Package ‘flowCore’

April 25, 2017

Title flowCore: Basic structures for flow cytometry data

Version 1.42.0

Author B. Ellis, P. Haaland, F. Hahne, N. Le Meur, N. Gopalakrishnan, J. Spidlen, M. Jiang

Description Provides S4 data structures and basic functions to deal with flow cytometry data.

Maintainer M.Jiang <wjiang2@fhcrc.org>

Depends R (>= 2.10.0)

Imports Biobase, BiocGenerics (>= 0.1.14), graph, graphics, methods, rrcov, stats, utils, stats4, corpcor, Rcpp, matrixStats

Suggests Rgraphviz, flowViz, flowStats, testthat, flowWorkspace, flowWorkspaceData, openCyto, knitr, ggcyto, gridExtra

Collate AllGenerics.R AllClasses.R flowFrame-accessors.R
flowSet-accessors.R Rm-methods.R actionItem-accessors.R
coerce.R logicalFilterResult-accessors.R
summarizeFilter-methods.R filterSummary-accessors.R
fcReference-accessors.R manyFilterResult-accessors.R
transformList-accessors.R identifier-methods.R
workFlow-accessors.R filterResultList-accessors.R IO.R
show-methods.R length-methods.R names-methods.R
RcppExports.R

License Artistic-2.0

biocViews Infrastructure, FlowCytometry, CellBasedAssays

LinkingTo Rcpp, BH(>= 1.62.0-1)

VignetteBuilder knitr

SystemRequirements GNU make, C++11

RoxygenNote 5.0.1

NeedsCompilation yes
R topics documented:

- `flowCore-package` .................................................. 4
- `actionItem-class` .................................................. 4
- `arcsinhTransform` ............................................... 6
- `asinht-class` .................................................... 7
- `asinhtGml2-class` .............................................. 8
- `biexponentialTransform` ...................................... 9
- `boundaryFilter-class` ....................................... 11
- `characterOrTransformation-class` ......................... 13
- `checkOffset` .................................................... 13
- `coerce` .......................................................... 14
- `compensateActionItem-class` ................................ 14
- `compensatedParameter-class` ................................ 16
- `compensateView-class` ..................................... 17
- `compensation-class` .......................................... 18
- `concreteFilter-class` ...................................... 21
- `dg1polynomial-class` ..................................... 22
- `each_col` ......................................................... 23
- `EHtrans-class` .................................................. 24
- `ellipsoidGate-class` ......................................... 25
- `estimateMedianLogicle` ..................................... 27
- `exponential-class` ........................................... 27
- `expressionFilter-class` ..................................... 29
- `fcReference-class` ........................................... 31
- `FCSTransTransform` ........................................ 34
- `filter` .......................................................... 35
- `filter-and-methods` ....................................... 37
- `filter-class` .................................................... 37
- `filter-in-methods` ........................................... 38
- `filterDetails-methods` ....................................... 39
- `filterList-class` ............................................ 39
- `filterReference-class` ..................................... 40
- `filterResult-class` .......................................... 41
- `filterResultList-class` .................................... 42
- `filters-class` .................................................. 43
- `filterSet-class` ................................................ 44
- `filterSummary-class` ....................................... 45
- `filterSummaryList-class` .................................. 47
- `flowFrame-class` ............................................. 48
- `flowSet-class` ................................................ 54
- `fsApply` ........................................................ 59
- `gateActionItem-class` ..................................... 60
- `gateView-class` ................................................ 62
- `getChannelMarker` ........................................... 64
- `getIndexSort` ................................................... 64
- `GvHD` ........................................................... 65
- `hyperlog-class` ............................................... 66
- `hyperlogtGml2-class` ...................................... 67
- `identifier` ........................................................ 69
- `inverseLogicleTransform` .................................. 70
- `invsplitscale-class` ......................................... 71
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>keyword-methods</td>
</tr>
<tr>
<td>kmeansFilter-class</td>
</tr>
<tr>
<td>linearTransform</td>
</tr>
<tr>
<td>lnTransform</td>
</tr>
<tr>
<td>logarithm-class</td>
</tr>
<tr>
<td>logicalFilterResult-class</td>
</tr>
<tr>
<td>logicleTransform</td>
</tr>
<tr>
<td>logtGml2-class</td>
</tr>
<tr>
<td>logTransform</td>
</tr>
<tr>
<td>manyFilterResult-class</td>
</tr>
<tr>
<td>markernames</td>
</tr>
<tr>
<td>multipleFilterResult-class</td>
</tr>
<tr>
<td>norm2Filter-class</td>
</tr>
<tr>
<td>normalization-class</td>
</tr>
<tr>
<td>normalizeActionItem-class</td>
</tr>
<tr>
<td>normalizeView-class</td>
</tr>
<tr>
<td>parameterFilter-class</td>
</tr>
<tr>
<td>parameters</td>
</tr>
<tr>
<td>parameters-class</td>
</tr>
<tr>
<td>parameterTransform-class</td>
</tr>
<tr>
<td>polygonGate-class</td>
</tr>
<tr>
<td>polytopeGate-class</td>
</tr>
<tr>
<td>quadGate-class</td>
</tr>
<tr>
<td>quadratic-class</td>
</tr>
<tr>
<td>quadraticTransform</td>
</tr>
<tr>
<td>randomFilterResult-class</td>
</tr>
<tr>
<td>ratio-class</td>
</tr>
<tr>
<td>ratiotGml2-class</td>
</tr>
<tr>
<td>read.FCS</td>
</tr>
<tr>
<td>read.FCSHeader</td>
</tr>
<tr>
<td>read.flowSet</td>
</tr>
<tr>
<td>rectangleGate-class</td>
</tr>
<tr>
<td>sampleFilter-class</td>
</tr>
<tr>
<td>scaleTransform</td>
</tr>
<tr>
<td>setOperationFilter-class</td>
</tr>
<tr>
<td>singleParameterTransform-class</td>
</tr>
<tr>
<td>sinht-class</td>
</tr>
<tr>
<td>spillover</td>
</tr>
<tr>
<td>split-methods</td>
</tr>
<tr>
<td>splitscale-class</td>
</tr>
<tr>
<td>splitScaleTransform</td>
</tr>
<tr>
<td>squareroot-class</td>
</tr>
<tr>
<td>Subset</td>
</tr>
<tr>
<td>subsetting-class</td>
</tr>
<tr>
<td>subsettingActionItem-class</td>
</tr>
<tr>
<td>subsettingView-class</td>
</tr>
<tr>
<td>summarizeFilter-methods</td>
</tr>
<tr>
<td>timeFilter-class</td>
</tr>
<tr>
<td>transform-class</td>
</tr>
<tr>
<td>transformActionItem-class</td>
</tr>
</tbody>
</table>
**flowCore-package**

Provides S4 data structures and basic infrastructure and functions to deal with flow cytometry data.

**Description**

Define important flow cytometry data classes: `flowFrame`, `flowSet` and their accessors.

Provide important transformation, filter, gating, workflow, and summary functions for flow cytometry data analysis.

Most of flow cytometry related Bioconductor packages (such as `flowStats`, `flowFP`, `flowQ`, `flowViz`, `flowMerge`, `flowClust`) are heavily depended on this package.

**Details**

- **Package:** flowCore
- **Type:** Package
- **Version:** 1.11.20
- **Date:** 2009-09-16
- **License:** Artistic-2.0

**Author(s)**

Maintainer: Florian Hahne <fhahne@fhcrc.org>

Authors: B. Ellis, P. Haaland, F. Hahne, N. Le Meur, N. Gopalakrishnan

**Index**

- `actionItem-class`  
  Class “actionItem”
actionItem-class

Description

Class and method to capture standard operations in a flow cytometry workflow.

Details

actionItems provide a means to bind standard operations on flow cytometry data in a workflow. Usually, the user doesn’t have to create these objects, instead they will be automatically created when applying one of the standard operations (gating, transformation, compensation) to a workflow object. Each actionItem creates one or several new views, which again can be the basis to apply further operations. One can conceptualize actionItems being the edges in the workflow tree connecting views, which are the nodes of the tree. There are more specific subclasses for the three possible types of operation: gateActionItem for gating operations, transformActionItem for transformations, and compensateActionItem for compensation operations. See their documentation for details.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Object of class &quot;character&quot;. A unique identifier for the actionItem.</td>
</tr>
<tr>
<td>name</td>
<td>Object of class &quot;character&quot;. A more human-readable name</td>
</tr>
<tr>
<td>parentView</td>
<td>Object of class &quot;fcViewReference&quot;. A reference to the parent view the actionItem is applied on.</td>
</tr>
<tr>
<td>alias</td>
<td>Object of class &quot;fcAliasReference&quot;. A reference to the alias table.</td>
</tr>
<tr>
<td>env</td>
<td>Object of class &quot;environment&quot;. The evaluation environment in the workFlow.</td>
</tr>
</tbody>
</table>

Methods

- **identifier** signature(object = "actionItem"): Accessor for the ID slot.
- **names** signature(x = "actionItem"): Accessor for the name slot.
- **parent** signature(object = "actionItem"): Accessor for the parentView slot. Note that the reference is resolved, i.e., the view object is returned.
- **alias** signature(object = "actionItem"): Get the alias table from a actionItem.
- **Rm** signature(symbol = "actionItem", envir = "workFlow", subSymbol = "character"): Remove a actionItem from a workFlow. This method is recursive and will also remove all dependent views and actionItems.

Author(s)

Florian Hahne

See Also

workFlow, gateActionItem, transformActionItem, compensateActionItem, view

Examples

showClass("view")
arcsinhTransform  

Create the definition of an arcsinh transformation function (base specified by user) to be applied on a data set

Description

Create the definition of the arcsinh Transformation that will be applied on some parameter via the transform method. The definition of this function is currently x<-asinh(a+b*x)+c). The transformation would normally be used to convert to a linear valued parameter to the natural logarithm scale. By default a and b are both equal to 1 and c to 0.

Usage

arcsinhTransform(transformationId="defaultArcsinhTransform", a=1, b=1, c=0)

Arguments

transformationId
  character string to identify the transformation

  a
  positive double that corresponds to the base of the logarithm.

  b
  positive double that corresponds to a scale factor.

  c
  positive double that corresponds to a scale factor

Value

Returns an object of class transform.

Author(s)

B. Ellis

See Also

transform-class, transform, asinh

Examples

samp <- read.FCS(system.file("extdata",  
  "0877408774.B08", package="flowCore"))
asinhTrans <- arcsinhTransform(transformationId="ln-transformation", a=1, b=1, c=1)  
translist <- transformList("FSC-H", asinhTrans)  
dataTransform <- transform(samp, translist)
**asinht-class**

Class "asinht"

**Description**

Inverse hyperbolic sin transformation is defined by the function

\[ f(\text{parameter}, a, b) = \sinh^{-1}(a \ast \text{parameter}) \ast b \]

**Objects from the Class**

Objects can be created by calls to the constructor `asinht(parameter,a,b,transformationId)`

**Slots**

- `.Data`: Object of class "function"
- `a`: Object of class "numeric" - non zero constant
- `b`: Object of class "numeric" - non zero constant
- `parameters`: Object of class "transformation" - flow parameter to be transformed
- `transformationId`: Object of class "character" - unique ID to reference the transformation

**Extends**


**Methods**

No methods defined with class "asinht" in the signature.

**Note**

The inverse hyperbolic sin transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

**Author(s)**

Gopalakrishnan N, F.Hahne

**References**

Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry V 1.5

**See Also**

sinht
Examples

dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
asinh1<asinht(parameters="FSC-H",a=2,b=1,transformationId="asinH1")
transOut<-eval(asinh1)(exprs(dat))

asinhtGml2-class

Class asinhtGml2

Description

Inverse hyperbolic sin transformation as parameterized in Gating-ML 2.0 is defined by the following function:

\[
\text{bound}(f, \text{boundMin}, \text{boundMax}) = \max(\min(f, \text{boundMax}), \text{boundMin})
\]

where

\[
f(\text{parameter}, T, M, A) = \left( \frac{\text{asinh}(\text{parameter} \times \sinh(M \times \ln(10)))}{T} + A \times \ln(10) \right) / ((M + A) \times \ln(10))
\]

This transformation is equivalent to Logicle(T, 0, M, A) (i.e., with W=0). It provides an inverse hyperbolic sine transformation that maps a data value onto the interval \([0,1]\) such that:

- The top of scale value (i.e., \(T\)) is mapped to 1.
- Large data values are mapped to locations similar to an \((M + A)\)-decade logarithmic scale.
- A decades of negative data are brought on scale.

In addition, if a boundary is defined by the boundMin and/or boundMax parameters, then the result of this transformation is restricted to the \([\text{boundMin}, \text{boundMax}]\) interval. Specifically, should the result of the \(f\) function be less than boundMin, then let the result of this transformation be boundMin. Analogically, should the result of the \(f\) function be more than boundMax, then let the result of this transformation be boundMax. The boundMin parameter shall not be greater than the boundMax parameter.

Objects from the Class

Objects can be created by calls to the constructor

asinhtGml2(\text{parameter}, \text{T}, \text{M}, \text{A}, \text{transformationId}, \text{boundMin}, \text{boundMax})

Slots

- \(\text{Data}\): Object of class function
- \(\text{T}\): Object of class numeric - positive constant (top of scale value)
- \(\text{M}\): Object of class numeric - positive constant (desired number of decades)
- \(\text{A}\): Object of class numeric - non-negative constant that is less than or equal to \(M\) (desired number of additional negative decades)
- \(\text{parameters}\): Object of class "transformation" - flow parameter to be transformed
- \(\text{transformationId}\): Object of class "character" - unique ID to reference the transformation
- \(\text{boundMin}\): Object of class numeric - lower bound of the transformation, default -Inf
- \(\text{boundMax}\): Object of class numeric - upper bound of the transformation, default Inf
biexponentialTransform

## Extends


## Methods

No methods defined with class `asinhtGml2` in the signature.

## Note

The inverse hyperbolic sin transformation object can be evaluated using the `eval` method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

## Author(s)

Spidlen, J.

## References


## See Also

`asinht`, `transform-class`, `transform`

## Examples

```r
myDataIn <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))
myASinH1 <- asinhtGml2(parameters = "FSC-H", T = 1000, M = 4.5, A = 0, transformationId="myASinH1")
transOut <- eval(myASinH1)(exprs(myDataIn))
```

---

### biexponentialTransform

*Compute a transform using the 'biexponential' function*

## Description

The ‘biexponential’ is an over-parameterized inverse of the hyperbolic sine. The function to be inverted takes the form \( \text{biexp}(x) = a \cdot \exp(b \cdot (x-w)) - c \cdot \exp(-d \cdot (x-w)) + f \) with default parameters selected to correspond to the hyperbolic sine.

## Usage

```r
biexponentialTransform(transformationId="defaultBiexponentialTransform", a = 0.5, b = 1, c = 0.5, d = 1, f = 0, w = 0, tol = .Machine$double.eps^0.25, maxit = as.integer(5000))
```
biexponentialTransform

Arguments

transformationId
A name to assign to the transformation. Used by the transform/filter integration routines.

a
See the function description above. Defaults to 0.5

b
See the function description above. Defaults to 1.0

c
See the function description above. Defaults to 0.5 (the same as a)

d
See the function description above. Defaults to 1 (the same as b)

f
A constant bias for the intercept. Defaults to 0.

w
A constant bias for the 0 point of the data. Defaults to 0.

tol
A tolerance to pass to the inversion routine (uniroot usually)

maxit
A maximum number of iterations to use, also passed to uniroot

Value

Returns values giving the inverse of the biexponential within a certain tolerance. This function should be used with care as numerical inversion routines often have problems with the inversion process due to the large range of values that are essentially 0. Do not be surprised if you end up with population splitting about w and other odd artifacts.

Author(s)
B. Ellis, N Gopalakrishnan

See Also
transform

Examples

# Construct some "flow-like" data which tends to be heteroscedastic.
data(GvHD)
biexp <- biexponentialTransform("myTransform")
after.1 <- transform(GvHD, transformList('FSC-H', biexp))
biexp <- biexponentialTransform("myTransform",w=10)
after.2 <- transform(GvHD, transformList('FSC-H', biexp))
	opar = par(mfcol=c(3, 1))
plot(density(exprs(GvHD[[1]])[, 1]), main="Original")
plot(density(exprs(after.1[[1]])[, 1]), main="Standard Transform")
plot(density(exprs(after.2[[1]])[, 1]), main="Shifted Zero Point")
boundaryFilter-class

Class "boundaryFilter"

Description

Class and constructor for data-driven filter objects that discard margin events.

Usage

boundaryFilter(x, tolerance=.Machine$double.eps, side=c("both", "lower", "upper"), filterId="defaultBoundaryFilter")

Arguments

x    Character giving the name(s) of the measurement parameter(s) on which the filter is supposed to work. Note that all events on the margins of any of the channels provided by x will be discarded, which is often not desired. Such events may not convey much information in the particular channel on which their value falls on the margin, however they may well be informative in other channels.
tolerance    Numeric vector, used to set the tolerance slot of the object. Can be set separately for each element in x. R’s recycling rules apply.
side    Character vector, used to set the side slot of the object. Can be set separately for each element in x. R’s recycling rules apply.
filterId    An optional parameter that sets the filterId slot of this filter. The object can later be identified by this name.

Details

Flow cytometry instruments usually operate on a given data range, and the limits of this range are stored as keywords in the FSC files. Depending on the amplification settings and the dynamic range of the measured signal, values can occur that are outside of the measurement range, and most instruments will simply pile those values at the minimum or maximum range limit. The boundaryFilter removes these values, either for a single parameter, or for a combination of parameters. Note that it is often desirable to treat boundary events on a per-parameter basis, since their values might be uninformative for one particular channel, but still be useful in all of the other channels.

The constructor boundaryFilter is a convenience function for object instantiation. Evaluating a boundaryFilter results in a single sub-populations, an hence in an object of class filterResult.

Value

Returns a boundaryFilter object for use in filtering flowFrames or other flow cytometry objects.

