.e. 'raw', 'adjusted', 'normalized' or 'all'.
...	optional arguments to be passed to rleplot.

Details

Create boxplots of Relative Log Expression (RLE) values for the set of arrays, i.e. of M plots, where M is determined relative to a pseudo-median reference chip.

Alternatively it is possible to extract either the summary statistics as data.frame (type="stats") or all RLE values as data.frame (type="values").

If an object of class QualTreeSet was created by fitting a probe level model with qualopt="all" then RLE will plot or extract RLE for "all" quality options. If you want to plot or extract RLE for a certain quality option only, e.g. "normalized" data only, then you can use parameter qualopt with qualopt="<qualopt>".

Author(s)

Christian Stratowa

See Also

plotRLE, rleplot

Examples

```
## Not run:
## 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.root",sep="/"))

## qualification - rlm
rlm.all <- rmaPLM(data.test3, "tmp_Test3RLMall", filedir=getwd(), tmpdir="", qualopt="all", option="transcri

## plot expression levels
if (interactive()) {
RLE(rlm.all)
```

```
RLE(rlm.all, qualopt="normalized")
qcRLE <- RLE(rlm.all, type="stats")
qcRLE <- RLE(rlm.all, type="values")
qcRLE <- RLE(rlm.all, treename="TestA1_normalized.rlm", type="stats")
qcRLE <- RLE(rlm.all, treename="TestA1_normalized.rlm", type="values")
}

## End(Not run)
```

`rleplot-methods` *Box Plots of Relative Log Expression (RLE)*

Description

Produce boxplots of Relative Log Expression (RLE) values for the set of arrays.

Usage

```
rleplot(x, which = "UnitName", size = 0, range = 0, names = "name
```

Arguments

`x`	object of class `ExprTreeSet` or `QualTreeSet`.
`which`	type of probes to be used, for details see `validData`.
`size`	length of sequence to be generated as subset.
`range`	determines how far the plot whiskers extend out from the box.
`names`	optional vector of sample names.
`main`	the main title for the plot.
`ylim`	range for the plotted y values.
`las`	the style of axis labels.
`add.line`	`logical`, if TRUE a horizontal line is drawn.
`outline`	if `outline` is not true, the outliers are not drawn.
`...`	optional arguments to be passed to `boxplot`.

Details

Create boxplots of Relative Log Expression (RLE) values for the set of arrays, i.e. of M plots, where M is determined relative to a pseudo-median 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.

If an object of class `QualTreeSet` was created by fitting a probe level model with `qualopt="all"` then `rleplot` will plot RLE for "all" quality options. If you want to plot RLE for a certain quality option only, e.g. "normalized" data only, then you can use parameter `names` with `names="namepart:<qualopt>"`, e.g. `names="namepart:normalized"`.

Author(s)

Christian Stratowa

See Also

RLE, plotRLE, mboxplot, nuseplot

Examples

```
# load existing ROOT scheme file and ROOT expression file for rma
scheme.test3 <- root.scheme(paste(path.package("xps"),"schemes/SchemeTest3.root",sep="/"))
data.rma <- root.expr(scheme.test3, paste(path.package("xps"),"rootdata/tmp_Test3RMA.root",sep="/"), "mdp")

if (interactive()) {
rleplot(data.rma)
}
```

---

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  = "",
    params     = list(16384, 0.0, 1.0, 10, 0.01, 1),
    xps.scheme = NULL,
    add.data   = TRUE,
    verbose    = TRUE)

xpsRMA(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.
background	probes used to compute background, one of 'pmonly', 'mmonly', 'both'; for genome/exon arrays one of 'genomic', 'antigenomic'
normalize	logical. If TRUE normalize data using quantile normalization.
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.
params	list of (default) parameters for rma.
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.
object	object of class DataTreeSet.
...	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_cluste
exon:	expression levels are computed for exon clusters, i.e. probe sets containing the same 'exon_id', where each
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 (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+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".

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

xpsRMA is the DataTreeSet 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.

Please note that the default settings of params gives results which are identical to the results obtained with APT (Affymetrix Power Tools) and with package affy_1.14.2 or earlier. If you want to obtain results which are identical to the results obtained with affy_1.16.0 or later then you need to set params = list(16384, 0.0, 0.4, 10, 0.01, 1).

By setting parameter background="none" it is possible to skip background correction .

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 seperate 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.root",sep="/"))

data.rma <- rma(data.test3,"tmp_Test3RMA",tmpdir="",background="pmonly",normalize=TRUE,verbose=FALSE)

## 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="antigenomic",
                  normalize=T,option="transcript",exonlevel="metacore")

## 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 TObject, 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.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` *Create class CallTreeSet accessing ROOT detection call file*

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 SchemeTreeSet 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.root",sep="/"))

## 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 *Create class DataTreeSet accessing ROOT data file*

Description

Create class DataTreeSet accessing ROOT data file.

Usage

```
root.data(xps.scheme, rootfile = character(0), celnames = "*")
```

Arguments

xps.scheme	A SchemeTreeSet 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"),"raw",sep="/"),verb

## use subset of CEL-files
subdata.test3 <- root.data(scheme.test3,"tmp_datatest3_cel.root", celnames=c("TestA1.cel","TestB2.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", save.as = "", w = 540
```

Arguments

x	object of class DataTreeSet or ExprTreeSet.
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.
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",sep="/"))

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 SchemeTreeSet 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.root",sep="/"))

# rma
all.rma <- rma(data.test3,"tmp_Test3RMAAll",tmpdir="",background="pmonly",normalize=TRUE,verbose=FALSE)

## 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 = "Graph1D", sa
```

Arguments

x	object of class DataTreeSet or ExprTreeSet.
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",sep="/"))

root.graph1D(data.test3, "TestA1.cel")

## End(Not run)
```

`root.graph2D` *ROOT 2D-Graph*

Description

Creates a ROOT 2D-graph for a ROOT tree.

Usage

```
root.graph2D(x, treename1 = character(0), treename2 = character(0),logbase = "log2", option = "P",
```

Arguments

`x`	object of class `DataTreeSet` or `ExprTreeSet`.
`treename1`	name of first tree, must be present in `rootfile` of object `x`.
`treename2`	name of second 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 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.mvaplot

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",sep="/"))

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 = "HIST", canvasnam
```

Arguments

x	object of class DataTreeSet or ExprTreeSet.
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.
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",sep="/"))

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 = "log2", option = "COLZ
```

Arguments

x	object of class DataTreeSet or ExprTreeSet.
treename1	name of first tree, must be present in rootfile of object x.
treename2	name of second 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 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",sep="/"))

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 = character(0), logba
```

Arguments

x	object of class DataTreeSet or ExprTreeSet.
treename1	name of first tree, must be present in rootfile of object x.
treename2	name of second tree, must be present in rootfile of object x.
treename3	name of third 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 hist TH3 option, usually one of "HIST", "SCAT", "BOX".
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 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",sep="/"))

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", option = "COLZ", zli
```

Arguments

x	object of class DataTreeSet.
treename	name of tree, must be present in rootfile 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",sep="/"))

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"),"raw",sep="/"),verb

## get subset of CEL-files
subdataA <- root.data(scheme.test3,"tmp_datatest3_cel.root", celnames=c("TestA1.cel","TestA2.cel"))
subdataB <- root.data(scheme.test3,"tmp_datatest3_cel.root", celnames=c("TestB1.cel","TestB2.cel"))

## merge data
dataAB <- root.merge.data(scheme.test3,c(rootFile(subdataA),rootFile(subdataB)), celnames=c("TestB1.cel","T
```

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 = "log2", option = "P",
```

Arguments

x	object of class ExprTreeSet or DataTreeSet.
treename1	name of first tree, must be present in rootfile of object x.
treename2	name of second 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", "*".
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.root",sep="/"))

# 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` *ROOT Profile Plot*

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, boxes = TRUE, yl
```

Arguments

`x`	S4 object, usually of class `DataTreeSet` or `ExprTreeSet`.
`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`, or for a subset of trees. 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",sep="/"))

root.profile(data.test3)