Extends

Class "parameterFilter", directly.
Class "concreteFilter", by class parameterFilter, distance 2.
Class "filter", by class parameterFilter, distance 3.
Slots

tolerance: Object of class "numeric". The machine tolerance used to decide whether an event is on the measurement boundary. Essentially, this is done by evaluating \( x > \text{minRange} + \text{tolerance} \) & \( x < \text{maxRange} - \text{tolerance} \).

side: Object of class "character". The margin on which to evaluate the filter. Either upper for the upper margin or lower for the lower margin or both for both margins.

Objects from the Class

Objects can be created by calls of the form `new("boundaryFilter", ...)` or using the constructor `boundaryFilter`. Using the constructor is the recommended way of object instantiation:

Methods

`%in%` signature(x = "flowFrame", table = "boundaryFilter"): The workhorse used to evaluate the filter on data. This is usually not called directly by the user, but internally by calls to the `filter` methods.

`show` signature(object = "boundaryFilter"): Print information about the filter.

Author(s)

Florian Hahne

See Also

`flowFrame`, `flowSet`, `filter` for evaluation of `boundaryFilters` and `Subset` for subsetting of flow cytometry data sets based on that.

Examples

```r
## Loading example data
dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))

## Create directly. Most likely from a command line
boundaryFilter("FSC-H", filterId="myBoundaryFilter")

## To facilitate programmatic construction we also have the following
bf <- boundaryFilter(filterId="myBoundaryFilter", x=c("FSC-H"))

## Filtering using boundaryFilter
fres <- filter(dat, bf)
fres
summary(fres)

## We can subset the data with the result from the filtering operation.
Subset(dat, fres)

## A boundaryFilter on the lower margins of several channels
bf2 <- boundaryFilter(x=c("FSC-H", "SSC-H"), side="lower")
```
class "characterOrTransformation"

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class "characterOrTransformation" in the signature.

References

~put references to the literature/web site here~

Examples

showClass("characterOrTransformation")

checkOffset

Fix the offset when its values recorded in header and TEXT don’t agree

Description

Fix the offset when its values recorded in header and TEXT don’t agree

Usage

checkOffset(offsets, x, ignore.text.offset = FALSE, ...)

Arguments

offsets the named vector returned by findOffsets
x the text segmented returned by readFCStext
ignore.text.offset whether to ignore the offset info stored in TEXT segment
... not used.

Value

the updated offsets
coerce  \hspace{1cm} \textit{Convert an object to another class}

\textbf{Description}

These functions manage the relations that allow coercing an object to a given class.

\textbf{Arguments}

\begin{itemize}
\item \textbf{from}, \textbf{to} \hspace{1cm} The classes between which \texttt{def} performs coercion. (In the case of the \texttt{coerce} function, these are objects from the classes, not the names of the classes, but you're not expected to call \texttt{coerce} directly.)
\end{itemize}

\textbf{Details}

The function supplied as the third argument is to be called to implement \texttt{as(x, to)} when \texttt{x} has class \texttt{from}. Need we add that the function should return a suitable object with class \texttt{to}.

\textbf{Author(s)}

F. Hahne, B. Ellis

\textbf{Examples}

\begin{verbatim}
samp1 <- read.FCS(system.file("extdata","0877408774.E07", package="flowCore"))
samp2 <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
samples <- list("sample1"=samp1,"sample2"=samp2)
experiment <- as(samples,"flowSet")
\end{verbatim}

\textbf{compensateActionItem-class}

\textit{Class "compensateActionItem"}

\textbf{Description}

Class and method to capture compensation operations in a flow cytometry workflow.

\textbf{Usage}

\begin{verbatim}
compensateActionItem(ID = paste("compActionRef", guid(), sep = "_"),
                      name = paste("action", identifier(get(compensate)), sep = "_"),
                      parentView, compensate, workflow)
\end{verbatim}
**Arguments**

- **workflow**: An object of class `workFlow` for which a view is to be created.
- **ID**: A unique identifier of the view, most likely created by using the internal `guid` function.
- **name**: A more human-readable name of the view.
- **parentView, compensate**: References to the parent `view` and `compensation` objects, respectively.

**Details**

`compensateActionItems` provide a means to bind compensation operations in a workflow. Each `compensateActionItem` represents a single `compensation`.

**Value**

A reference to the `compensateActionItem` that is created inside the `workFlow` environment as a side effect of calling the `add` method.

A `compensateActionItem` object for the constructor.

**Objects from the Class**

Objects should be created using the `add` method, which creates a `compensateActionItem` from a `compensation` object and directly assigns it to a `workFlow`. Alternatively, one can use the `compensateActionItem` constructor function for more programmatic access.

**Slots**

- **compensate**: Object of class "fcCompensateReference". A reference to the `compensation` object that is used for the compensation operation.
- **ID**: Object of class "character". A unique identifier for the `actionItem`.
- **name**: Object of class "character". A more human-readable name.
- **parentView**: Object of class "fcViewReference". A reference to the parent `view` the `compensateActionItem` is applied on.
- **env**: Object of class "environment". The evaluation environment in the `workFlow`.

**Extends**

Class "`actionItem`", directly.

**Methods**

- **print** signature(`x = "compensateActionItem"`): Print details about the object.
- **Rm** signature(`symbol = "compensateActionItem", envir = "workFlow", subSymbol = "character"`): Remove a `compensateActionItem` from a `workFlow`. This method is recursive and will also remove all dependent views and `actionItems`.
- **show** signature(`object = "compensateActionItem"`): Print details about the object.

**Author(s)**

Florian Hahne
compensatedParameter-class

**Description**

Emission spectral overlap can be corrected by subtracting the amount of spectral overlap from the total detected signals. This compensation process can be described by using spillover matrices. compensatedParameter objects allow for compensation of specific parameters the user is interested in by creating compensatedParameter objects and evaluating them. This allows for use of compensatedParameter in gate definitions.

**Objects from the Class**

Objects can be created by calls of the form `compensatedParameter(parameters, spillRefId, transformationId, searchEnv)`.

**Slots**

- `.Data`: Object of class "function"
- `parameters`: Object of class "character" - flow parameters to be compensated
- `spillRefId`: Object of class "character" - name of the compensation object (The compensation object contains the spillover matrix)
- `searchEnv`: Object of class "environment" - environment in which the compensation object is defined
- `transformationId`: Object of class "character" - unique Id to reference the compensatedParameter object

**Extends**


**Methods**

No methods defined with class "compensatedParameter" in the signature.

**Note**

The transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)
compensateView-class

Author(s)
Gopalakrishnan N,F.Hahne

References

See Also
compensation

Examples
samp <- read.flowSet(path=system.file("extdata", "compdata", "data", package="flowCore"))
cfile <- system.file("extdata", "compdata", "compmatrix", package="flowCore")
comp.mat <- read.table(cfile, header=TRUE, skip=2, check.names = FALSE)
comp.mat

## create a compensation object
comp <- compensation(comp.mat,compensationId="comp1")

## create a compensated parameter object
cPar1<-compensatedParameter(c("FL1-H","FL3-H"),"comp",searchEnv=.GlobalEnv)
compOut<-eval(cPar1)(exprs(samp[[1]]))

compensateView-class

Class "compensateView"

Description
Class and method to capture the result of compensation operations in a flow cytometry workflow.

Usage
compensateView(workflow, ID=paste("compViewRef", guid(), sep="_"),
               name="default", action, data)

Arguments

workflow An object of class workflow for which a view is to be created.

ID A unique identifier of the view, most likely created by using the internal guid
function.

name A more human-readable name of the view.

data, action References to the data and actionItem objects, respectively.

Value
A reference to the compensateView that is created inside the workflow environment as a side effect
of calling the add method.
A compensateView object for the constructor.
**Objects from the Class**

Objects should be created using the `add` method, which creates a `compensateView` from a compensation object and directly assigns it to a `workFlow`. Alternatively, one can use the `compensateView` constructor function for more programmatic access.

**Slots**

- **ID:** Object of class "character". A unique identifier for the view.
- **name:** Object of class "character". A more human-readable name
- **action:** Object of class "fcActionReference". A reference to the `actionItem` that generated the view.
- **env:** Object of class "environment". The evaluation environment in the `workFlow`.
- **data:** Object of class "fcDataReference". A reference to the data that is associated to the view.

**Extends**

Class "view", directly.

**Methods**

- **Rm** signature(symbol = "compensateView", envir = "workFlow", subSymbol = "character"): Remove a `compensateView` from a `workFlow`. This method is recursive and will also remove all dependent views and `actionItems`.

**Author(s)**

Florian Hahne

**See Also**

`workFlow, view, gateView, transformView, normalizeView, actionItem`

**Examples**

```r
showClass("view")
```

---

### compensation-class

Class "compensation"

**Description**

Class and methods to compensate for spillover between channels by applying a spillover matrix to a `flowSet` or a `flowFrame` assuming a simple linear combination of values.

**Usage**

```r
compensation(..., spillover, 
  compensationId="defaultCompensation")

compensate(x, spillover, ...)
```
Arguments

- **spillover**: The spillover or compensation matrix.
- **compensationId**: The identifier for the compensation object.
- **x**: An object of class `flowFrame` or `flowSet`.
- **...**: Further arguments.

The constructor is designed to be useful in both programmatic and interactive settings, and ... serves as a container for possible arguments. The following combinations of values are allowed:

- Elements in ... are character scalars of parameter names or transform objects and the colnames in spillover match to these parameter names.
- The first element in ... is a character vector of parameter names or a list of character scalars or transform objects and the colnames in spillover match to these parameter names.
- Argument spillover is missing and the first element in ... is a matrix, in which case it is assumed to be the spillover matrix.
- ... is missing, in which case all parameter names are taken from the colnames of spillover.

Details

The essential premise of compensation is that some fluorochromes may register signals in detectors that do not correspond to their primary detector (usually a photomultiplier tube). To compensate for this fact, some sort of standard is used to obtain the background signal (no dye) and the amount of signal on secondary channels for each fluorochrome relative to the signal on their primary channel.

To calculate the spillover percentage we use either the mean or the median (more often the latter) of the secondary signal minus the background signal for each dye to obtain an \( n \times n \) matrix, \( S \), of so-called spillover values, expressed as a percentage of the primary channel. The observed values are then considered to be a linear combination of the true fluorescence and the spillover from each other channel so we can obtain the true values by simply multiplying by the inverse of the spillover matrix.

The spillover matrix can be obtained through several means. Some flow cytometers provide a spillover matrix calculated during acquisition, possibly by the operator, that is made available in the metadata of the flowFrame. While there is a theoretical standard keyword \$SPILL\ it can also be found in the SPILLOVER or SPILL keyword depending on the cytometry. More commonly the spillover matrix is calculated using a series of compensation cells or beads collected before the experiment. If you have set of FCS files with one file per fluorochrome as well as an unstained FCS file you can use the `spillover` method for `flowSets` to automatically calculate a spillover matrix.

The `compensation` class is essentially a wrapper around a matrix that allows for transformed parameters and method dispatch.

Value

- A compensation object for the constructor.
- A `flowFrame` or `flowSet` for the compensate methods.

Objects from the Class

Objects should be created using the constructor `compensation()`. See the Usage and Arguments sections for details.
Slots

spillover: Object of class matrix; the spillover matrix.

compensationId: Object of class character. An identifier for the object.

parameters: Object of class parameters. The flow parameters for which the compensation is defined. This can also be objects of class transform, in which case the compensation is performed on the compensated parameters.

Methods

compensate signature(x = "flowFrame", spillover = "compensation"): Apply the compensation defined in a compensation object on a flowFrame. This returns a compensated flowFrame.

Usage:
compensate(flowFrame, compensation)

compensate signature(x = "flowFrame", spillover = "matrix"): Apply a compensation matrix to a flowFrame. This returns a compensated flowFrame.

Usage:
compensate(flowFrame, matrix)

compensate signature(x = "flowFrame", spillover = "data.frame"): Try to coerce the data.frame to a matrix and apply that to a flowFrame. This returns a compensated flowFrame.

Usage:
compensate(flowFrame, data.frame)

identifier, identifier< signature(object = "compensation"): Accessor and replacement methods for the compensationId slot.

Usage:
identifier(compensation)
identifier(compensation) <- value

parameters signature(object = "compensation"): Get the parameter names of the compensation object. This method also tries to resolve all transforms and transformReferences before returning the parameters as character vectors. Unresolvable references return NA.

Usage:
parameters(compensation)

show signature(object = "compensation"): Print details about the object.

Usage:
This method is automatically called when the object is printed on the screen.

Author(s)

F. Hahne, B. Ellis, N. Le Meur

See Also

spillover
## concreteFilter-class

### Examples

```r
## Read sample data and a sample spillover matrix
samp <- read.flowSet(path=system.file("extdata", "compdata", "data", package="flowCore"))
cfile <- system.file("extdata","compdata","compmatrix", package="flowCore")
comp.mat <- read.table(cfile, header=TRUE, skip=2, check.names = FALSE)
comp.mat

## compensate using the spillover matrix directly
summary(samp)
samp <- compensate(samp, comp.mat)
summary(samp)

## create a compensation object and compensate using that
comp <- compensation(comp.mat)
compensate(samp, comp)

## demo the sample-specific compensation
## create a list of comps (each element could be a different compensation tailored for the specific sample)
comps <- sapply(sampleNames(samp), function(sn)comp, simplify = FALSE)
# the names of comps must be matched to sample names of the flowset
compensate(samp, comps)
```

---

### concreteFilter-class

**Class** "concreteFilter"

#### Description

The `concreteFilter` serves as a base class for all filters that actually implement a filtering process. At the moment this includes all filters except `filterReference`, the only non-concrete filter at present.

#### Objects from the Class

Objects of this class should never be created directly. It serves only as a point of inheritance.

#### Slots

- **filterId**: The identifier associated with this class.

#### Extends

Class "filter", directly.

#### Author(s)

B. Ellis

#### See Also

`parameterFilter`
**Class** "dg1polynomial"

**Description**

dg1polynomial allows for scaling, linear combination and translation within a single transformation defined by the function

\[ f(\text{parameter}_1, \ldots, \text{parameter}_n, a_1, \ldots, a_n, b) = b + \sum_{i=1}^{n} a_i \times \text{parameter}_i \]

**Objects from the Class**

Objects can be created by using the constructor `dg1polynomial(parameter, a, b, transformationId)`.

**Slots**

- `.Data`: Object of class "function"
- `.parameters`: Object of class "parameters" - the flow parameters that are to be transformed
- `a`: Object of class "numeric" - coefficients of length equal to the number of flow parameters
- `b`: Object of class "numeric" - coefficient of length 1 that performs the translation
- `transformationId`: Object of class "character" unique ID to reference the transformation

**Extends**


**Methods**

No methods defined with class "dg1polynomial" in the signature.

**Note**

The transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

**Author(s)**

Gopalakrishnan N, F.Hahne

**References**

Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry V 1.5

**See Also**

ratio, quadratic, squareroot
Examples

dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
dg1<-dgipolynomial(c("FSC-H","SSC-H"),a=c(1,2),b=1,transformationId="dg1")
transOut<-eval(dg1)(exprs(dat))

Arguments

x Object of class flowFrame.
FUN the function to be applied. In the case of functions like `+`, `%*%`, etc., the function name must be backquoted or quoted.
... optional arguments to `FUN`.

Description

Returns a vector or array of values obtained by applying a function to the margins of a flowFrame. This is equivalent of running apply on the output of exprs(flowFrame).

Usage

each.col(x, FUN, ...)
each.row(x, FUN, ...)

Author(s)

B. Ellis, N. LeMeur, F. Hahne

See Also

apply

Examples

samp <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"),
transformation="linearize")
each.col(samp, summary)
Description

EH transformation of a parameter is defined by the function

\[
EH(parameter, a, b) = 10^{\left(\frac{\text{parameter}}{a}\right)} + \frac{b \times \text{parameter}}{a} - 1 \quad \text{parameter} \geq 0
\]

\[
-10^{\left(-\frac{\text{parameter}}{a}\right)} + \frac{b \times \text{parameter}}{a} + 1 \quad \text{parameter} < 0
\]

Objects from the Class

Objects can be created by calls to the constructor EHtrans(parameters, a, b, transformationId)

Slots

.Data: Object of class "function" ~-

a: Object of class "numeric" - numeric constant greater than zero

b: Object of class "numeric" - numeric constant greater than zero

parameters: Object of class "transformation" - flow parameter to be transformed

transformationId: Object of class "character" - unique ID to reference the transformation

Extends


Methods

No methods defined with class "EHtrans" in the signature.

Note

The transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

Author(s)

Gopalakrishnan N, F.Hahne

References

Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry V 1.5

See Also

hyperlog
ellipsoidGate-class

Examples

```r
dat <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))
eh1 <- EHtrans("FSC-H", a=1250, b=4, transformationId="eh1")
transOut <- eval(eh1)(exprs(dat))
```

Description

Class and constructor for n-dimensional ellipsoidal filter objects.

Usage

```r
ellipsoidGate(..., .gate, mean, distance=1, filterId="defaultEllipsoidGate")
```

Arguments

- `filterId`: An optional parameter that sets the filterId of this gate.
- `.gate`: A definition of the gate via a covariance matrix.
- `mean`: Numeric vector of equal length as dimensions in `.gate`.
- `distance`: Numeric scalar giving the Mahalanobis distance defining the size of the ellipse. This mostly exists for compliance reasons to the gatingML standard as `mean` and `gate` should already uniquely define the ellipse. Essentially, `distance` is merely a factor that gets applied to the values in the covariance matrix.
- `...`: You can also directly describe the covariance matrix through named arguments, as described below.

Details

A convenience method to facilitate the construction of a ellipsoid filter objects. Ellipsoid gates in n dimensions (n \(\geq\) 2) are specified by a covariance matrix and a vector of mean values giving the center of the ellipse.

This function is designed to be useful in both direct and programmatic usage. In the first case, simply describe the covariance matrix through named arguments. To use this function programmatically, you may pass a covariance matrix and a mean vector directly, in which case the parameter names are the colnames of the matrix.

Value

Returns a ellipsoidGate object for use in filtering flowFrames or other flow cytometry objects.

Extends

Class "parameterFilter", directly.
Class "concreteFilter", by class parameterFilter, distance 2.
Class "filter", by class parameterFilter, distance 3.
Slots

- **mean**: Objects of class "numeric". Vector giving the location of the center of the ellipse in \(n\) dimensions.
- **cov**: Objects of class "matrix". The covariance matrix defining the shape of the ellipse.
- **distance**: Objects of class "numeric". The Mahalanobis distance defining the size of the ellipse.
- **parameters**: Object of class "character", describing the parameter used to filter the `flowFrame`.
- **filterId**: Object of class "character", referencing the filter.

Objects from the Class

Objects can be created by calls of the form `new("ellipsoidGate", ...)` or by using the constructor `ellipsoidGate`. Using the constructor is the recommended way of object instantiation:

Methods

- `%in%` signature(x = "flowFrame", table = "ellipsoidGate"): The workhorse used to evaluate the filter on data. This is usually not called directly by the user, but internally by calls to the `filter` methods.
- `show` signature(object = "ellipsoidGate"): Print information about the filter.

Note

See the documentation in the `flowViz` package for plotting of `ellipsoidGates`.

Author(s)

F.Hahne, B. Ellis, N. LeMeur

See Also

- `flowFrame`, `polygonGate`, `rectangleGate`, `polytopeGate`, `filter` for evaluation of `rectangleGates` and `split` and `Subset` for splitting and subsetting of flow cytometry data sets based on that.

Examples

```r
## Loading example data
dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))

## Defining the gate
cov <- matrix(c(6879, 3612, 3612, 5215), ncol=2, rownames=list(c("FSC-H", "SSC-H")), colnames=c("FSC-H", "SSC-H"))
mean <- c("FSC-H"=430, "SSC-H"=175)
eg <- ellipsoidGate(filterId= "myEllipsoidGate", .gate=cov, mean=mean)

## Filtering using ellipsoidGates
fres <- filter(dat, eg)
fres
summary(fres)

## The result of ellipsoid filtering is a logical subset
Subset(dat, fres)
```
estimateMedianLogicle

Estimates a common logicle transformation for a flowSet.

Description

Of the negative values for each channel specified, the median of the specified quantiles are used.

Usage

estimateMedianLogicle(flow_set, channels, m = 4.5, q = 0.05)

Arguments

- `flow_set`: object of class 'flowSet'
- `channels`: character vector of channels to transform
- `m`: TODO – default value from .lgclTrans
- `q`: quantile

Value

TODO

exponential-class

Class "exponential"

Description

Exponential transform class defines a transformation given by the function

\[ f(\text{parameter}, a, b) = e^{\text{parameter}/b} \cdot \frac{1}{a} \]

Objects from the Class

Objects can be created by calls to the constructor exponential(parameters, a, b).
Slots

.Data: Object of class "function" ~
a: Object of class "numeric" - non zero constant
b: Object of class "numeric" - non zero constant
parameters: Object of class "transformation" - flow parameter to be transformed
transformationId: Object of class "character" - unique ID to reference the transformation

Extends


Methods

No methods defined with class "exponential" in the signature.

Note

The exponential transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column

Author(s)

Gopalakrishnan N, F.Hahne

References

Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry V 1.5

See Also

logarithm

Examples

dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
expl<-exponential(parameters="FSC-H",a=1,b=37,transformationId="exp1")
transOut<-eval(expl)(exprs(dat))
expressionFilter-class

Class "expressionFilter"

Description

A filter holding an expression that can be evaluated to a logical vector or a vector of factors.

Usage

expressionFilter(expr, ..., filterId="defaultExpressionFilter")
char2ExpressionFilter(expr, ..., filterId="defaultExpressionFilter")

Arguments

filterId  An optional parameter that sets the filterId of this filter. The object can later be identified by this name.
expr      A valid R expression or a character vector that can be parsed into an expression.
...       Additional arguments that are passed to the evaluation environment of the expression.

Details

The expression is evaluated in the environment of the flow cytometry values, hence the parameters of a flowFrame can be accessed through regular R symbols. The convenience function char2ExpressionFilter exists to programmatically construct expressions.

Value

Returns a expressionFilter object for use in filtering flowFrames or other flow cytometry objects.

Extends

Class "concreteFilter", directly.
Class "filter", by class concreteFilter, distance 2.

Slots

expr: The expression that will be evaluated in the context of the flow cytometry values.
args: An environment providing additional parameters.
deparse: A character scalar of the deparsed expression.
filterId: The identifier of the filter

Objects from the Class

Objects can be created by calls of the form new("expressionFilter", ...), using the expressionFilter constructor or, programmatically, from a character string using the char2ExpressionFilter function.
expressionFilter-class

Methods

%in% signature(x = "flowFrame", table = "expressionFilter"): The workhorse used to evaluate the gate on data. This is usually not called directly by the user, but internally by calls to the filter methods.

show signature(object = "expressionFilter"): Print information about the gate.

Author(s)

F. Hahne, B. Ellis

See Also

flowFrame, filter for evaluation of sampleFilters and split and Subset for splitting and sub-setting of flow cytometry data sets based on that.

Examples

## Loading example data
dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))

#Create the filter
ef <- expressionFilter("FSC-H > 200", filterId="myExpressionFilter")
ef

## Filtering using sampleFilters
fres <- filter(dat, ef)
fres
summary(fres)

## The result of sample filtering is a logical subset
newDat <- Subset(dat, fres)
all(exprs(newDat)[,"FSC-H"] > 200)

## We can also split, in which case we get those events in and those not in the gate as separate populations
split(dat, fres)

## Programmatically construct an expression
dat <- dat[, -8]
r <- range(dat)
cn <- paste("\", colnames(dat), "\", sep="")
exp <- paste(cn, ", "r[1,], "&", cn, "<", r[2,], collapse=" & ")
2 <- char2ExpressionFilter(exp, filterId="myExpressionFilter")
ef2
fres2 <- filter(dat, ef2)
fres2
summary(fres2)
**fcReference-class**  

**Class "fcReference" and its subclasses**

### Description

Classes and methods to provide reference-based semantics for flow cytometry workflows.

### Usage

```r
fcReference(ID=paste("genericRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcTreeReference(ID=paste("treeRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcAliasReference(ID=paste("aliasRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcDataReference(ID=paste("dataRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcActionReference(ID=paste("actionRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcViewReference(ID=paste("viewRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcFilterResultReference(ID=paste("fresRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcFilterReference(ID=paste("filterRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcCompensateReference(ID=paste("compRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcNormalizationReference(ID=paste("normRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcSubsettingReference(ID=paste("subRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcTransformReference(ID=paste("transRef", guid(), sep="_"), env=new.env(parent=emptyenv()))
fcNullReference(...)
```

```r
assign(x, value, pos = -1, envir = as.environment(pos), inherits = FALSE, immediate = TRUE)
```

```r
## S4 method for signature 'fcReference,missing,missing'
get(x, pos = -1, envir = as.environment(pos), mode = "any", inherits = TRUE)
```
isNull(f)

Rm(symbol, envir, subSymbol, ...)

Arguments

x, f, symbol  An object of class or inheriting from class fcReference.
ID          The reference identifier.
value      An arbitrary R object which is supposed to be assigned to the environment in
           the workflow object and to which a reference is returned.
env         An environment, usually within a workflow object.
pos, envir  Objects of class workflow.
inherits, immediate, mode, subSymbol, ...
           Further arguments from the generics that are not used in this context.

Details

These classes provide references to objects within an R environment and allow for method dispatch
based on the nature of the referenced object. The parent fcReference class is used for references to
all R objects, unless there exists a more specific subclass. fcTreeReference, fcViewReference,
and fcActionReference are used to reference to graphNEL, view, and actionItem objects, respec-
tively. fcDataReference should be used for flowFrame or flowSet objects, whereas fcFilterResultReference,
fcFilterReference, fcTransformReference, fcCompensateReference, and fcNormalizationReference
link to filterResult, filter, transform and compensation objects. fsStructureReference
only exists to jointly dispatch on certain subgroups of references.

Value

An object of class fcReference or one of its subclasses for the assign constructor.
The object referenced to for the get method.
A character string of the object symbol for the identifier method.
A logical scalar for the isNull method.

Extends

fcStructureReference:
Class "fcReference", directly.
fcTreeReference:
Class "fcStructureReference", directly. Class "fcReference", by class "fcStructureReference".
distance 2.
fcAliasReference:
Class "fcStructureReference", directly. Class "fcReference", by class "fcStructureReference".
distance 2.
fcDataReference:
Class "fcReference", directly.
fcActionReference:
Objects from the Class

Objects should be created via the assign constructor. Whenever an object is assigned to a workFlow using the assign method, an appropriate instance of class fcReference or one of its subclasses is returned. In addition, there are the usual constructor functions of same names as the classes that can be used for object instantiation without assignment. Note that this might lead to unresolvable references unless the object referenced to is available in the environment.

Slots

ID: Object of class "character" The name of the object in env referenced to.

env: Object of class "environment" An environment that contains the referenced objects. Usually, this will be the environment that’s part of a workFlow object.

Methods

get signature(x = "fcReference", pos = "missing", envir = "missing", mode = "missing", inherits = "")

Resolve the reference, i.e., get the object from the environment.
get signature(x = "fcNullReference", pos = "missing", envir = "missing", mode = "missing", inherits = "missing")
Resolve the reference. This always returns NULL.

identifier signature(object = "fcReference"): Return a character string of the object name.

isNull signature(f = "fcReference"): Check whether a fcReference is a NULL reference. Note that this is different from an unresolvable reference.

Rm signature(symbol = "fcReference", envir = "missing", subSymbol = "character"): Remove the object referenced to by a fcReference from its environment. The argument subSymbol will be automatically set by the generic and should never be provided by the user.

Rm signature(symbol = "fcReference", envir = "workFlow", subSymbol = "character"): Remove the object referenced to by a fcReference from a workFlow. The argument subSymbol will be automatically set by the generic and should never be provided by the user.

Rm signature(symbol = "fcNullReference", envir = "missing", subSymbol = "character"): Essentially, this doesn’t do anything since there is no object referenced to.

show signature(object = "fcReference"): Print details about the object.

show signature(object = "fcNullReference"): Print details about the object.

Author(s)
Florian Hahne

See Also
workFlow

Examples

tShowClass("fcReference")

---

**FCSTransTransform**

*Computes a transform using the 'iplogicle' function*

**Description**

Transforms FCS data using the iplogicle function from FCSTrans by Quian et al. The core functionality of FCSTrans has been imported to produce transformed FCS data rescaled and truncated as produced by FCSTrans. The \( w \) parameter is estimated by iplogicle automatically, then makes a call to iplogicore which in turn uses the logicle transform code of Wayne Moore.


**Usage**

FCSTransTransform(transformationId = "defaultFCSTransTransform", channelrange = 2^18, channeldecade = 4.5, range = 4096, cutoff = -111, w = NULL, rescale = TRUE)
Arguments

transformationId
A name to assign to the transformation. Used by the transform/filter routines.

channelrange
is the range of the data. By default, \(2^{18} = 262144\).

channeldecade
is the number of logarithmic decades. By default, it is set to 4.5.

range
the target resolution. The default value is \(2^{12} = 4096\).

cutoff
a threshold below which the logicle transformation maps values to 0.

w
the logicle width. This is estimated by iplogicle by default. Details can be found in the Supplementary File from Quian et al.

rescale
logical parameter whether or not the data should be rescaled to the number of channels specified in range. By default, the value is TRUE but can be set to FALSE if you want to work on the transformed scale.

Author(s)

Wayne Moore, N Gopalakrishnan

References

Y Quian, Y Liu, J Campbell, E Thompson, YM Kong, RH Scheuermann; FCSTrans: An open source software system for FCS file conversion and data transformation. Cytometry A, 2012

See Also

inverseLogicleTransform, estimateLogicle, logicleTransform

Examples

data(GvHD)
samp <- GvHD[[1]]
## User defined logicle function
lgcl <- transformList(c("FL1-H", "FL2-H"), FCSTransTransform())
after <- transform(samp, lgcl)

filter
Filter FCS files

Description

These methods link filter descriptions to a particular set of flow cytometry data allowing for the lightweight calculation of summary statistics common to flow cytometry analysis.

Usage

filter(x, filter, method = c("convolution", "recursive"),
    sides = 2L, circular = FALSE, init = NULL)
Arguments

- `x`: Object of class `flowFrame` or `flowSet`.
- `filter`: An object of class `filter` or a named list `filters`.
- `method`, `sides`, `circular`, `init`

These arguments are not used.

Details

The `filter` method conceptually links a filter description, represented by a `filter` object, to a particular `flowFrame`. This is accomplished via the `filterResult` object, which tracks the linked frame as well as caching the results of the filtering operation itself, allowing for fast calculation of certain summary statistics such as the percentage of events accepted by the filter. This method exists chiefly to allow the calculation of these statistics without the need to first `Subset` a `flowFrame`, which can be quite large.

When applying on a `flowSet`, the `filter` argument can either be a single `filter` object, in which case it is recycled for all frames in the set, or a named list of `filter` objects. The names are supposed to match the frame identifiers (i.e., the output of `sampleNames(x)` of the `flowSet`). If some frames identifiers are missing, the particular frames are skipped during filtering. Accordingly, all `filters` in the filter list that can’t be mapped to the `flowSet` are ignored. Note that all `filter` objects in the list must be of the same type, e.g. `rectangleGates`.

Value

A `filterResult` object or a `filterResultList` object if `x` is a `flowSet`. Note that `filterResult` objects are themselves filters, allowing them to be used in filter expressions or `Subset` operations.

Author(s)

F Hahne, B. Ellis, N. Le Meur

See Also

- `Subset`, `filterResult`

Examples

```
## Filtering a flowFrame
samp <- read.FCS(system.file("extdata","0877488774.B08", package="flowCore"))
rectGate <- rectangleGate(filterId="nonDebris","FSC-H"=c(200,Inf))
fr <- filter(samp, rectGate)
class(fr)
summary(fr)

## filtering a flowSet
data(GvHD)
foo <- GvHD[1:3]
fr2 <- filter(foo, rectGate)
class(fr2)
summary(fr2)

## filtering a flowSet using different filters for each frame
rg2 <- rectangleGate(filterId="nonDebris","FSC-H"=c(300,Inf))
rg3 <- rectangleGate(filterId="nonDebris","FSC-H"=c(400,Inf))
```
flist <- list(rectGate, rg2, rg3)
names(flist) <- sampleNames(foo)
fr3 <- filter(foo, flist)

filter-and-methods  Take the intersection of two filters

Description

There are two notions of intersection in flowCore. First, there is the usual intersection boolean operator & that has been overridden to allow the intersection of two filters or of a filter and a list for convenience. There is also the %&% or %subset% operator that takes an intersection, but with subset semantics rather than simple intersection semantics. In other words, when taking a subset, calculations from summary and other methods are taken with respect to the right hand filter. This primarily affects calculations, which are ordinarily calculated with respect to the entire population as well as data-driven gating procedures which will operate only on elements contained by the right hand filter. This becomes especially important when using filters such as norm2Filter

Author(s)

B. Ellis

filter-class  A class for representing filtering operations to be applied to flow data.

Description

The filter class is the virtual base class for all filter/gating objects in flowCore. In general you will want to subclass or create a more specific filter.

Slots

filterId: A character vector that identifies this filter. This is typically user specified but can be automatically deduced by certain filter operations, particularly boolean and set operations.

Objects from the Class

All filter objects in flowCore should be instantiated through their constructors. These are functions that share the same name with the respective filter classes. E.g., rectangleGate() is the constructor function for rectangular gates, and kmeansFilter() creates objects of class kmeansFilter. Usually these constructors can deal with various different inputs, allowing to utilize the same function in different programmatic or interactive settings. For all filters that operate on specific flow parameters (i.e., those inheriting from parameterFilter), the parameters need to be passed to the constructor, either as names or colnames of additional input arguments or explicitly as separate arguments. See the documentation of the respective filter classes for details. If parameters are explicitly defined as separate arguments, they may be of class character, in which case they will be evaluated literally as colnames in a flowFrame, or of class transform, in which case the filtering is performed on a temporarily transformed copy of the input data. See here for details.
Methods

%in% Used in the usual way this returns a vector of values that identify which events were accepted by the filter. A single filter may encode several populations so this can return either a logical vector, a factor vector or a numeric vector of probabilities that the event is accepted by the filter. Minimally, you must implement this method when creating a new type of filter.

&, |, ! Two filters can be composed using the usual boolean operations returning a filter class of a type appropriate for handling the operation. These methods attempt to guess an appropriate filterId for the new filter.

%subset%, %&% Defines a filter as being a subset of another filter. For deterministic filters the results will typically be equivalent to using an & operation to compose the two filters, though summary methods will use subset semantics when calculating proportions. Additionally, when the filter is data driven, such as norm2Filter, the subset semantics are applied to the data used to fit the filter possibly resulting in quite different, and usually more desirable, results.

%on% Used in conjunction with a transformList to create a transformFilter. This filter is similar to the subset filter in that the filtering operation takes place on transformed values rather than the original values.

filter A more formal version of %in%, this method returns a filterResult object that can be used in subsequent filter operations as well as providing more metadata about the results of the filtering operation.

summarizeFilter When implementing a new filter this method is used to update the filterDetails slot of a filterResult. It is optional and typically only needs to be implemented for data-driven filters.

Author(s)

B. Ellis, P.D. Haaland and N. LeMeur

See Also

transform, filter

Description

Membership methods must be defined for every object of type filter with respect to a flowFrame object. The operation is considered to be general and may return a logical, numeric or factor vector that will be handled appropriately. The ability to handle logical matrices as well as vectors is also planned but not yet implemented.

Author(s)

F.Hahne, B. Ellis
filterDetails-methods  Obtain details about a filter operation

Description

A filtering operation captures details about its metadata and stores it in a filterDetails slot that is accessed using the filterDetails method. Each set of metadata is indexed by the filterId of the filter allowing for all the metadata in a complex filtering operation to be recovered after the final filtering.

Methods

- result = "filterResult", filterId = "missing"  When no particular filterId is specified all the details are returned
- result = "filterResult", filterId = "ANY"  You can also obtain a particular subset of details

Author(s)

B. Ellis, P.D. Haaland and N. LeMeur

filterList-class  Class "filterList"

Description

Container for a list of filter objects. The class mainly exists for method dispatch.