## End(Not run)
```

---

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 tpye, either 'GeneChip', 'GenomeChip' or 'ExonChip'.

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

unitname: Object of class "data.frame". The data.frame can contain the mapping between the internal UNIT_IDs and the UnitNames, i.e. the probeset IDs.

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.

probe: Object of class "data.frame". The data.frame can contain the probe info for the oligos as e.g. probe sequence, G/C content.

`setname`: Object of class `"character"` representing the name to the ROOT file subdirectoy 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 subdirectoy `setname`.

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

Extends

Class `"TreeSet"`, directly.

Methods

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

attachProbe `signature(object = "SchemeTreeSet")`: exports probe tree from ROOT scheme file and and saves `varlist` as data.frame `probe`.

attachProbeContentGC `signature(object = "SchemeTreeSet")`: exports probe tree from ROOT scheme file and and saves `fNumberGC` as data.frame `probe`.

attachProbeSequence `signature(object = "SchemeTreeSet")`: exports probe tree from ROOT scheme file and and saves `fSequence` as data.frame `probe`.

attachUnitNames `signature(object = "SchemeTreeSet")`: exports unit tree from ROOT scheme file and and saves as data.frame `unitname`.

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`.

chipProbe `signature(object = "SchemeTreeSet")`: extracts data.frame `probe`.

chipProbe<- `signature(object = "SchemeTreeSet", value = "data.frame")`: replaces data.frame `probe`.

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`.

probeContentGC `signature(object = "SchemeTreeSet")`: extracts all or selected GC contents from data.frame `probe`.

probeInfo `signature(object = "SchemeTreeSet")`: extracts slot `probeinfo`.

probeSequence `signature(object = "SchemeTreeSet")`: extracts all or selected probe sequences from data.frame `probe`.

probesetID2unitID `signature(object = "SchemeTreeSet")`: extracts all or selected probesetIDs from data.frame `unitname` with UnitName, i.e. probeset ID, as (row)names.

removeMask `signature(object = "SchemeTreeSet")`: replaces data.frame `mask` with an empty data.frame of dim(0,0).

removeProbe `signature(object = "SchemeTreeSet")`: replaces data.frame `probe` with an empty data.frame of dim(0,0).

removeProbeContentGC `signature(object = "SchemeTreeSet")`: replaces data.frame `probe` with an empty data.frame of dim(0,0).

removeProbeSequence `signature(object = "SchemeTreeSet")`: replaces data.frame `probe` with an empty data.frame of dim(0,0).

removeUnitNames `signature(object = "SchemeTreeSet")`: replaces data.frame `unitname` with an empty data.frame of dim(0,0).

symbol2unitID `signature(object = "SchemeTreeSet")`: extracts internal UNIT_ID(s) for one or more gene symbols.

transcriptID2unitID `signature(object = "SchemeTreeSet")`: extracts all or selected transcriptIDs from data.frame `unitname` with UnitName, i.e. transcript ID, as (row)names.

unitID2probesetID `signature(object = "SchemeTreeSet")`: extracts all or selected unitIDs from data.frame `unitname` with UNIT_ID as (row)names.

symbol2unitID `signature(object = "SchemeTreeSet")`: extracts gene symbols for one or more internal UNIT_ID(s).

unitID2transcriptID `signature(object = "SchemeTreeSet")`: extracts all or selected unitIDs from data.frame `unitname` with UNIT_ID as (row)names.

unitNames `signature(object = "SchemeTreeSet")`: extracts data.frame `unitname`.

unitNames<- `signature(object = "SchemeTreeSet", value = "data.frame")`: replaces data.frame `unitname`.

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

summarize.mas4(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE, c

summarize.mas5(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE, c

summarize.rma(xps.data, filename = character(0), filedir = getwd(), tmpdir = "", update = FALSE, op

xpsSummarize(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.
update	logical. If TRUE the existing ROOT data file filename will be updated.
select	type of probes to select for summarization.
method	summarization method to use.
option	option determining the grouping of probes for summarization, one of 'transcript', 'exon', 'probeset'; exon arrays only.
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.
params	vector of parameters for summarization method.
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.
object	object of class DataTreeSet.
...	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.root",sep="/"))

## RMA background
data.bg.rma <- bgcorrect.rma(data.test3,"tmp_Test3RMA",filedir=getwd(),tmpdir="",verbose=FALSE)
## normalize quantiles
data.qu.rma <- normalize.quantiles(data.bg.rma,"tmp_Test3RMA",filedir=getwd(),tmpdir="",update=TRUE,verbose
## summarize medianpolish
data.mp.rma <- summarize.rma(data.qu.rma,"tmp_Test3RMA",filedir=getwd(),tmpdir="",update=TRUE,verbose=FALSE

## 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]))
}
```

---

symbol2unitID-methods *Conversion between Gene Symbols and UnitIDs*

Description

Convert gene symbols to internal UNIT_IDs and vice verse.

Usage

```
symbol2unitID(object, symbol, unittype = "transcript", as.list = TRUE)
unitID2symbol(object, unitID, unittype = "transcript", as.list = TRUE)
```

Arguments

object	Object of class "SchemeTreeSet" or "DataTreeSet".
symbol	character vector of gene symbol(s).
unitID	vector of UNIT_IDs.
unittype	character vector, "transcript" or "probeset".
as.list	if TRUE a list will be returned (default is data.frame).

Details

Functions symbol2unitID and unitID2symbol returns the UNIT_ID(s) for selected gene symbols and vice verse.

For exon arrays the internal UNIT_ID(s) depend on unittype.

By default a list is returned, however for as.list=FALSE a character vector of IDs is returned.

Value

A `list` or character `vector`.

Author(s)

Christian Stratowa

See Also

`transcriptID2unitID`, `probesetID2unitID`

Examples

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

## unitnames not attached
id <- symbol2unitID(scheme.test3, symbol="ACTB", as.list=TRUE)
id
id <- unitID2symbol(scheme.test3, unitID=274, as.list=TRUE)
id

## unitnames attached
scheme.test3 <- attachUnitNames(scheme.test3)
id <- symbol2unitID(scheme.test3, symbol="ACTB", as.list=TRUE)
id
id <- unitID2symbol(scheme.test3, unitID=274, as.list=TRUE)
id
scheme.test3 <- removeUnitNames(scheme.test3)

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

---

treeInfo-methods *Get UserInfo from ROOT Trees*

Description

Extract the UserInfo from `ROOT` trees, i.e. quality control information.

Usage

```
treeInfo(object, treename = "*", treetype = character(0), varlist = "*",
```

Arguments

`object`	Object of class `"TreeSet"`.
`treename`	Object of class `"list"` representing the names of the `ROOT` trees.
`treetype`	type of tree to export, see `validTreetype`
`varlist`	names of tree leaves to export.
`qualopt`	option determining the data to which to apply qualification, one of 'raw', 'adjusted', 'normalized', 'all'.

Details

ROOT trees have a pointer to a list fUserInfo where it is possible to store data which do not fit into the usual tree structure. Taking advantage of this feature xps stores certain pre-processed results of the tree(s) in this list. For example, data trees store the minimal/maximal intensities and the number of oligos with minimal/maximal intensities of the CEL-files in list fUserInfo, while call trees store the number and percentage of P/M/A calls.

Function `treeInfo` allows to export this user information as a `data.frame`, whereby the parameters of `varlist` depend on the `treetype`:

Parameters for data trees with extensions `"cel"`, `"int"`, and background trees:
`fMinInten`: minimal intensity.
`fMaxInten`: maximal intensity.
`fNMinInten`: number of probes with minimal intensity.
`fNMaxInten`: number of probes with maximal intensity.
`fMaxNPixels`: maximal number of pixels.
`fNQuantiles`: number of precalculated quantiles.
`fQuantiles`: quantiles.
`fIntenQuant`: intensities at quantiles.

Parameters for expression trees:
`fNUnits`: number of units, i.e. probesets.
`fMinLevel`: minimal expression level.
`fMaxLevel`: maximal expression level.
`fNQuantiles`: number of precalculated quantiles.
`fQuantiles`: quantiles.
`fLevelQuant`: expression levels at quantiles.

Parameters for call trees:
`fNUnits`: number of units, i.e. probesets.
`fNAbsent`: number of units with absent call.
`fNMarginal`: number of units with marginal call.
`fNPresent`: number of units with present call.
`fPcAbsent`: percentage of units with absent call.
`fPcMarginal`: percentage of units with marginal call.
`fPcPresent`: percentage of units with present call.
`fMinPValue`: minimal p-value.
`fMaxPValue`: maximal p-value.

Parameters for border trees with extension `"brd"`:
`fMeanLeft`: mean intensity of left border.
`fMeanRight`: mean intensity of right border.
`fMeanTop`: mean intensity of top border.
`fMeanBottom`: mean intensity of bottom border.
`fCOIXhi`: x-location of COI for the positive elements.
`fCOIYhi`: y-location of COI for the positive elements.
`fCOIXlo`: x-location of COI for the negative elements.
`fCOIYlo`: y-location of COI for the negative elements.