Usage

filterList(x, filterId=identifier(x[[1]]))

Arguments

- x  A list of filter objects.
- filterId  The global identifier of the filter list. As default, we take the filterId of the first filter object in x.

Value

A filterList object for the constructor

Objects from the Class

Objects are created from regular lists using the constructor filterList.
Slots

.Data: Object of class "list". The class directly extends list, and this slot holds the list data.

filterId: Object of class "character". The identifier for the object.

Extends

Class "list", from data part.

Methods

show signature(object = "filterList"): Print details about the object.

identifier, identifier<- signature(object = "filtRef") Accessor and replacement method for the object's filterId slot.

Author(s)

Florian Hahne

See Also

filter.

Examples

f1 <- rectangleGate(FSC=c(100,200), filterId="testFilter")
f2 <- rectangleGate(FSC=c(200,400))
f1 <- filterList(list(a=f1, b=f2))
f1

identifier(f1)

filterReference-class Class filterReference

Description

A reference to another filter inside a reference. Users should generally not be aware that they are using this class, but it is used heavily by "filterSet" classes.

Objects from the Class

Objects are generally not created by users so there is no constructor function.

Slots

name: The R name of the referenced filter

env: The environment where the filter must live

filterId: The filterId, not really used since you always resolve
filterResult-class

Extends

Class "filter", directly.

Author(s)

B. Ellis

See Also

"filterSet"

filterResult-class  Class "filterResult"

Description

Container to store the result of applying a filter on a flowFrame object

Slots

frameId: Object of class "character" referencing the flowFrame object filtered. Used for sanity checking.
filterDetails: Object of class "list" describing the filter applied
filterId: Object of class "character" referencing the filter applied

Extends

Class "filter", directly.

Methods

==  test equality

Author(s)

B. Ellis, N. LeMeur

See Also

filter, "logicalFilterResult", "multipleFilterResult", "randomFilterResult"

Examples

showClass("filterResult")
Class "filterResultList"

Description

Container to store the result of applying a filter on a flowSet object

Objects from the Class

Objects are created by applying a filter on a flowSet. The user doesn’t have to deal with manual object instantiation.

Slots

Data: Object of class "list". The class directly extends list, and this slot holds the list data.
frameId: Object of class "character". The IDs of the flowFrames in the filtered flowSet.
filterDetails: Object of class "list". Since filterResultList inherits from filterResult, this slot has to be set. It contains only the input filter.
filterId: Object of class "character". The identifier for the object.

Extends


Methods

[ signature(x = "filterResultList", i = "ANY"): Subset to filterResultList.
[I signature(x = "filterResultList", i = "ANY"): Subset to individual filterResult.
names signature(x = "filterResultList"): Accessor to the frameId slot.
parameters signature(object = "filterResultList"): Return parameters on which data has been filtered.
show signature(object = "filterResultList"): Print details about the object.
split signature(x = "flowSet", f = "filterResultList"): Split a flowSet based on the results in the filterResultList. See split for details.
summary signature(object = "filterResultList"): Summarize the filtering operation. This creates a filterSummaryList object.

Author(s)

Florian Hahne

See Also

filter, filterResult, logicalFilterResult, multipleFilterResult, randomFilterResult
Examples

```r
## Not run:
library(flowStats)
## Loading example data and creating a curv1Filter
data(GvHD)
dat <- GvHD[1:3]
c1f <- curv1Filter(filterId="myCurv1Filter", x=list("FSC-H"), bwFac=2)

## applying the filter
fres <- filter(dat, c1f)
fres

## subsetting the list
fres[[1]]
fres[1:2]

## details about the object
parameters(fres)
names(fres)
summary(fres)

## splitting based on the filterResults
split(dat, fres)

## End(Not run)
```

filters-class  
Class "filters" and "filtersList"

Description

The filters class is the container for a list of filter objects. The class mainly exists for displaying multiple filters/gates on one single panel(flowFrame) of `xyplot`. Note that it is different from `filterList` class which is to be applied to a flowSet. In other words, filter objects of a `filterList` are to be applied to different flowFrames. However, all of filter objects of a filters are for one single flowFrame, more specifically for one pair of projections(parameters). So these filters should share the common parameters.

And `filtersList` is a list of filters objects, which are to be applied to a flowSet.

Usage

```r
filters(x)
filtersList(x)
```

Arguments

- `x` A list of filter or filters objects.

Value

A filters or filtersList object for the constructor
Slot 'Data': Object of class "list". The class directly extends list, and this slot holds the list data.

Extends
Class "list", from data part.

Objects from the Class
Objects are created from regular lists using the constructors filters and filtersList.

Author(s)
Mike Jiang

See Also
filter, filterList
Details

filterSet objects are intended to provide a convenient grouping mechanism for a particular gating strategy. To accomplish this, much like the flowSet object, the filterSet object introduces reference semantics through the use of an environment, allowing users to change an upstream filter via the usual assignment mechanism and have that change reflected in all dependent filters. We do this by actually creating two filters for each filter in the filterSet. The first is the actual concrete filter, which is assigned to a variable of the form .name where name is the original filter name. A second filterReference filter is the created with the original name to point to the internal name. The allows us to evaluate a formula in the environment without creating a copy of the original filter.

Author(s)

B. Ellis

See Also

filterSet

filterSummary-class  Class "filterSummary"

Description

Class and methods to handle the summary information of a gating operation.

Usage

## S4 method for signature 'filterResult'
summary(object, ...)

Arguments

object An object inheriting from class filterResult which is to be summarized.
...
Further arguments that are passed to the generic.

Details

Calling summary on a filterResult object prints summary information on the screen, but also creates objects of class filterSummary for computational access.

Value

An object of class filterSummary for the summary constructor, a named list for the subsetting operators. The $ operator returns a named vector of the respective value, where each named element corresponds to one sub-population.

Objects from the Class

Objects are created by calling summary on a link(filterResult) object. The user doesn’t have to deal with manual object instantiation.
filterSummary-class

Slots

name: Object of class "character" The name(s) of the populations created in the filtering operation. For a logicalFilterResult this is just a single value; the name of the link(filter).

t: Object of class "numeric". The number of events within the population(s).

count: Object of class "numeric". The total number of events in the gated flowFrame.

Methods

[[ signature(x = "filterSummary", i = "numeric"): Subset the filterSummary to a single population. This only makes sense for multipleFilterResults. The output is a list of summary statistics.

[[ signature(x = "filterSummary", i = "character"): see above

$ signature(x = "filterSummary", name = "ANY"): A list-like accessor to the slots and more. Valid values are n and count (those are identical), true and in (identical), false and out (identical), name, p and q(1-p).

coerce signature(from = "filterSummary", to = "data.frame"): Coerce object to data.frame.

length signature(x = "filterSummary"): The number of populations in the filterSummary.

names signature(x = "filterSummary"): The names of the populations in the filterSummary.

print signature(x = "filterSummary"): Print details about the object.

show signature(object = "filterSummary"): Print details about the object.

toTable signature(x = "filterSummary"): Coerce object to data.frame.

Author(s)

Florian Hahne, Byron Ellis

See Also

filterResult, logicalFilterResult, multipleFilterResult, flowFrame filterSummaryList

Examples

## Not run:
library(flowStats)

## Loading example data, creating and applying a curv1Filter
dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
c1f <- curv1Filter(filterId="myCurv1Filter", x=list("FSC-H"), bwFac=2)
fres <- filter(dat, c1f)

## creating and showing the summary
summary(fres)
s <- summary(fres)

## subsetting
s[[1]]
s["peak 2"]
## Description

Class and methods to handle summary statistics for filtering operations on whole `flowSets`.

### Arguments

- **object**: An object of class `filterResultList` which is to be summarized.
- **...**: Further arguments that are passed to the generic.

### Details

Calling `summary` on a `filterResult` object prints summary information on the screen, but also creates objects of class `filterSummary` for computational access.

### Value

An object of class `filterSummaryList`.

### Objects from the Class

Objects are created by calling `summary` on a `filterResultList` object. The user doesn’t have to deal with manual object instantiation.

### Slots

- **.Data**: Object of class "list". The class directly extends list, and this slot holds the list data.

### Extends

Class "list", from data part.

### Usage

`summary(object,...)`

### Methods

- **toTable** signature(`x = "filterSummaryList"`): Coerce object to `data.frame`. Additional factors are added to indicate list items in the original object.
flowFrame-class

'flowFrame': a class for storing observed quantitative properties for a population of cells from a FACS run

Description

This class represents the data contained in a FCS file or similar data structure. There are three parts of the data:

1. a numeric matrix of the raw measurement values with rows=events and columns=parameters
2. annotation for the parameters (e.g., the measurement channels, stains, dynamic range)
3. additional annotation provided through keywords in the FCS file

Details

Objects of class flowFrame can be used to hold arbitrary data of cell populations, acquired in flow-cytometry.

FCS is the Data File Standard for Flow Cytometry, the current version is FCS 3.0. See the vignette of this package for additional information on using the object system for handling of flow-cytometry data.

Examples

## Not run:

library(flowStats)

## Loading example data, creating and applying a curv1Filter
data(GvHD)
dat <- GvHD[1:3]
c1f <- curv1Filter(filterId="myCurv1Filter", x=list("FSC-H"), bwFac=2)
fres <- filter(dat, c1f)

## creating and showing the summary
summary(fres)
s <- summary(fres)

## subsetting
s[,1]

## accessing details
toTable(s)

## End(Not run)
Creating Objects

Objects can be created using

```r
new("flowFrame",
.exprs = ...., Object of class matrix
.parameters = ...., Object of class AnnotatedDataFrame
description = ...., Object of class list
)
```

or the constructor `flowFrame`, with mandatory arguments `exprs` and optional arguments `parameters` and `description`.

```r
flowFrame(exprs, parameters, description=list())
```

To create a `flowFrame` directly from an FCS file, use function `read.FCS`. This is the recommended and safest way of object creation, since `read.FCS` will perform basic data quality checks upon import. Unless you know exactly what you are doing, creating objects using `new` or the constructor is discouraged.

Slots

- **exprs**: Object of class `matrix` containing the measured intensities. Rows correspond to cells, columns to the different measurement channels. The `colnames` attribute of the matrix is supposed to hold the names or identifiers for the channels. The `rownames` attribute would usually not be set.

- **parameters**: An `AnnotatedDataFrame` containing information about each column of the `flowFrame`. This will generally be filled in by `read.FCS` or similar functions using data from the FCS keywords describing the parameters.

- **description**: A list containing the metadata included in the FCS file.

Methods

There are separate documentation pages for most of the methods listed here which should be consulted for more details.

Subsetting. Returns an object of class `flowFrame`. The subsetting is applied to the `exprs` slot, while the description slot is unchanged. The syntax for subsetting is similar to that of `data.frames`. In addition to the usual index vectors (integer and logical by position, character by parameter names), `flowFrames` can be subset via `filterResult` and `filter` objects.

```r
flowFrame[i,j]
flowFrame[filter,]
flowFrame[filterResult,]
```

Note that the value of argument `drop` is ignored when subsetting `flowFrames`.

\$ Subsetting by channel name. This is similar to subsetting of columns of `data.frames`, i.e., `frame$FSC.H` is equivalent to `frame[, "FSC.H"]`. Note that column names may have to be quoted if they are no valid R symbols (e.g. `frame$"FSC-H"`).

**exprs, exprs<-** Extract or replace the raw data intensities. The replacement value must be a numeric matrix with column names matching the parameter definitions. Implicit subsetting is allowed (i.e. less columns in the replacement value compared to the original `flowFrame`, but all have to be defined there).

```r
exprs(flowFrame)
exprs(flowFrame) <- value
```
head, tail  Show first/last elements of the raw data matrix
Usage:
head(flowFrame)
tail(flowFrame)

description, description<-  Extract or replace the whole list of annotation keywords. Usually one
would only be interested in a subset of keywords, in which case the keyword method is more
appropriate. The optional hideInternal parameter can be used to exclude internal FCS pa-
rameters starting with$.
Usage:
description(flowFrame)
description(flowFrame) <- value

keyword, keyword<-  Extract ore replace one or more entries from the description slot by key-
word. Methods are defined for character vectors (select a keyword by name), functions (select
a keyword by evaluating a function on their content) and for lists (a combination of the above).
See keyword for details.
Usage:
keyword(flowFrame)
keyword(flowFrame, character)
keyword(flowFrame, list)
keyword(flowFrame) <- list(value)

parameters, parameters<-  Extract parameters and return an object of class AnnotatedDataFrame,
or replace such an object. To access the actual parameter annotation, use pData(parameters(frame)).
Replacement is only valid with AnnotatedDataFrames containing all varLabels name, desc,
range, minRange and maxRange, and matching entries in the name column to the colnames of
the exprs matrix. See parameters for more details.
Usage:
parameters(flowFrame)
parameters(flowFrame) <- value

show  Display details about the flowFrame object.

summary  Return descriptive statistical summary (min, max, mean and quantile) for each channel
Usage:
summary(flowFrame)

plot  Basic plots for flowFrame objects. If the object has only a single parameter this produces a
histogram. For exactly two parameters we plot a bivariate density map (see smoothScatter
and for more than two parameters we produce a simple splom plot. To select specific param-
eters from a flowFrame for plotting, either subset the object or specify the parameters as a
character vector in the second argument to plot. The smooth parameters lets you toggle be-
tween density-type smoothScatter plots and regular scatterplots. For far more sophisticated
plotting of flow cytometry data, see the flowViz package.
Usage:
plot(flowFrame, ...)
plot(flowFrame, character, ...)
plot(flowFrame, smooth=FALSE, ...)

ncol, nrow, dim  Extract the dimensions of the data matrix.
Usage:
ncol(flowFrame)
nrow(flowFrame)
dim(flowFrame)
**featureNames**, colnames, colnames<-. colnames and featureNames are synonyms, they extract parameter names (i.e., the colnames of the data matrix). For colnames there is also a replacement method. This will update the name column in the parameters slot as well.

*Usage:*

```r
featureNames(flowFrame)
colnames(flowFrame)
colnames(flowFrame) <- value
```

**names** Extract pretty formatted names of the parameters including parameter descriptions.

*Usage:*

```r
names(flowFrame)
```

**identifier** Extract GUID of a flowFrame. Returns the file name if no GUID is available. See **identifier** for details.

*Usage:*

```r
identifier(flowFrame)
```

**range** Get instrument or actual data range of the flowFrame. Note that instrument dynamic range is not necessarily the same as the range of the actual data values, but the theoretical range of values the measurement instrument was able to capture. The values of the dynamic range will be transformed when using the transformation methods for flowFrames.

*Parameters:*
  
x: flowFrame object.
  
type: Range type. either "instrument" or "data". Default is "instrument"

*Usage:*

```r
range(x, type = "data")
```

**each_row, each_col** Apply functions over rows or columns of the data matrix. These are convenience methods. See **each_col** for details.

*Usage:*

```r
each_row(flowFrame, function, ...)
each_col(flowFrame, function, ...)
```

**transform** Apply a transformation function on a flowFrame object. This uses R's **transform** function by treating the flowFrame like a regular data.frame. flowCore provides an additional inline mechanism for transformations (see **%on%**) which is strictly more limited than the out-of-line transformation described here.

*Usage:*

```r
transform(flowFrame, translist, ...)
```

**filter** Apply a filter object on a flowFrame object. This returns an object of class **filterResult**, which could then be used for subsetting of the data or to calculate summary statistics. See **filter** for details.

*Usage:*

```r
filter(flowFrame, filter)
```

**split** Split flowFrame object according to a filter, a filterResult or a factor. For most types of filters, an optional flowSet=TRUE parameter will create a flowSet rather than a simple list. See **split** for details.

*Usage:*

```r
split(flowFrame, filter, flowSet=FALSE, ...)
split(flowFrame, filterResult, flowSet=FALSE, ...)
split(flowFrame, factor, flowSet=FALSE, ...)
```
Subset  Subset a flowFrame according to a filter or a logical vector. The same can be done using the standard subsetting operator with a filter, filterResult, or a logical vector as first argument.

Usage:
Subset(flowFrame, filter)
Subset(flowFrame, logical)

cbind2  Expand a flowFrame by the data in a numeric matrix of the same length. The matrix must have column names different from those of the flowFrame. The additional method for numerics only raises a useful error message.

Usage:
cbind2(flowFrame, matrix)
cbind2(flowFrame, numeric)

compensate  Apply a compensation matrix (or a compensation object) on a flowFrame object. This returns a compensated flowFrame.

Usage:
compensate(flowFrame, matrix)  compensate(flowFrame, data.frame)

spillover  Extract spillover matrix from description slot if present. It is equivalent to keyword(x, c("spillover", "SPILL")), thus will simply return a list of keywords value for "spillover" and "SPILL".

Usage:
spillover(flowFrame)

==  Test equality between two flowFrames

<, >, <=, >=  These operators basically treat the flowFrame as a numeric matrix.

initialize(flowFrame): Object instantiation, used by new; not to be called directly by the user.