Parameters for quality trees with extension `"rlm"`:
`fNUnits`: number of units, i.e. probesets.
`fMinLevel`: minimal expression level.
`fMaxLevel`: maximal expression level.
`fNQuantiles`: number of precalculated quantiles.
`fQuantiles`: quantiles.
`fLevelQuant`: expression levels at quantiles.

fNUSEQuant: NUSE at quantiles.
fRLEQuant: RLE at quantiles.
fQualOption: value of qualopt.

Parameters for residual trees with extension "res":
fNQuantiles: number of precalculated quantiles.
fQuantiles: quantiles.
fResiduQuant: residual at quantiles.
fWeightQuant: weight at quantiles.
fQualOption: value of qualopt.

Value

A data.frame.

Note

Taking advantage of function treeInfo plotting methods boxplot, callplot, coiplot, nuseplot and rleplot are able to display their results much faster, which is especially useful for large datasets.

Author(s)

Christian Stratowa

See Also

validTreetype

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.root",sep="/"))

userinfo <- treeInfo(data.test3, treetype="cel", varlist="*")
userinfo

userinfo <- treeInfo(data.test3, treename="TestB1", treetype="cel", varlist = "fNQuantiles:fIntenQuant")
userinfo

## Not run:
userinfo <- treeInfo(rlm.all, treetype="rlm", varlist = "fNQuantiles:fNUSEQuant:fRLEQuant", qualopt = "raw")
userinfo

userinfo <- treeInfo(rlm.all, treetype="brd")
userinfo

userinfo <- treeInfo(rlm.all, treetype="res", qualopt = "raw")
userinfo

userinfo <- treeInfo(rlm.all, treetype="res", varlist = "fResiduQuant", qualopt = "raw")
userinfo

## End(Not run)
```

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 subdirectoy `setname`.

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

treeInfo `signature(object = "TreeSet")`: extracts UserInfo from ROOT trees.

treeNames `signature(object = "TreeSet")`: extracts slot `treenames`.

Author(s)

Christian Stratowa

See Also

derived classes `SchemeTreeSet`, `DataTreeSet`, `ExprTreeSet`, `CallTreeSet`.

Examples

```
showClass("TreeSet")
```

`trma` *transposed Robust Multi-Array Average Expression Measure*

Description

This function converts a `DataTreeSet` into an `ExprTreeSet` using the transposed robust multi-array average (RMA) expression measure.

Usage

```
trma(xps.data,
     filename   = character(0),
     filedir    = getwd(),
     tmpdir     = "",
     background = "pmonly",
     normalize  = TRUE,
     option     = "transcript",
     exonlevel  = "",
     params     = list(16384, 0.0, 1.0, 10, 0.01, 2),
     xps.scheme = NULL,
     add.data   = TRUE,
     verbose    = TRUE)
```

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.
`background`	probes used to compute background, one of 'pmonly', 'mmonly', 'both'; for genome/exon arrays one of 'genomic', 'antigenomic'
`normalize`	logical. If `TRUE` normalize data using quantile normalization.
`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.
`params`	list of (default) parameters for `rma`.
`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 tRMA (transposed Robust Multichip Average) expression measure described in Giorgi et al. for both expression arrays and exon arrays.

To use method xpsRMA or function express to compute trma you need to set params = list(16384, 0.0, 1.0, 10, 0.

For further details please see rma

Value

An ExprTreeSet

Author(s)

Christian Stratowa

References

Federico M. Giorgi, Anthony M. Bolger, Marc Lohse and Bjoern Usadel (2010), Algorithm-driven Artifacts in median polish summarization of Microarray data. BMC Bioinformatics 11:553

See Also

rma, xpsRMA, 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.root",sep="/"))

data.trma <- trma(data.test3,"tmp_Test3tRMA",tmpdir="",background="pmonly",normalize=TRUE,verbose=FALSE)

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

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

---

type2Exten *Convert Method Type to Tree Extension*

Description

Convert Method Type to Tree Extension.

Usage