Author(s)
F. Hahne, B. Ellis, P. Haaland and N. Le Meur

See Also
flowSet, read.FCS

Examples

## load example data
data(GvHD)
frame <- GvHD[[1]]

## subsetting
frame[1:4,]
frame[,3]
frame[,]"FSC-H"
frame$"SSC-H"

## accessing and replacing raw values
head(exprs(frame))
exprs(frame) <- exprs(frame)[1:3000,]
frame
exprs(frame) <- exprs(frame)[,1:6]
frame
## access FCS keywords
head(description(frame))
keyword(frame, c("FILENAME", "$FIL"))

## parameter annotation
parameters(frame)
pData(parameters(frame))

## summarize frame data
summary(frame)

## plotting
plot(frame)
if(require(flowViz)){
plot(frame)
plot(frame, c("FSC-H", "SSC-H"))
plot(frame[,1])
plot(frame, c("FSC-H", "SSC-H"), smooth=FALSE)
}

## frame dimensions
ncol(frame)
nrow(frame)
dim(frame)

## accessing and replacing parameter names
featureNames(frame)
all(featureNames(frame) == colnames(frame))
colnames(frame) <- make.names(colnames(frame))
colnames(frame)
parameters(frame)$name
names(frame)

## accessing a GUID
identifier(frame)
identifier(frame) <- "test"

## range of a frame
range(frame) #instrument range
range(frame, type = "data") #actual data range
range(frame)$FSC.H

## iterators
head(each_row(frame, mean))
head(each_col(frame, mean))

## transformation
opar <- par(mfcol=c(1:2))
if(require(flowViz))
plot(frame, c("FL1.H", "FL2.H"))
frame <- transform(frame, transformList(c("FL1.H", "FL2.H"), log))
if(require(flowViz))
plot(frame, c("FL1.H", "FL2.H"))
par(opar)
range(frame)
## filtering of flowFrames
rectGate <- rectangleGate(filterId="nonDebris","FSC.H"=c(200,Inf))
fres <- filter(frame, rectGate)
summary(fres)

## splitting of flowFrames
split(frame, rectGate)
split(frame, rectGate, flowSet=TRUE)
split(frame, fres)
f <- cut(exprs(frame$FSC.H), 3)
split(frame, f)

## subsetting according to filters and filter results
Subset(frame, rectGate)
Subset(frame, fres)
Subset(frame, as.logical(exprs(frame$FSC.H) < 300))
frame[rectGate,]
frame[fres,]

## accessing the spillover matrix
try(spillover(frame))

## check equality
frame2 <- frame
frame == frame2
exprs(frame2) <- exprs(frame)*2
frame == frame2

---

flowSet-class

'flowSet': a class for storing flow cytometry raw data from quantitative cell-based assays

Description
This class is a container for a set of flowFrame objects

Creating Objects

Objects can be created using
new('flowSet',
frames = ....,  # environment with flowFrames
phenoData = .... # object of class AnnotatedDataFrame
colnames = .... # object of class character
)

or via the constructor flowSet, which takes arbitrary numbers of flowFrames, either as a list or directly as arguments, along with an optional AnnotatedDataFrame for the phenoData slot and a character scalar for the name by which the object can be referenced.

flowSet(..., phenoData)

Alternatively, flowSets can be coerced from list and environment objects.

as(list("A"=frameA,"B"=frameB),"flowSet")
The safest and easiest way to create flowSets directly from FCS files is via the \texttt{read.flowSet} function, and there are alternative ways to specify the files to read. See the separate documentation for details.

**Slots**

frames: An \texttt{environment} containing one or more \texttt{flowFrame} objects.

\texttt{phenoData}: A \texttt{AnnotatedDataFrame} containing the phenotypic data for the whole data set. Each row corresponds to one of the \texttt{flowFrames} in the \texttt{frames} slot. The \texttt{sampleNames} of \texttt{phenoData} (see below) must match the names of the \texttt{flowFrame} in the \texttt{frames} environment.

\texttt{colnames}: A character object with the (common) column names of all the data matrices in the \texttt{flowFrames}.

**Methods**

\texttt{[}, \texttt{[[} Subsetting. \(x[i]\) where \(i\) is a scalar, returns a \texttt{flowSet} object, and \(x[[i]]\) a \texttt{flowFrame} object. In this respect the semantics are similar to the behavior of the subsetting operators for lists. \(x[i, j]\) returns a \texttt{flowSet} for which the parameters of each \texttt{flowFrame} have been subset according to \(j\). \(x[[i,j]]\) returns the subset of a single \texttt{flowFrame} for all parameters in \(j\). Similar to data frames, valid values for \(i\) and \(j\) are logicals, integers and characters.

Usage:

\begin{verbatim}
flowSet[i]
flowSet[i,j]
flowSet[[i]]
\end{verbatim}

\$ Subsetting by frame name. This will return a single \texttt{flowFrame} object. Note that names may have to be quoted if they are no valid R symbols (e.g. \texttt{flowSet$"sample 1"})

\texttt{colnames}, \texttt{colnames<-} Extract or replace the \texttt{colnames} slot.

Usage:

\begin{verbatim}
colnames(flowSet)
colnames(flowSet) <- value
\end{verbatim}

\texttt{identifier}, \texttt{identifier<-} Extract or replace the \texttt{identifier} item from the environment.

Usage:

\begin{verbatim}
identifier(flowSet)
identifier(flowSet) <- value
\end{verbatim}

\texttt{phenoData}, \texttt{phenoData<-} Extract or replace the \texttt{AnnotatedDataFrame} from the \texttt{phenoData} slot.

Usage:

\begin{verbatim}
phenoData(flowSet)
phenoData(flowSet) <- value
\end{verbatim}

\texttt{pData}, \texttt{pData<-} Extract or replace the data frame (or columns thereof) containing actual phenotypic information from the \texttt{phenoData} slot.

Usage:

\begin{verbatim}
pData(flowSet)
pData(flowSet)$someColumn <- value
\end{verbatim}

\texttt{varLabels}, \texttt{varLabels<-} Extract and set \texttt{varLabels} in the \texttt{AnnotatedDataFrame} of the \texttt{phenoData} slot.

Usage:

\begin{verbatim}
varLabels(flowSet)
varLabels(flowSet) <- value
\end{verbatim}
sampleNames Extract and replace sample names from the phenoData object. Sample names correspond to frame identifiers, and replacing them will also replace the GUID slot for each frame. Note that sampleName need to be unique.

Usage:
  sampleNames(flowSet)
  sampleNames(flowSet) <- value

keyword Extract or replace keywords specified in a character vector or a list from the description slot of each frame. See keyword for details.

Usage:
  keyword(flowSet, list(keywords))
  keyword(flowSet, keywords)
  keyword(flowSet) <- list(foo="bar")

length number of flowFrame objects in the set.

Usage:
  length(flowSet)

show display object summary.

summary Return descriptive statistical summary (min, max, mean and quantile) for each channel of each flowFrame

Usage:
  summary(flowSet)

fsApply Apply a function on all frames in a flowSet object. Similar to sapply, but with additional parameters. See separate documentation for details.

Usage:
  fsApply(flowSet, function, ...)
  fsApply(flowSet, function, use.exprs=TRUE, ...)

compensate Apply a compensation matrix on all frames in a flowSet object. See separate documentation for details.

Usage:
  compensate(flowSet, matrix)

transform Apply a transformation function on all frames of a flowSet object. See separate documentation for details.

Usage:
  transform(flowSet, ...)

filter Apply a filter object on a flowSet object. There are methods for filters, filterSets and lists of filters. The latter has to be a named list, where names of the list items are matching sampleNames of the flowSet. See filter for details.

Usage:
  filter(flowSet, filter)
  filter(flowSet, list(filters))

split Split all flowSet objects according to a filter, filterResult or a list of such objects, where the length of the list has to be the same as the length of the flowSet. This returns a list of flowFrames or an object of class flowSet if the flowSet argument is set to TRUE. Alternatively, a FlowSet can be split into separate subsets according to a factor (or any vector that can be coerced into factors), similar to the behaviour of split for lists. This will return a list of flowSets. See split for details.

Usage:
split(flowSet, filter)
split(flowSet, filterResult)
split(flowSet, list(filters))
split(flowSet, factor)

**Subset** Returns a flowSet of flowFrames that have been subset according to a filter or filterResult, or according to a list of such items of equal length as the flowSet.

**Usage:**
- Subset(flowSet, filter)
- Subset(flowSet, filterResult)
- Subset(flowSet, list(filters))

**rbind2** Combine two flowSet objects, or one flowSet and one flowFrame object.

**Usage:**
- rbind2(flowSet, flowSet)
- rbind2(flowSet, flowFrame)

**spillover** Compute spillover matrix from a compensation set. See separate documentation for details.

**Important note on storage and performance**

The bulk of the data in a flowSet object is stored in an environment, and is therefore not automatically copied when the flowSet object is copied. If x is an object of class flowSet, then the code

\[ y \leftarrow x \]

will create an object y that contains copies of the phenoData and administrative data in x, but refers to the same environment with the actual fluorescence data. See below for how to create proper copies.

The reason for this is performance. The pass-by-value semantics of function calls in R can result in numerous copies of the same data object being made in the course of a series of nested function calls. If the data object is large, this can result in considerable cost of memory and performance. flowSet objects are intended to contain experimental data in the order of hundreds of Megabytes, which can effectively be treated as read-only: typical tasks are the extraction of subsets and the calculation of summary statistics. This is afforded by the design of the flowSet class: an object of that class contains a phenoData slot, some administrative information, and a reference to an environment with the fluorescence data; when it is copied, only the reference is copied, but not the potentially large set of fluorescence data themselves.

However, note that subsetting operations, such as \[ y \leftarrow x[i] \] do create proper copies, including a copy of the appropriate part of the fluorescence data, as it should be expected. Thus, to make a proper copy of a flowSet x, use \[ y \leftarrow x[seq(along=x)] \]

**Author(s)**

F. Hahne, B. Ellis, P. Haaland and N. Le Meur

**See Also**

flowFrame, read.flowSet
Examples

```r
## load example data and object creation
data(GvHD)

## subsetting to flowSet
data(GvHD)
sel <- sampleNames(GvHD)[1:2]                 # sel[1, "FSC-H"]
GvHD[sampleNames(GvHD) == sel[1], colnames(GvHD[1]) == "SSC-H"]

## subsetting to flowFrame
GvHD[[1]]
GvHD[[1, 1:3]]                                # GvHD[[1, "FSC-H"]]
GvHD[[1, colnames(GvHD[1]) == "SSC-H"]]
GvHD$s5a02

## constructor
flowSet(GvHD[[1]], GvHD[[2]])                 # flowSet(GvHD[[1]], GvHD[[2]])
set <- phenoData(GvHD[[1:2,]])               # phenoData(GvHD[[1:2,]])
flowSet(s5a01=GvHD[[1]], s5a02=GvHD[[2]], phenoData=pd)

## colnames
colnames(set)
colnames(set) <- make.names(colnames(set))

## object name
identifier(set)
identifier(set) <- "test"

## phenoData
pd <- phenoData(set)
pd
pd$test <- "test"
phenoData(set) <- pd
pData(set)
varLabels(set)
varLabels(set)[6] <- "Foo"
varLabels(set)

## sampleNames
sampleNames(set)
sampleNames(set) <- LETTERS[1:length(set)]
sampleNames(set)

## keywords
keyword(set, list("transformation"))

## length
length(set)

## compensation
samp <- read.flowSet(path=system.file("extdata", "compdata", "data", package="flowCore"))
```
fsApply <- system.file("extdata","compdata","compmatrix", package="flowCore")
comp.mat <- read.table(fsApply, header=TRUE, skip=2, check.names = FALSE)
comp.mat
summary(samp[[1]])
samp <- compensate(samp, as.matrix(comp.mat))
summary(samp[[1]])

## transformation
opar <- par(mfcol=c(1:2))
plot(set[[1]], c("FL1.H", "FL2.H"))
set <- transform(set, transformList(c("FL1.H", "FL2.H"), log))
plot(set[[1]], c("FL1.H", "FL2.H"))
par(opar)

## filtering of flowSets
rectGate <- rectangleGate(filterId="nonDebris", FSC.H=c(200,Inf))
fres <- filter(set, rectGate)
class(fres)
summary(fres[[1]])
rectGate2 <- rectangleGate(filterId="nonDebris2", SSC.H=c(300,Inf))
fres2 <- filter(set, list(A=rectGate, B=rectGate2, C=rectGate, D=rectGate2))

## Splitting frames of a flowSet
split(set, rectGate)
split(set[1:2], rectGate, population="nonDebris2+")
split(set, c(1,1,2,2))

## subsetting according to filters and filter results
Subset(set, rectGate)
Subset(set, filter(set, rectGate))
 Subset(set, list(A=rectGate, B=rectGate2, C=rectGate, D=rectGate2))

## combining flowSets
rbind2(set[1:2], set[3:4])
rbind2(set[1:3], set[[4]])
rbind2(set[[4]], set[1:2])

fsApply

Apply a Function over values in a flowSet

Description

fsApply like many of the apply-style functions in R acts as an iterator for flowSet objects, allowing the application of a function to either the flowFrame or the data matrix itself. The output can be reconstructed as either a flowSet, a list or a matrix depending on options and the type of objects returned.

Usage

fsApply(x, FUN, ..., simplify=TRUE, use.exprs=FALSE)
Arguments

- **x**  
  flowSet to be used

- **FUN**  
  the function to be applied to each element of x

- **simplify**  
  logical (default: TRUE); if all true and all objects are flowFrame objects, a flowSet object will be constructed. If all of the values are of the same type there will be an attempt to construct a vector or matrix of the appropriate type (e.g. all numeric results will return a matrix).

- **use.exprs**  
  logical (default: FALSE); should the FUN be applied on the flowFrame object or the expression values.

- **...**  
  optional arguments to FUN.

Author(s)

- B. Ellis

See Also

- apply, sapply

Examples

```r
fcs.loc <- system.file("extdata", package="flowCore")
file.location <- paste(fcs.loc, dir(fcs.loc), sep="/"
samp <- read.flowSet(file.location[1:3])

#Get summary information about each sample.
fSApply(samp, summary)

#Obtain the median of each parameter in each frame.
fSApply(samp, each_col, median)
```

---

gateActionItem-class  
Class "gateActionItem"

Description

Class and method to capture gating operations in a flow cytometry workflow.

Usage

```r
gateActionItem(ID = paste("gateActionRef", guid(), sep = ","), name = paste("action", identifier(get(gate)), sep = ","), parentView, gate, filterResult, workflow)
```
Arguments

workflow: An object of class `workFlow` for which a view is to be created.
ID: A unique identifier of the view, most likely created by using the internal guid function.
name: A more human-readable name of the view.
parentView, gate, filterResult: References to the parent `view`, `filter`, and `filterResult` objects, respectively.

Details

gateActionItems provide a means to bind gating operations in a workflow. Each gateActionItem represents a single `filter`.

Value

A reference to the gateActionItem that is created inside the `workFlow` environment as a side effect of calling the add method.
A gateActionItem object for the constructor.

Objects from the Class

Objects should be created using the add method, which creates a gateActionItem from a `filter` object and directly assigns it to a `workFlow`. Alternatively, one can use the gateActionItem constructor function for more programmatic access.

Slots

gate: Object of class "fcFilterReference". A reference to the `filter` that is used for the gating operation.
filterResult: Object of class "fcFilterResultReference". A reference to the `filterResult` produced by the gating operation.
ID: Object of class "character". A unique identifier for the actionItem.
name: Object of class "character". A more human-readable name
parentView: Object of class "fcViewReference". A reference to the parent `view` the gateActionItem is applied on.
env: Object of class "environment". The evaluation environment in the `workFlow`.

Extends

Class "actionItem", directly.

Methods

gate signature(object = "gateActionItem"): Accessor to the gate slot. Note that this resolved the reference, i.e., the `filter` object is returned.
print signature(x = "gateActionItem"): Print details about the object.
Rm signature(symbol = "gateActionItem", envir = "workFlow", subSymbol = "character"): Remove a gateActionItem from a `workFlow`. This method is recursive and will also remove all dependent views and actionItems.
show signature(object = "gateActionItem"): Print details about the object.
summary signature(object = "gateActionItem"): Summarize the gating operation and return the appropriate `filterSummary` object.
Author(s)
Florian Hahne

See Also
workFlow, actionItem, transformActionItem, compensateActionItem, view

Examples
showClass("view")

Description
Class and method to capture the result of gating operations in a flow cytometry workflow.

Usage
gateView(workflow, ID=paste("gateViewRef", guid(), sep="_"),
        name="default", action, data, indices,
        filterResult, frEntry)

Arguments

- **workflow**: An object of class `workFlow` for which a view is to be created.
- **ID**: A unique identifier of the view, most likely created by using the internal `guid` function.
- **name**: A more human-readable name of the view.
- **data, action, filterResult**: References to the data, `filterResult`, and `actionItem` objects, respectively.
- **indices**: A logical vector of indices in the parent data.
- **frEntry**: A character vector indicating the name of the population in the `filterResult`.

Details
gateViews provide a means to bind the results of gating operations in a workflow. Each gateView represents one of the populations that arise from the gating. `logicalFilterResults` create two gateViews (events in the gate and events not in the gate), `multipleFilterResults` one view for each population. See the documentation of the parent class `view` for more details.

Value
A reference to the gateView that is created inside the `workFlow` environment as a side effect of calling the `add` method.

A gateView object for the constructor.
Objects from the Class

Objects should be created using the `add` method, which creates a gateView from a filter object and directly assigns it to a workFlow. Alternatively, one can use the gateView constructor function for more programmatic access.

Slots

- **indices**: Object of class "logical". The indices in the parent data for events that are within the filter.
- **filterResult**: Object of class "fcFilterResultReference". A reference to the outcome of the filtering operation.
- **frEntry**: Object of class "character". The population in the filterResult that corresponds to the current view. See details for further explanation.
- **ID**: Object of class "character". A unique identifier for the view.
- **name**: Object of class "character". A more human-readable name
- **action**: Object of class "fcActionReference". A reference to the actionItem that generated the view.
- **env**: Object of class "environment". The evaluation environment in the workFlow.
- **data**: Object of class "fcDataReference". A reference to the data that is associated to the view. Subsets of the data are only generated when a further action is invoked on a particular gateView. Summary statistics about the view can be acquired through the usual process of summarizing filterResults.

Extends

Class "view", directly.

Methods

- **Rm** signatue(symbol = "gateView", envir = "workFlow", subSymbol = "character"):
  Remove a gateView from a workFlow. This method is recursive and will also remove all dependent views and actionItems.
- **summary** signature(x = "formula", data = "gateView"):
  Summarize the gating operation.
- **xyplot** signature(x = "formula", data = "gateView"):
  Plot the data of the gateView along with the gate.

Author(s)

Florian Hahne

See Also

workFlow, view, transformView, compensateView, actionItem

Examples

showClass("view")
getChannelMarker

get channel and marker information from a flowFrame that matches to the given keyword

Description

This function tries best to guess the flow parameter based on the keyword supplied by name. It first does a complete word match (case insensitive) between name and flow channels and markers. If there are duplicated matches, throw the error. If no matches, it will try the partial match.

Usage

getChannelMarker(frm, name, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frm</td>
<td>flowFrame object</td>
</tr>
<tr>
<td>name</td>
<td>character the keyword to match</td>
</tr>
<tr>
<td>...</td>
<td>other arguments: not used.</td>
</tr>
</tbody>
</table>

Value

an one-row data.frame that contains "name" (i.e. channel) and "desc" (i.e. stained marker) columns.

getIndexSort

Extract Index Sorted Data from an FCS File

Description

Retrieve a data frame of index sorted data and sort indices from an FCS file.

Details

The input FCS file should already be compensated. Index sorting permits association of cell-level fluorescence intensities with downstream data collection on the sorted cells. Cells are sorted into a plate with X, Y coordinates, and those coordinates are stored in the FCS file.

This function will extract the data frame of flow data and the X, Y coordinates for the cell-level data, which can be written to a text file, or concatenated with sample-level information and analyzed in R. The coordinates are names `XLoc`, `YLoc`, and a 'name' column is also prepended with the FCS file name.

Value

Matrix of fluorescence intensities and sort indices for plate location. When no index sorting data is available, invisibly returns 0. Test for 0 to check success.

Methods

x = "flowFrame" Return a matrix of fluorescence intensities and indices into the sorting plate for each cell.
GvHD

Author(s)
G. Finak

Examples

samp <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
# This will return a message that no index sorting data is available
getIndexSort(samp)

---

GvHD    Extract of a Graft versus Host Disease monitoring experiment
        (Rizzieri et al., 2007)

Description
A flow cytometry high throughput screening was used to identify biomarkers that would predict
the development of GvHD. The GvHD dataset is an extract of a collection of weekly peripheral blood
samples obtained from patients following allogenic blood and marrow transplant. Samples were
taken at various time points before and after graft.

Usage
data(GvHD)

Format
The format is an object of class flowSet composed of 35 flowFrames. Each flowFrame corre-
sponds to one sample at one time point. The phenodata lists:

- **Patient**  The patient Id code
- **Visit**    The number of visits to the hospital
- **Days**    The number of days since the hospital
- **Grade**  Grade of the cancer

Details
This GvHD dataset represents the measurements of one biomarker (leukocyte) for 5 patients over
7 visits (7 time points). The blood samples were labeled with four different fluorescent probes to
identify the biomarker and the fluorescent intensity was determined for at least ten thousand cells
per sample.

Source
Complete dataset available at [http://www.ficcs.org/software.html#Data_Files](http://www.ficcs.org/software.html#Data_Files), the Flow Informatics and Computational Cytometry Society website (FICCS)

References
Description

Hyperlog transformation of a parameter is defined by the function

\[ f(\text{parameter}, a, b) = \sqrt{EH(y, a, b)} - \text{parameter} \]

where \( EH \) is a function defined by

\[ EH(y, a, b) = 10^{\left(\frac{y}{a}\right)} + \frac{b \cdot y}{a} - 1 \quad y \geq 0 \]

\[ -10^{\left(-\frac{y}{a}\right)} + \frac{b \cdot y}{a} + 1 \quad y < 0 \]

Objects from the Class

Objects can be created by calls to the constructor `hyperlog(parameter, a, b, transformationId)`

Slots

- `.Data`: Object of class "function" ~
- `a`: Object of class "numeric" - numeric constant greater than zero
- `b`: Object of class "numeric" numeric constant greater than zero
- `parameters`: Object of class "transformation" - flow parameter to be transformed
- `transformationId`: Object of class "character" - unique ID to reference the transformation

Extends


Methods

No methods defined with class "hyperlog" in the signature.

Note

The transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

Author(s)

Gopalakrishnan N, F.Hahne

References

Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry V 1.5
**See Also**

EHtrans

**Examples**

```r
dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
hlog1<-hyperlog("FSC-H",a=1,b=1,transformationId="hlog1")
transOut<-eval(hlog1)(exprs(dat))
```

---

**Description**

Hyperlog transformation parameterized according to Gating-ML 2.0 is defined by the following function:

\[
\text{bound}(\text{hyperlog}, \text{boundMin}, \text{boundMax}) = \max(\min(\text{hyperlog}, \text{boundMax}), \text{boundMin})
\]

where

\[
\text{hyperlog}(x, T, W, M, A) = \root(\text{EH}(y, T, W, M, A) - x)
\]

and \(\text{EH}\) is defined as:

\[
\text{EH}(y, T, W, M, A) = ae^{by} + cy - f
\]

where

- \(x\) is the value that is being transformed (an FCS dimension value). Typically, \(x\) is less than or equal to \(T\), although the transformation function is also defined for \(x\) greater than \(T\).
- \(y\) is the result of the transformation.
- \(T\) is greater than zero and represents the top of scale value.
- \(M\) is greater than zero and represents the number of decades that the true logarithmic scale approached at the high end of the Hyperlog scale would cover in the plot range.
- \(W\) is positive and not greater than half of \(M\) and represents the number of such decades in the approximately linear region.
- \(A\) is the number of additional decades of negative data values to be included. \(A\) shall be greater than or equal to \(-W\), and less than or equal to \(M - 2W\).
- \(\root\) is a standard root finding algorithm (e.g., Newton’s method) that finds \(y\) such as \(B(y, T, W, M, A) = x\).

and \(a, b, c\) and \(f\) are defined by means of \(T, W, M, A, w, x0, x1, x2, e0, ca\) and \(fa\) as:

\[
w = W/(M + A)
\]

\[
x2 = A/(M + A)
\]

\[
x1 = x2 + w
\]

\[
x0 = x2 + 2 \times w
\]

\[
b = (M + A) \times \ln(10)
\]
\[ e^0 = e^{bx_0} \]
\[ ca = e^0/w \]
\[ fa = e^{bx_1} + ca \times x_1 \]
\[ a = T/(e^b + ca - fa) \]
\[ c = ca \times a \]
\[ f = fa \times a \]

In addition, if a boundary is defined by the boundMin and/or boundMax parameters, then the result of this transformation is restricted to the \([\text{boundMin}, \text{boundMax}]\) interval. Specifically, should the result of the hyperlog function be less than boundMin, then let the result of this transformation be boundMin. Analogically, should the result of the hyperlog function be more than boundMax, then let the result of this transformation be boundMax. The boundMin parameter shall not be greater than the boundMax parameter.

**Objects from the Class**

Objects can be created by calls to the constructor

\[
\text{hyperlogGml2}(\text{parameter}, T, M, W, A, \text{transformationId}, \text{boundMin}, \text{boundMax})
\]

**Slots**

\[ .\text{Data}: \text{Object of class function \sim} \]
\[ T: \text{Object of class numeric -positive constant (top of scale value)} \]
\[ M: \text{Object of class numeric -positive constant (desired number of decades)} \]
\[ W: \text{Object of class numeric -positive constant that is not greater than half of M (the number of such decades in the approximately linear region)} \]
\[ A: \text{Object of class numeric -a constant that is greater than or equal to -W, and also less than or equal to M-2W. (A represents the number of additional decades of negative data values to be included.)} \]
\[ \text{parameters}: \text{Object of class "transformation" -flow parameter to be transformed} \]
\[ \text{transformationId}: \text{Object of class "character" -unique ID to reference the transformation} \]
\[ \text{boundMin}: \text{Object of class numeric -lower bound of the transformation, default -Inf} \]
\[ \text{boundMax}: \text{Object of class numeric -upper bound of the transformation, default Inf} \]

**Extends**


**Methods**

No methods defined with class `hyperlogGml2` in the signature.
Note

That hyperlogtGml2 transformation brings "reasonable" data values to the scale of [0, 1]. The transformation is somewhat similar to logicletGml2. (See Gating-ML 2.0 for detailed comparison) The hyperlog transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

Author(s)

Spidlen, J., Moore, W.

References


See Also

hyperlog, logicleTransform, transform-class, transform

Examples

myDataIn <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))
myHyperLg <- hyperlogtGml2(parameters = "FSC-H", T = 1023, M = 4.5, W = 0.5, A = 0, transformationId="myHyperLg")
transOut <- eval(myHyperLg)(exprs(myDataIn))

identifier

Retrieve the GUID of flowCore objects

Description

Retrieve the GUID (globally unique identifier) of a flowFrame that was generated by the cytometer or the identifier of a filter or filterResult given by the analyst.

Usage

identifier(object)

Arguments

object Object of class flowFrame, filter or filterResult.

Details

GUID or Globally Unique Identifier is a pseudo-random number used in software applications. While each generated GUID is not guaranteed to be unique, the total number of unique keys (2^128) is so large that the probability of the same number being generated twice is very small. Note that if no GUID has been recorded along with the FCS file, the name of the file is returned.
Value

Character vector representing the GUID or the name of the file.

Methods

- `object = "filter"` Return identifier of a filter object.
- `object = "filterReference"` Return identifier of a filterReference object.
- `object = "filterResult"` Return identifier of a filterResult object.
- `object = "transform"` Return identifier of a transform object.
- `object = "flowFrame"` Return GUID from the description slot of a flowFrame object or, alternatively, the name of the input FCS file in case none can be found. For flowFrame objects there also exists a replacement method.

Author(s)

- N. LeMeur

Examples

```r
samp <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
identifier(samp)
```

Description

inverseLogicleTransform can be used to compute the inverse of the Logicle transformation. The parameters w, t, m, a for calculating the inverse are obtained from the 'trans' input passed to the 'inverseLogicleTransform' function. (The inverseLogicleTransform method makes use of the C++ implementation of the inverse logicle transform contributed by Wayne Moore et al.)

Usage

```r
inverseLogicleTransform(transformationId, trans)
```

Arguments

- `transformationId`
  - A name to assigned to the inverse transformation. Used by the transform routines.
- `trans`
  - An object of class 'transform' created using the 'logicleTransform' function. The parameters w, t, m, a for calculating the inverse are obtained from the 'trans' input passed to the 'inverseLogicleTransform' function.
Author(s)
Wayne Moore, N. Gopalakrishnan

References

See Also
logicleTransform

Examples

data(GvHD)
samp <- GvHD[[1]]
logicle <- logicleTransform(t = 10000, w = 0.5, m = 4.5, a = 0, "logicle")
## transform FL1-H parameter using logicle transformation
after <- transform(samp, transformList('FL1-H', logicle))

## Inverse transform the logicle transformed data to retrieve the original data
invLogicle <- inverseLogicleTransform(trans = logicle)
before <- transform (after, transformList('FL1-H', invLogicle))

Description
The inverse split scale transformation is defined by the function

\[ f(\text{parameter}, r, \text{maxValue}, \text{transitionChannel}) = \frac{(\text{parameter} - b)}{a} \]

where,

\[ b = \frac{\text{transitionChannel}}{2} \]

\[ d = \frac{2 \ast \log_{10}(e) \ast r}{\text{transitionChannel}} + \log_{10}(\text{maxValue}) \]

\[ t = 10^{\log_{10}t} \]

\[ a = \frac{\text{transitionChannel}}{2 \ast t} \]

\[ \log_{10}ct = \frac{(a \ast t + b) \ast d}{r} \]

\[ c = 10^{\log_{10}ct} \]
**Objects from the Class**

Objects can be created by calls to the constructor `invsplitscale(parameters,r,maxValue,transitionChannel,transformationId)`.

**Slots**

- `.Data`: Object of class "function"
- `r`: Object of class "numeric" - a positive value indicating the range of the logarithmic part of the display
- `maxValue`: Object of class "numeric" - a positive value indicating the maximum value the transformation is applied to
- `transitionChannel`: Object of class "numeric" - non-negative value that indicates where to split the linear vs. logarithmic transformation
- `parameters`: Object of class "transformation" - flow parameter to be transformed
- `transformationId`: Object of class "character" - unique ID to reference the transformation

**Extends**


**Methods**

No methods defined with class "invsplitscale" in the signature.

**Note**

The transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

**Author(s)**

Gopalakrishnan N,F.Hahne

**References**

Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry

**See Also**

splitscale

**Examples**

```r
dat <- read.FCS(system.file("extdata","0877408774.B08",package="flowCore"))
sp1 <- invsplitscale("FSC-H",r=512,maxValue=2000,transitionChannel=512)
transOut <- eval(sp1)(exprs(dat))
```
**keyword-methods**

Methods to retrieve keywords of a flowFrame

**Description**

Accessor and replacement methods for items in the description slot (usually read in from a FCS file header). It lists the keywords and its values for a flowFrame specified by a character vector. Additional methods for function and lists exists for more programmatic access to the keywords.

**Usage**

```r
keyword(object, keyword, ...)
```

**Arguments**

- `object`: Object of class `flowFrame`.
- `keyword`: Character vector or list of potential keywords or function. If missing all keywords are returned.
- `...`: compact: logical scaler to indicate whether to hide all the cytometer instrument and laser settings from keywords.

**Details**

The `keyword` methods allow access to the keywords stored in the FCS files, either for a `flowFrame` or for a list of frames in a `flowSet`. The most simple use case is to provide a character vector or a list of character strings of keyword names. A more sophisticated version is to provide a function which has to take one mandatory argument, the value of this is the `flowFrame`. This can be used to query arbitrary information from the `flowFrames` description slot or even the raw data. The function has to return a single character string. The list methods allow to combine functional and direct keyword access. The replacement method takes a named character vector or a named list as input. R’s usual recycling rules apply when replacing keywords for a whole `flowSet`.

**Methods**

- `object = "flowFrame", keyword = "character"`  
  Returns values for all keywords from the description slot in object that match the character vector keyword.

- `object = "flowFrame", keyword = "function"`  
  Apply the function in `keyword` on the `flowFrame` object. The function needs to be able to cope with a single argument and it needs to return a single character string. A typical use case is for instance to paste together values from several different keywords or to compute some statistic on the `flowFrame` and combine it with one or several other keywords.

- `object = "flowFrame", keyword = "list"`  
  Combine characters and functions in a list to select keyword values.

- `object = "flowFrame", keyword = "missing"`  
  This is essentially an alias for `description` and returns all keyword-value pairs.

- `object = "flowSet", keyword = "list"`  
  This is a wrapper around `fsApply(object, keyword, keyword)` which essentially iterates over the frames in the `flowSet`.

- `object = "flowSet", keyword = "ANY"`  
  This first coerces the keyword (mostly a character vector) to a list and then calls the next applicable method.
Author(s)
N LeMeur, F Hahne, B Ellis

See Also
description

Examples

samp <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
keyword(samp)
keyword(samp, compact = TRUE)
keyword(samp, "FCSversion")

keyword(samp, function(x,...) paste(keyword(x, "SAMPLE ID"), keyword(x, "GUID"), sep="_"))

keyword(samp) <- list(foo="bar")
data(GvHD)
keyword(GvHD, list("GUID", cellnumber=function(x) nrow(x)))

keyword(GvHD) <- list(sample=sampleNames(GvHD))

---

**kmeansFilter-class**  Class "kmeansFilter"

Description
A filter that performs one-dimensional k-means (Lloyd-Max) clustering on a single flow parameter.

Usage

kmeansFilter(..., filterId="defaultKmeansFilter")

Arguments
... kmeansFilter are defined by a single flow parameter and an associated list of k population names. They can be given as a character vector via a named argument, or as a list with a single named argument. In both cases the name will be used as the flow parameter and the content of the list or of the argument will be used as population names, after coercing to character. For example kmeansFilter(FSC=c("a", "b", "c")) or kmeansFilter(list(SSC=1:3))
If the parameter is not fully realized, but instead is the result of a transformation operation, two arguments need to be passed to the constructor: the first one being the transform object and the second being a vector of population names which can be coerced to a character. For example

```r
kmeansFilter(tf, c("D", "E"))
```

**filterId**

An optional parameter that sets the filterId of the object. The filter can later be identified by this name.

## Details

The one-dimensional k-means filter is a multiple population filter capable of operating on a single flow parameter. It takes a parameter argument associated with two or more populations and results in the generation of an object of class multipleFilterResult. Populations are considered to be ordered such that the population with the smallest mean intensity will be the first population in the list and the population with the highest mean intensity will be the last population listed.

## Value

Returns a kmeansFilter object for use in filtering flowFrames or other flow cytometry objects.

## Extends

Class parameterFilter, directly.

Class concreteFilter, by class parameterFilter, distance 2.

Class filter, by class parameterFilter, distance3.

## Slots

- **populations**: Object of class character. The names of the k populations (or clusters) that will be created by the kmeansFilter. These names will later be used for the respective subpopulations in split operations and for the summary of the filterResult.

- **parameters**: Object of class parameters, defining a single parameter for which the data in the flowFrame is to be clustered. This may also be a transformation object.

- **filterId**: Object of class character, an identifier or name to reference the kmeansFilter object later on.

## Objects from the Class

Like all other filter objects in flowCore, kmeansFilter objects should be instantiated through their constructor kmeansFilter(). See the Usage section for details.

## Methods

- `%in%` signature(x = "flowFrame", table = "kmeansFilter"): The workhorse used to evaluate the filter on data.

  **Usage:**

  This is usually not called directly by the user, but internally by the filter methods.

- **show** signature(object = "kmeansFilter"): Print information about the filter.

  **Usage:**

  The method is called automatically whenever the object is printed on the screen.
linearTransform

Create the definition of a linear transformation function to be applied on a data set

Description

Create the definition of the linear Transformation that will be applied on some parameter via the transform method. The definition of this function is currently \( x \leftarrow a \times x + b \)

Usage

linearTransform(transformationId="defaultLinearTransform", a = 1, b = 0)
Arguments

transformationId
class string to identify the transformation

a
double that corresponds to the multiplicative factor in the equation

b
double that corresponds to the additive factor in the equation

Value

Returns an object of class transform.

Author(s)

N. LeMeur

See Also

transform-class, transform

Examples

samp <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
linearTrans <- linearTransform(transformationId="Linear-transformation", a=2, b=0)
dataTransform <- transform(samp, transformList('FSC-H',linearTrans))
Slots

.Data: Object of class function ~~

T: Object of class numeric - positive constant (top of scale value)

A: Object of class numeric - non-negative constant that is less than or equal to T; it is determining the bottom end of the transformation

parameters: Object of class "transformation" - flow parameter to be transformed

transformationId: Object of class "character" - unique ID to reference the transformation

boundMin: Object of class numeric - lower bound of the transformation, default -Inf

boundMax: Object of class numeric - upper bound of the transformation, default Inf

Extends


Methods

No methods defined with class lintGml2 in the signature.

Note

The linear transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

Author(s)

Spidlen, J.