```
type2Exten(type, datatype)
```

Arguments

type	method type.
datatype	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:

mean:	trimmed mean
median:	median
quantile:	quantile
tukeybiweight:	tukey biweight
medianpolish:	median polish

For datatype 'normation' the following methods can be applied:

mean:	trimmed mean
median:	median
quantile:	quantile
lowess:	lowess
supsmu:	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 *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

xps.expr	object of class ExprTreeSet.
filename	file name of ROOT filter file.
filedir	system directory where ROOT filter file should be stored.
filter	object of class UniFilter.
minfilters	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 xps.expr to one of two groups.
treename	tree name to be used in ROOT filter file.
xps.fltr	optional object of class FilterTreeSet.
xps.call	optional object of class CallTreeSet.
update	logical. If TRUE the existing ROOT filter file filename will be updated.
verbose	logical, if TRUE print status information.
object	object of class ExprTreeSet.
...	same arguments as function unifilter.

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.root",sep="/"))

## second, create an ExprTreeSet
data.rma <- rma(data.test3,"tmp_Test3_RMA",tmpdir="",background="pmonly",normalize=TRUE,verbose=FALSE)
## note: do not copy/paste this code, it is necessary only because R CMD check fails since it does not find tmp
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),foldchange=c(1.3,"both"),

## finally, create an AnalysisTreeSet
rma.ufr <- unifilter(data.rma,"tmp_Test3Unifilter",getwd(),unifltr,group=c("GrpA","GrpA","GrpB","GrpB"),ver
str(rma.ufr)

## End(Not run)
```

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 contructor 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)
```

uniTest-methods *A Two-Group Unitest*

Description

Unitest performs a a two group uni-test such as the `t.test` on each row of the expression dataframe. The Unitest returns a dataframe containing the results of the test.

Usage

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

Arguments

`object`	object of class `UniFilter`.
`value`	character vector `c(type, alternative, correction, numperm, mu, paired, conflevel, vare`

Details

The method `uniTest` initializes the following parameters:

`type`:	a character string specifying the type of test: currently `"t.test"` (default) or `"normal.test"`.
`alternative`:	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"great`
`correction`:	a correction to adjust p-values for multiple comparisons: `correction="none"`: no correction (default). `correction="bonferroni"`: Bonferroni correction. `correction="BH" or "fdr"`: correction for false discovery rate (Benjamini & Hochberg). `correction="BY"`: correction for false discovery rate (Benjamini & Yekutieli). `correction="hochberg"`: Hochberg correction. `correction="holm"`: Holm correction. `correction="wy"`: Westfall-Young step-down adjusted p-chance (E.Manduchi).
`numperm`:	optional number of permutations used to determine p-chance (default is 0).
`mu`:	a number indicating the true value of the difference in means for a two sample test (default is 0).
`paired`:	a logical indicating whether you want a paired uni-test (default is FALSE).
`conflevel`:	confidence level of the interval (default is 0.95).
`varequ`:	a logical variable indicating whether to treat the two variances as being equal. If `TRUE` then the pooled

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)
```

`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

`object`	object of class `UniFilter`.
`value`	character vector `c(cutoff, variable)`.

Details

The method `unitestFilter` initializes the following parameters:

`cutoff`:	the cutoff level for the filter.
`variable`:	`variable="pval"` (default): p-value. `variable="stat"`: univariate statistic. `variable="padj"`: optional adjusted p-value. `variable="pcha"`: 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)
```

---

validCall-methods *Get Valid Detection Call Values*

Description

Extracts valid present call values with unit names as row names.

Usage

```
validCall(object, which = "UnitName")
validPVal(object, which = "UnitName")
```

Arguments

object	object of class CallTreeSet.
which	name of column containing unit name.

Details

Method validCall returns the present calls from slot detcall as data.frame and uses column which as row names, usually the probeset IDs stored in column "UnitName".

Method validPVal returns the detection call p-values from slot data as data.frame and uses column which as row names, usually the probeset IDs stored in column "UnitName".

Value

A data.frame.

Author(s)

Christian Stratowa

See Also

validData, validExpr

validData-methods *Extract Subset of Data*

Description

Extracts a subset of valid data from data.frame data.

Usage