References


See Also

linearTransform, transform-class, transform

Examples

myDataIn <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))
myLinTr1 <- lintGml2(parameters = "FSC-H", T = 1000, A = 0, transformationId="myLinTr1")
transOut <- eval(myLinTr1)(exprs(myDataIn))
Create the definition of a ln transformation function (natural logarithm) to be applied on a data set

Description

Create the definition of the ln Transformation that will be applied on some parameter via the transform method. The definition of this function is currently x<-log(x)*(r/d). The transformation would normally be used to convert to a linear valued parameter to the natural logarithm scale. Typically r and d are both equal to 1.0. Both must be positive.

Usage

```
lnTransform(transformationId="defaultLnTransform", r=1, d=1)
```

Arguments

- **transformationId**
  - character string to identify the transformation
- **r**
  - positive double that corresponds to a scale factor.
- **d**
  - positive double that corresponds to a scale factor

Value

Returns an object of class `transform`.

Author(s)

B. Ellis and N. LeMeur

See Also

`transform-class`, `transform`

Examples

```
data(GvHD)
lnTrans <- lnTransform(transformationId="ln-transformation", r=1, d=1)
ln1 <- transform(GvHD, transformList('FSC-H', lnTrans))

opar = par(mfcol=c(2, 1))
plot(density(exprs(GvHD[[1]])[,1]), main="Original")
plot(density(exprs(ln1[[1]])[,1]), main="Ln Transform")
```
logarithm-class  

Class "logarithm"

Description
Logarithmic transformation of an argument is a transformation defined by the function

\[ f(\text{parameter}, a, b) = \ln(a \ast \text{parameter}) \ast b \quad a \ast \text{parameter} > 0 \]

\[ \quad 0 \quad a \ast \text{parameter} \leq 0 \]

Objects from the Class
Objects can be created by calls to the constructor `logarithm(parameters,a,b,transformationId)`

Slots
\'.Data': Object of class "function"
\'a': Object of class "numeric" - non zero multiplicative constant
\'b': Object of class "numeric" - non zero multiplicative constant
\'parameters': Object of class "transformation" - flow parameters to be transformed
\'transformationId': Object of class "character" - unique ID to reference the transformation

Extends

Methods
No methods defined with class "logarithm" in the signature.

Note
The logarithm transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

Author(s)
Gopalakrishnan N, F.Hahne

References
Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry V 1.5

See Also
exponential, quadratic
Examples

dat <- read.FCS(system.file("extdata","0877408774.B08",package="flowCore"))
lg1<-logarithm(parameters="FSC-H",a=2,b=1,transformationId="lg1")
transOut<-eval(lg1)(exprs(dat))

logicalFilterResult-class

Class "logicalFilterResult"

Description

Container to store the result of applying a filter on a flowFrame object

Slots

subSet: Object of class "numeric"
frameId: Object of class "character" referencing the flowFrame object filtered. Used for sanity checking.
filterDetails: Object of class "list" describing the filter applied
filterId: Object of class "character" referencing the filter applied

Extends

Class "filterResult", directly. Class "filter", by class "filterResult", distance 2.

Author(s)

B. Ellis

See Also

filter

Examples

showClass("logicalFilterResult")
Description

Logicle transformation as published by Moore and Parks is defined by the following function:

\[
\text{bound}(\text{logicle}, \text{boundMin}, \text{boundMax}) = \max(\min(\text{logicle}, \text{boundMax}), \text{boundMin})
\]

where

\[
\text{logicle}(x, T, W, M, A) = \text{root}(B(y, T, W, M, A) - x)
\]

and \(B\) is a modified biexponential function:

\[
B(y, T, W, M, A) = a e^{by} - c e^{-dy} - f
\]

where

- \(x\) is the value that is being transformed (an FCS dimension value). Typically, \(x\) is less than or equal to \(T\), although the transformation function is also defined for \(x\) greater than \(T\).
- \(y\) is the result of the transformation.
- \(T\) is greater than zero and represents the top of scale value.
- \(M\) is greater than zero and represents the number of decades that the true logarithmic scale approached at the high end of the Logicle scale would cover in the plot range.
- \(W\) is non-negative and not greater than half of \(M\) and represents the number of such decades in the approximately linear region. The choice of \(W = M/2\) specifies a scale that is essentially linear over the whole range except for a small region of large data values. For situations in which values of \(W\) approaching \(M/2\) might be chosen, ordinary linear display scales will usually be more appropriate. The choice of \(W = 0\) gives essentially the hyperbolic sine function.
- \(A\) is the number of additional decades of negative data values to be included. \(A\) shall be greater than or equal to \(-W\), and less than or equal to \(M - 2W\).
- \(\text{root}\) is a standard root finding algorithm (e.g., Newton’s method) that finds \(y\) such as \(B(y, T, W, M, A) = x\).

and \(a, b, c, d\) and \(f\) are defined by means of \(T, W, M, A, w, x0, x1, x2, ca\) and \(fa\) as:

\[
w = \frac{W}{(M + A)}
\]

\[
x2 = \frac{A}{(M + A)}
\]

\[
x1 = x2 + w
\]

\[
x0 = x2 + 2 * w
\]

\[
b = (M + A) * \ln(10)
\]

and \(d\) is a constant so that

\[
2 * (\ln(d) - \ln(b)) + w * (d + b) = 0
\]

given \(b\) and \(w\), and

\[
ca = e^{x0 * (b + d)}
\]
The Logicle scale is the inverse of a modified biexponential function. It provides a Logicle display that maps scale values onto the \([0, 1]\) interval such that the data value \(T\) is mapped to 1, large data values are mapped to locations similar to an \((M + A)\)-decade logarithmic scale, and \(A\) decades of negative data are brought on scale. For implementation purposes, it is recommended to follow guidance in Moore and Parks publication.

In addition, if a boundary is defined by the boundMin and/or boundMax parameters, then the result of this transformation is restricted to the \([\text{boundMin}, \text{boundMax}]\) interval. Specifically, should the result of the logicle function be less than boundMin, then let the result of this transformation be boundMin. Analogically, should the result of the logicle function be more than boundMax, then let the result of this transformation be boundMax. The boundMin parameter shall not be greater than the boundMax parameter.

**Objects from the Class**

Objects can be created by calls to the constructor

\[
\text{logicletGml2(} \text{parameter, } T, M, W, A, \text{transformationId, boundMin, boundMax})
\]

**Slots**

- **.Data**: Object of class function
- **T**: Object of class numeric - positive constant (top of scale value)
- **M**: Object of class numeric - positive constant (desired number of decades)
- **W**: Object of class numeric - non-negative constant that is not greater than half of \(M\) (the number of such decades in the approximately linear region)
- **A**: Object of class numeric - a constant that is greater than or equal to \(-W\), and also less than or equal to \(M-2W\). (\(A\) represents the number of additional decades of negative data values to be included.)

**parameters**: Object of class "transformation" - flow parameter to be transformed

**transformationId**: Object of class "character" - unique ID to reference the transformation

**boundMin**: Object of class numeric - lower bound of the transformation, default -Inf

**boundMax**: Object of class numeric - upper bound of the transformation, default Inf

**Extends**


**Methods**

No methods defined with class logicletGml2 in the signature.
logicleTransform

Note

Please note that logicletGml2 and logicleTransform are similar transformations; however, the Gating-ML 2.0 compliant logicletGml2 brings "reasonable" data values to the scale of \([0, 1]\) while the logicleTransform scales these values to \([0, M]\).

The logicle transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

Author(s)

Spidlen, J., Moore, W.

References


See Also

logicleTransform, transform-class, transform

Examples

myDataIn <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))
myLogicle <- logicletGml2(parameters = "FSC-H", T = 1023, M = 4.5, W = 0.5, A = 0, transformationId="myLogicle")
transOut <- eval(myLogicle)(exprs(myDataIn))

logicleTransform Computes a transform using the 'logicle_transform' function

Description

Logicle transformation creates a subset of biexponentialTransform hyperbolic sine transformation functions that provides several advantages over linear/log transformations for display of flow cytometry data. (The logicleTransform method makes use of the C++ implementation of the logicle transform contributed by Wayne Moore et al.)

Usage

logicleTransform(transformationId="defaultLogicleTransform", w = 0.5, t = 262144, m = 4.5, a = 0)
estimateLogicle(x, channels,...)
logicleTransform

Arguments

transformationId

A name to assign to the transformation. Used by the transform/filter routines.

w

w is the linearization width in asymptotic decades. w should be > 0 and determines the slope of transformation at zero. w can be estimated using the equation w=(m-log10(t/abs(r)))/2, where r is the most negative value to be included in the display

T

Top of the scale data value, e.g. 10000 for common 4 decade data or 262144 for a 18 bit data range. t should be greater than zero

m

m is the full width of the transformed display in asymptotic decades. m should be greater than zero

a

Additional negative range to be included in the display in asymptotic decades. Positive values of the argument brings additional negative input values into the transformed display viewing area. Default value is zero corresponding to a Standard logicle function.

x

Input flow frame for which the logicle transformations are to be estimated.

channels

channels or markers for which the logicle transformation is to be estimated.

... other arguments:

q: a numeric type specifying quantile value, default is 0.05

Author(s)

Wayne Moore, N Gopalakrishnan

References


See Also

inverseLogicleTransform, estimateLogicle

Examples

data(GvHD)
samp <- GvHD[[1]]
## User defined logicle function
lgcl <- logicleTransform(w = 0.5, t = 10000, m = 4.5)
trans <- transformList(c("FL1-H", "FL2-H"), lgcl)
after <- transform(samp, trans)
invLgcl <- inverseLogicleTransform(trans = lgcl)
trans <- transformList(c("FL1-H", "FL2-H"), invLgcl)
before <- transform(after, trans)

## Automatically estimate the logicle transformation based on the data
lgcl <- estimateLogicle(samp, channels = c("FL1-H", "FL2-H", "FL3-H", "FL2-A", "FL4-H"))
## transform parameters using the estimated logicle transformation
after <- transform(samp, lgcl)
Class `logtGml2`

Description

Log transformation as parameterized in Gating-ML 2.0 is defined by the following function:

\[
\text{bound}(f, \text{boundMin}, \text{boundMax}) = \max(\min(f, \text{boundMax}), \text{boundMin})
\]

where

\[
f(\text{parameter}, T, M) = (1/M) \times \log_{10}(x/T) + 1
\]

This transformation provides a logarithmic display that maps scale values from the \((0, T]\) interval to the \((-\infty, 1]\) interval such that the data value \(T\) is mapped to 1 and \(M\) decades of data are mapped into the interval. Also, the limit for \(x\) going to 0 is \(-\infty\).

In addition, if a boundary is defined by the \(\text{boundMin}\) and/or \(\text{boundMax}\) parameters, then the result of this transformation is restricted to the \([\text{boundMin}, \text{boundMax}]\) interval. Specifically, should the result of the \(f\) function be less than \(\text{boundMin}\), then let the result of this transformation be \(\text{boundMin}\). Analogically, should the result of the \(f\) function be more than \(\text{boundMax}\), then let the result of this transformation be \(\text{boundMax}\). The \(\text{boundMin}\) parameter shall not be greater than the \(\text{boundMax}\) parameter.

Objects from the Class

Objects can be created by calls to the constructor

`logtGml2(\text{parameter}, T, M, \text{transformationId}, \text{boundMin}, \text{boundMax})`

Slots

- `.Data`: Object of class `function`
- \(T\): Object of class `numeric` - positive constant (top of scale value)
- \(M\): Object of class `numeric` - positive constant (number of decades)
- \(\text{parameters}\): Object of class "transformation" - flow parameter to be transformed
- \(\text{transformationId}\): Object of class "character" - unique ID to reference the transformation
- \(\text{boundMin}\): Object of class `numeric` - lower bound of the transformation, default \(-\infty\)
- \(\text{boundMax}\): Object of class `numeric` - upper bound of the transformation, default \(\infty\)

Extends


Methods

No methods defined with class `logtGml2` in the signature.
**logTransform**

**Note**

The log transformation object can be evaluated using the `eval` method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

**Author(s)**

Spidlen, J.

**References**


**See Also**

`logTransform`, `transform-class`, `transform`

**Examples**

```r
myDataIn <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))
myLogTr1 <- logtGml2(parameters = "FSC-H", T = 1023, M = 4.5, transformationId="myLogTr1")
transOut <- eval(myLogTr1)(exprs(myDataIn))
```

```
logTransform

Create the definition of a log transformation function (base specified by user) to be applied on a data set

**Description**

Create the definition of the log Transformation that will be applied on some parameter via the `transform` method. The definition of this function is currently $x<-\log(x,\logbase)*(r/d)$. The transformation would normally be used to convert to a linear valued parameter to the natural logarithm scale. Typically r and d are both equal to 1.0. Both must be positive. `logbase` = 10 corresponds to base 10 logarithm.

**Usage**

```r
logTransform(transformationId="defaultLogTransform", logbase=10, r=1, d=1)
```

**Arguments**

- `transformationId`  
  character string to identify the transformation
- `logbase`  
  positive double that corresponds to the base of the logarithm.
- `r`  
  positive double that corresponds to a scale factor.
- `d`  
  positive double that corresponds to a scale factor
manyFilterResult-class

Value

Returns an object of class transform.

Author(s)

B. Ellis, N. LeMeur

See Also

transform-class, transform

Examples

samp <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))
logTrans <- logTransform(transformationId="log10-transformation", logbase=10, r=1, d=1)
trans <- transformList("FSC-H", logTrans)
dataTransform <- transform(samp, trans)

manyFilterResult-class

Class "manyFilterResult"

Description

The result of a several related, but possibly overlapping filter results. The usual creator of this object will usually be a filter operation of filterSet object on a flowFrame object.

Slots

subSet: Object of class "matrix"
frameId: Object of class "character" referencing the flowFrame object filtered. Used for sanity checking.
filterDetails: Object of class "list" describing the filter applied
filterId: Object of class "character" referencing the filter applied
dependency: Any dependencies between the filters. Currently not used.

Extends

Class "filterResult", directly. Class "filter", by class "filterResult", distance 2.

Methods

[, [[ subsetting. If x is manyFilterResult, then x[[i]] a filterResult object. The semantics is similar to the behavior of the subsetting operators for lists.
length number of filterResult objects in the set.
names names of the filterResult objects in the set.
summary summary filterResult objects in the set.
**markernames**

**Author(s)**

B. Ellis

**See Also**

`filterResult`

**Examples**

```r
showClass("manyFilterResult")
```

---

**Description**

marker names corresponds to the `desc` column of the phenoData of the flowFrame.

**Usage**

```r
markernames(object, ...)
```

```r
## S4 method for signature 'flowFrame'
markernames(object)
```

```r
markernames(object) <- value
```

```r
## S4 replacement method for signature 'flowFrame'
markernames(object) <- value
```

```r
## S4 method for signature 'flowSet'
markernames(object)
```

```r
## S4 replacement method for signature 'flowSet'
markernames(object) <- value
```

**Arguments**

- `object` : flowFrame or flowSet
- `...` : not used
- `value` : a named list or character vector. the names corresponds to the name(channel) and actual values are the desc(marker).

**Details**

When extract marker names from a flowSet, it throws the warning if the marker names are not all the same across samples.
Value

Marker names as a character vector. The marker names for FSC, SSC and Time channels are automatically excluded in the returned value. When object is a flowSet and the marker names are not consistent across flowFrames, it returns a list of unique marker sets.

Examples

data(GvHD)
fr <- GvHD[[1]]
markernames(fr)

chnls <- c("FL1-H", "FL3-H")
markers <- c("CD15", "CD14")
names(markers) <- chnls
markernames(fr) <- markers
markernames(fr)

fs <- GvHD[1:3]
markernames(fs)

--

multipleFilterResult-class
Class "multipleFilterResult"

Description

Container to store the result of applying filter on set of flowFrame objects

Slots

subSet: Object of class "factor"
frameId: Object of class "character" referencing the flowFrame object filtered. Used for sanity checking.
fILTERDetails: Object of class "list" describing the filter applied
filterId: Object of class "character" referencing the filter applied

Extends

Class "filterResult", directly. Class "filter", by class "filterResult", distance 2.

Methods

[, [[ subsetting. If x is multipleFilterResult, then x[[i]] a FilterResult object. The semantics is similar to the behavior of the subsetting operators for lists.

length number of FilterResult objects in the set.

names names of the FilterResult objects in the set.

summary summary FilterResult objects in the set.

Author(s)

B. Ellis
Class and constructors for a filter that fits a bivariate normal distribution to a data set of paired values and selects data points according to their standard deviation from the fitted distribution.

Usage

```r
norm2Filter(x, y, method="covMcd", scale.factor=1, n=50000, filterId="defaultNorm2Filter")
```

Arguments

- `x, y` Characters giving the names of the measurement parameter on which the filter is supposed to work on. `y` can be missing in which case `x` is expected to be a character vector of length 2 or a list of characters.
- `filterId` An optional parameter that sets the `filterId` slot of this filter. The object can later be identified by this name.
- `scale.factor`, `n` Numerics of length 1, used to set the `scale.factor` and `n` slots of the object.
- `method` Character in `covMcd` or `cov.rob`, used to set the `method` slot of the object.

Details

The filter fits a bivariate normal distribution to the data and selects all events within the Mahalanobis distance multiplied by the `scale.factor` argument. The constructor `norm2Filter` is a convenience function for object instantiation. Evaluating a `curv2Filter` results in an object of class `logicalFilterResult`. Accordingly, `norm2Filters` can be used to subset and to split flow cytometry data sets.

Value

Returns a `norm2Filter` object for use in filtering `flowFrames` or other flow cytometry objects.

Extends

Class "parameterFilter", directly.
Class "concreteFilter", by class `parameterFilter`, distance 2.
Class "filter", by class `parameterFilter`, distance 3.
Slots

- **method**: One of `covMcd` or `cov.rob` defining method used for computation of covariance matrix.
- **scale.factor**: Numeric vector giving factor of standard deviations used for data selection (all points within scalefac standard deviations are selected).
- **n**: Object of class "numeric", the number of events used to compute the covariance matrix of the bivariate distribution.
- **filterId**: Object of class "character" referencing the filter.
- **parameters**: Object of class "ANY" describing the parameters used to filter the `flowFrame` or `flowSet`.

Objects from the Class

Objects can be created by calls of the form `new("norm2Filter", ...)` or using the constructor `norm2Filter`. The constructor is the recommended way of object instantiation:

Methods

- `%in%` signature(`x = "flowFrame", table = "norm2Filter"`): The workhorse used to evaluate the filter on data. This is usually not called directly by the user, but internally by calls to the `filter` methods.
- `show` signature(`object = "norm2Filter"`): Print information about the filter.

Note

See the documentation in the `flowViz` package for plotting of `norm2Filters`.

Author(s)

F. Hahne

See Also

cov.rob, CovMcd, filter for evaluation of `norm2Filters` and `split` and `Subset` for splitting and subsetting of flow cytometry data sets based on that.

Examples