```
validData(object, which = "", unitID = NULL, unittype = "transcript")
```

Arguments

object	object of class DataTreeSet, ExprTreeSet or CallTreeSet.
which	type of probes to be returned for DataTreeSet, otherwise name of column containing unit name.
unitID	optional vector of UNIT_IDs.
unittype	character vector, "transcript" or "probeset".

Details

For class DataTreeSet and 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 class DataTreeSet and 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 AFFX controls.
all:	combination of above.
genomic:	genomic background probes.
antigenomic:	antigenomic background probes.

For class ExprTreeSet validData returns the valid expression levels from slot data with unit names as row names, usually the probeset IDs stored in column which="UnitName".

For class CallTreeSet validData returns the valid detection call p-values from slot data with unit names as row names, usually the probeset IDs stored in column which="UnitName".

Value

A data.frame.

Author(s)

Christian Stratowa

See Also

pm, mm, validExpr, validCall

---

validExpr-methods *Get Valid Expression Levels*

Description

Extracts valid expression levels with unit names as row names from data.frame data.

Usage

```
validExpr(object, which = "UnitName")
```

Arguments

object	object of class ExprTreeSet.
which	name of column containing unit name.

Details

Method validExpr returns the expression levels from slot data and uses column which as row names, usually the probeset IDs stored in column "UnitName".

Value

A data.frame.

Author(s)

Christian Stratowa

See Also

validData, validCall

validSE-methods *Get Valid Standard Errors*

Description

Extracts valid standard errors with unit names as row names.

Usage

```
validSE(object, which = "UnitName")
```

Arguments

object	object of class ExprTreeSet.
which	name of column containing unit name.

Details

Method validSE returns the standard errors (or standard deviations) from the expression trees and uses column which as row names, usually the probeset IDs stored in column "UnitName".

Value

A data.frame.

Author(s)

Christian Stratowa

See Also

validExpr

validTreetype *Validate Tree Type*

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'.
`rlm`: quality tree containing expression levels, NUSE, RLE computed with 'median polish'.
`res`: residual tree containing the residual SE and the model fit weights.
`brd`: border tree containing border intensities, mean border intensities and COI.

Trees with `datatype="normation"` 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
```

Arguments

`x`	object of class `AnalysisTreeSet`.
`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

`xpsOptions` *xps Options*

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

`xpsQAReport` *Create Quality Assessment Report.*

Description

Create a quality assessment report.

Usage

```
xpsQAReport(xps.data,
            xps.expr    = NULL,
            xps.call    = NULL,
            xps.qual    = NULL,
            dataset     = character(0),
            title       = "Quality Report",
            date        = "October, 2011",
            author      = "Christian Stratowa",
            outdir      = file.path(getwd(), "QAReport"),
            add.pseudo  = FALSE,
            overwrite   = FALSE,
            verbose     = TRUE,
            ...)
```

Arguments

xps.data	object of class DataTreeSet.
xps.expr	object of class ExprTreeSet.
xps.call	object of class CallTreeSet.
xps.qual	object of class QualTreeSet.
dataset	name of the dataset.
title	title of quality report.
date	date of quality report.
author	author(s) of quality report.
outdir	name of directory where to create the quality report.
add.pseudo	logical, if TRUE add pseudo-images to the quality report.
overwrite	logical, if TRUE overwrite outdir and its contents.
verbose	logical, if TRUE print status information.
...	optional arguments to be passed to xpsQAReport.

Details

Function xpsQAReport creates a quality assessment report "QAReport.pdf" for all TreeSets, which are passed as parameters to the function. It calls library(tools) and uses its function buildVignettes to create the report.

If parameter xps.qual is supplied, it is possible to create pseudo-images for every CEL-file by setting parameter add.pseudo=TRUE.

Value

None, the output is a pdf-file.

Note

Function xpsQAReport requires a working LaTeX implementation and so will only work on Windows platforms, and on OS X, if the user has installed the necessary LaTeX tools.

Author(s)

Christian Stratowa, based on ideas of package affyQCReport.

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",sep="/"))

 ## optional normalized expression levels
 data.rma <- rma(data.test3,"Test3RMA",tmpdir="",background="pmonly",normalize=TRUE,verbose=FALSE)

 ## optional MAS5 detection call
 call.mas5 <- mas5.call(data.test3,"Test3Call",tmpdir="",verbose=FALSE)

 ## optional quality measures
```

```
rlm.all <- rmaPLM(data.test3, "tmp_Test3RLMall", filedir=getwd(), tmpdir="", qualopt="all", option="transcr

## quality assessment report
xpsQAReport(data.test3, data.rma, call.mas5, rlm.all, dataset="My Dataset", add.pseudo=TRUE, overwrite=TRUE

## End(Not run)
```

Index