```r
## Loading example data
dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))

## Create directly. Most likely from a command line
norm2Filter("FSC-H", "SSC-H", filterId="myCurv2Filter")

## To facilitate programmatic construction we also have the following
n2f <- norm2Filter(filterId="myNorm2Filter", x=list("FSC-H", "SSC-H"),
    scale.factor=2)
n2f <- norm2Filter(filterId="myNorm2Filter", x=c("FSC-H", "SSC-H"),
    scale.factor=2)

## Filtering using norm2Filter
fres <- filter(dat, n2f)
```
The result of norm2 filtering is a logical subset
Subset(dat, fres)

We can also split, in which case we get those events in and those
not in the gate as separate populations
split(dat, fres)

Description
Class and methods to normalize a flowSet using a potentially complex normalization function.

Usage

normalization(parameters, normalizationId="defaultNormalization",
normFunction, arguments=list())

normalize(data, x,...)

Arguments

parameters Character vector of parameter names.
normalizationId The identifier for the normalization object.
x An object of class flowSet.
normFunction The normalization function
arguments The list of additional arguments to normFunction
data The flowSet to normalize.
... other arguments: see normalize-methods for details.

Details
Data normalization of a flowSet is a rather fuzzy concept and the class mainly exists for method
dispatch in the workflow tools. The idea is to have a rather general function that takes a flowSet
and a list of parameter names as input and applies any kind of normalization to the respective data
columns. The output of the function has to be a flowSet again. Although we don’t formally check
for it, the dimensions of the input and of the output set should remain the same. Additional argu-
ments may be passed to the normalization function via the arguments list. Internally we evaluate
the function using do.call and one should check its documentation for details.
Currently, the most prominent example for a normalization function is warping, as provided by the
flowStats package.
Value

A normalization object for the constructor.
A flowSet for the normalize methods.

Objects from the Class

Objects should be created using the constructor normalization(). See the Usage and Arguments sections for details.

Slots

parameters: Object of class "character". The flow parameters that are supposed to be normalized by the normalization function.

normalizationId: Object of class "character". An identifier for the object.

normFunction: Object of class "function". The normalization function. It has to take two mandatory arguments: x, the flowSet, and parameters, a character of parameter names that are to be normalized by the function. Additional arguments have to be passed in via arguments.

arguments: Object of class "list". A names list of additional arguments. Can be NULL.

Methods

add signature(wf = "workFlow", action = "normalization"): The constructor for the workFlow.

identifier<- signature(object = "normalization", value = "character"): Set method for the identifier slot.

identifier signature(object = "normalization"): Get method for the identifier slot.

normalize signature(data = "flowSet", x = "normalization"): Apply a normalization to a flowSet.

parameters signature(object = "normalization"): The more generic constructor.

Author(s)

F. Hahne

normalizeActionItem-class

Class "normalizeActionItem"

Description

Class and method to capture normalization operations in a flow cytometry workflow.

Usage

normalizeActionItem(ID = paste("normActionRef", guid(), sep = "_")),
name = paste("action", identifier(get(normalization)), sep = "_"),
parentView, normalization, workflow)
normalizeActionItem-class

Arguments

- **workflow**: An object of class `workFlow` for which a view is to be created.
- **ID**: A unique identifier of the view, most likely created by using the internal `guid` function.
- **name**: A more human-readable name of the view.
- **parentView, normalization**: References to the parent `view` and `normalization` objects, respectively.

Details

`normalizeActionItems` provide a means to bind normalization operations like warping in a workflow. Each `normalizeActionItem` represents a single `normalization`.

Value

A reference to the `normalizeActionItem` that is created inside the `workFlow` environment as a side effect of calling the `add` method.

A `normalizeActionItem` object for the constructor.

Objects from the Class

Objects should be created using the `add` method, which creates a `normalizeActionItem` from a `normalization` object and directly assigns it to a `workFlow`. Alternatively, one can use the `normalizeActionItem` constructor function for more programmatic access.

Slots

- **normalization**: Object of class "fcNormalizationReference". A reference to the `normalization` object that is used for the compensation operation.
- **ID**: Object of class "character". A unique identifier for the `actionItem`.
- **name**: Object of class "character". A more human-readable name.
- **parentView**: Object of class "fcViewReference". A reference to the parent `view` the `normalizeActionItem` is applied on.
- **env**: Object of class "environment". The evaluation environment in the `workFlow`.

Extends

Class "`actionItem`", directly.

Methods

- **print signature**(`x = "normalizeActionItem"`)**: Print details about the object.
- **Rm signature**(`symbol = "normalizeActionItem", envir = "workFlow", subSymbol = "character"`)**: Remove a `normalizeActionItem` from a `workFlow`. This method is recursive and will also remove all dependent `views` and `actionItems`.
- **show signature**(`object = "normalizeActionItem"`)**: Print details about the object.

Author(s)

Florian Hahne
normalizeView-class

Class "normalizeView"

Description

Class and method to capture the result of normalization operations in a flow cytometry workflow.

Usage

normalizeView(workflow, ID=paste("normViewRef", guid(), sep="_"),
              name="default", action, data)

Arguments

workflow An object of class workflow for which a view is to be created.
ID A unique identifier of the view, most likely created by using the internal guid function.
name A more human-readable name of the view.
data, action References to the data and actionItem objects, respectively.

Value

A reference to the normalizeView that is created inside the workflow environment as a side effect of calling the add method.
A normalizeView object for the constructor.

Objects from the Class

Objects should be created using the add method, which creates a normalizeView from a normalization object and directly assigns it to a workflow. Alternatively, one can use the normalizeView constructor function for more programmatic access.

Slots

ID: Object of class "character". A unique identifier for the view.
name: Object of class "character". A more human-readable name
action: Object of class "fcActionReference". A reference to the actionItem that generated the view.
env: Object of class "environment". The evaluation environment in the workflow.
data: Object of class "fcDataReference" A reference to the data that is associated to the view.
Extends

Class "view", directly.

Methods

Rm signature(symbol = "normalizeView", envir = "workFlow", subSymbol = "character"): Remove a normalizeView from a workFlow. This method is recursive and will also remove all dependent views and actionItems.

Author(s)

Florian Hahne

See Also

workFlow, view, gateView, transformView, compensateView, actionItem

Examples

showClass("view")
Obtain information about parameters for flow cytometry objects.

**Description**

Many different objects in flowCore are associated with one or more parameters. This includes filter, flowFrame and parameterFilter objects that all either describe or use parameters.

**Usage**

```
parameters(object, ...)
```

**Arguments**

- **object**: Object of class filter, flowFrame or parameterFilter.
- **...**: Further arguments that get passed on to the methods.

**Value**

When applied to a flowFrame object, the result is an AnnotatedDataFrame describing the parameters recorded by the cytometer. For other objects it will usually return a vector of names used by the object for its calculations.

**Methods**

- **object = "filter"** Returns for all objects that inherit from filter a vector of parameters on which a gate is defined.
- **object = "parameterFilter"** see above
- **object = "setOperationFilter"** see above
- **object = "filterReference"** see above
- **object = "flowFrame"** Returns an AnnotatedDataFrame containing detailed descriptions about the measurement parameters of the flowFrame. For flowFrame objects there also exists a replacement method.

**Author(s)**

B. Ellis, N. Le Meur, F. Hahne

**Examples**

```r
samp <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
parameters(samp)
print(samp@parameters@data)
```
**parameters-class**

*Class “parameters”*

**Description**

A representation of flow parameters that allows for referencing.

**Objects from the Class**

Objects will be created internally whenever necessary and this should not be of any concern to the user.

**Slots**

- **.Data**: A list of the individual parameters

**Extends**

Class "list", from data part. Class "vector", by class "list", distance 2.

**Methods**

No methods defined with class "parameters" in the signature.

**Author(s)**

Nishant Gopalakrishnan

---

**parameterTransform-class**

*Class “parameterTransform”*

**Description**

Link a transformation to particular flow parameters

**Objects from the Class**

Objects are created by using the %on% operator and are usually not directly instantiated by the user.

**Slots**

- **.Data**: Object of class "function", the transformation function.
- **parameters**: Object of class "character" The parameters the transformation is applied to.
- **transformationId**: Object of class "character". The identifier for the object.

**Extends**

Class "transform", directly. Class "function", by class "transform", distance 2.
polygonGate-class

Methods

% on % signature(e1 = "filter", e2 = "parameterTransform"): Apply the transformation.
% on % signature(e1 = "parameterTransform", e2 = "flowFrame"): see above

parameters signature(object = "parameterTransform"): Accessor to the parameters slot

Author(s)

Byron Ellis

polygonGate-class Class “polygonGate”

Description

Class and constructor for 2-dimensional polygonal filter objects.

Usage

polygonGate(..., .gate, boundaries, filterId="defaultPolygonGate")

Arguments

filterId An optional parameter that sets the filterId of this gate.
.gate, boundaries A definition of the gate. This can be either a list or a named matrix as described below. Note the argument boundaries is deprecated and will go away in the next release.
... You can also directly describe a gate without wrapping it in a list or matrix, as described below.

Details

Polygons are specified by the coordinates of their vertices in two dimensions. The constructor is designed to be useful in both direct and programmatic usage. It takes either a list or a named matrix with 2 columns and at least 3 rows containing these coordinates. Alternatively, vertices can be given as named arguments, in which case the function tries to convert the values into a matrix.

Value

Returns a polygonGate object for use in filtering flowFrames or other flow cytometry objects.

Extends

Class “parameterFilter”, directly.
Class "concreteFilter", by class parameterFilter, distance 2.
Class "filter", by class parameterFilter, distance 3.
**Slots**

boundaries: Object of class "matrix". The vertices of the polygon in two dimensions. There need to be at least 3 vertices specified for a valid polygon.

parameters: Object of class "character". Describing the parameter used to filter the flowFrame.

filterId: Object of class "character", referencing the filter.

**Objects from the Class**

Objects can be created by calls of the form `new("polygonGate", ...)` or by using the constructor `polygonGate`. Using the constructor is the recommended way of object instantiation.

**Methods**

```
%in% signature(x = "flowFrame", table = "polygonGate"): The workhorse used to evaluate the filter on data. This is usually not called directly by the user, but internally by calls to the filter methods.
```

```
show signature(object = "polygonGate"): Print information about the filter.
```

**Note**

See the documentation in the flowViz package for plotting of polygonGates.

**Author(s)**

F.Hahne, B. Ellis N. Le Meur

**See Also**

`flowFrame, rectangleGate, ellipsoidGate, polytopeGate, filter` for evaluation of rectangleGates and `split` and `Subset` for splitting and subsetting of flow cytometry data sets based on that.

**Examples**

```r
## Loading example data
dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))

## Defining the gate
sqrcut <- matrix(c(300,300,600,600,50,300,300,50),ncol=2,nrow=4)
colnames(sqrcut) <- c("FSC-H","SSC-H")
pg <- polygonGate(filterId="nonDebris", boundaries= sqrcut)
pg

## Filtering using polygonGates
fres <- filter(dat, pg)
fres
summary(fres)

## The result of polygon filtering is a logical subset
Subset(dat, fres)

## We can also split, in which case we get those events in and those
## not in the gate as separate populations
```
split(dat, fres)

polytopeGate-class  Define filter boundaries

Description
Convenience methods to facilitate the construction of filter objects

Usage
polytopeGate(..., .gate, b, filterId="defaultPolytopeGate")

Arguments
filterId  An optional parameter that sets the filterId of this gate.
.gate  A definition of the gate. This can be either a list, vector or matrix, described below.
b  Need documentation
...  You can also directly describe a gate without wrapping it in a list or matrix, as described below.

Details
These functions are designed to be useful in both direct and programmatic usage.
For rectangle gate in n dimensions, if n=1 the gate correspond to a range gate. If n=2, the gate is a rectangle gate. To use this function programmatically, you may either construct a list or you may construct a matrix with n columns and 2 rows. The first row corresponds to the minimal value for each parameter while the second row corresponds to the maximal value for each parameter. The names of the parameters are taken from the column names as in the third example.
Rectangle gate objects can also be multiplied together using the * operator, provided that both gate have orthogonal axes.
For polygon gate, the boundaries are specified as vertices in 2 dimensions, for polytope gate objects as vertices in n dimensions.
For quadrant gates, the boundaries are specified as a named list or vector of length two.

Value
Returns a rectangleGate or polygonGate object for use in filtering flowFrames or other flow cytometry objects.

Author(s)
F.Hahne, B. Ellis N. Le Meur

See Also
flowFrame, filter
**Description**

Class and constructors for quadrant-type `filter` objects.

**Usage**

```r
quadGate(..., .gate, filterId="defaultQuadGate")
```

**Arguments**

- `filterId` An optional parameter that sets the `filterId` of this `filter`. The object can later be identified by this name.
- `.gate` A definition of the gate for programmatic access. This can be either a named list or a named numeric vector, as described below.
- `...` The parameters of `quadGates` can also be directly described using named function arguments, as described below.

**Details**

`quadGates` are defined by two parameters, which specify a separation of a two-dimensional parameter space into four quadrants. The `quadGate` function is designed to be useful in both direct and programmatic usage:

For the interactive use, these parameters can be given as additional named function arguments, where the names correspond to valid parameter names in a `flowFrame` or `flowSet`. For a more programmatic approach, a named list or numeric vector of the gate boundaries can be passed on to the function as argument `.gate`.

Evaluating a `quadGate` results in four sub-populations, and hence in an object of class `multipleFilterResult`. Accordingly, `quadGates` can be used to split flow cytometry data sets.

**Value**

Returns a `quadGate` object for use in filtering `flowFrames` or other flow cytometry objects.

**Extends**

Class "parameterFilter", directly.
Class "concreteFilter", by class `parameterFilter`, distance 2.
Class "filter", by class `parameterFilter`, distance 3.

**Slots**

- `boundary`: Object of class "numeric", length 2. The boundaries of the quadrant regions.
- `parameters`: Object of class "character", describing the parameter used to filter the `flowFrame`.
- `filterId`: Object of class "character", referencing the gate.
Objects from the Class

Objects can be created by calls of the form `new("quadGate", ...)` or using the constructor `quadGate`. The latter is the recommended way of object instantiation:

Methods

- `%in%` signature(x = "flowFrame", table = "quadGate"): The workhorse used to evaluate the gate on data. This is usually not called directly by the user, but internally by calls to the `filter` methods.
- `show` signature(object = "quadGate"): Print information about the gate.

Note

See the documentation in the `flowViz` package for plotting of `quadGates`.

Author(s)

F.Hahne, B. Ellis N. Le Meur

See Also

`flowFrame, flowSet, filter` for evaluation of `quadGates` and `split` for splitting of flow cytometry data sets based on that.

Examples

```r
## Loading example data
dat <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))

## Create directly. Most likely from a command line
quadGate(filterId="myQuadGate1", "FSC-H"=100, "SSC-H"=400)

## To facilitate programmatic construction we also have the following
quadGate(filterId="myQuadGate2", list("FSC-H"=100, "SSC-H"=400))
## FIXME: Do we want this?
##quadGate(filterId="myQuadGate3", .gate=c("FSC-H"=100, "SSC-H"=400))

## Filtering using quadGates
qg <- quadGate(filterId="quad", "FSC-H"=600, "SSC-H"=400)
fres <- filter(dat, qg)
fres
summary(fres)
names(fres)

## The result of quadGate filtering are multiple sub-populations
## and we can split our data set accordingly
split(dat, fres)

## We can limit the splitting to one or several sub-populations
split(dat, fres, population="FSC-H-SSC-H-")
split(dat, fres, population=list(keep=c("FSC-H-SSC-H-", "FSC-H-SSC-H+")))
```
Description

Quadratic transform class defines a transformation defined by the function

\[ f(\text{parameter}, a) = a \times \text{parameter}^2 \]

Objects from the Class

Objects can be created by calls to the constructor `quadratic(parameters, a, transformationId)`

Slots

- `.Data`: Object of class "function"
- `a`: Object of class "numeric" - non-zero multiplicative constant
- `parameters`: Object of class "transformation" - flow parameter to be transformed
- `transformationId`: Object of class "character" - unique ID to reference the transformation

Extends


Methods

No methods defined with class "quadratic" in the signature.

Note

The quadratic transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a column vector. (See example below)

Author(s)

Gopalakrishnan N, F.Hahne

References

Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry V 1.5

See Also

dg1polynomial, ratio, squareroot
Examples

dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
quad1<-quadratic(parameters="FSC-H",a=2,transformationId="quad1")
transOut<-eval(quad1)(exprs(dat))

Examples

dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
quad1<-quadratic(parameters="FSC-H",a=2,transformationId="quad1")
transOut<-eval(quad1)(exprs(dat))

quadraticTransform Create the definition of a quadratic transformation function to be applied on a data set

Description

Create the definition of the quadratic Transformation that will be applied on some parameter via the transform method. The definition of this function is currently x <- a*x^2 + b*x + c

Usage

quadraticTransform(transformationId="defaultQuadraticTransform", a = 1, b = 1, c = 0)

Arguments

transformationId character string to identify the transformation

a double that corresponds to the quadratic coefficient in the equation

b double that corresponds to the linear coefficient in the equation

c double that corresponds to the intercept in the equation

Value

Returns an object of class transform.

Author(s)

N. Le Meur

See Also

transform-class, transform

Examples

samp <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))
quadTrans <- quadraticTransform(transformationId="Quadratic-transformation", a=1, b=1, c=0)
dataTransform <- transform(samp, transformList('FSC-H', quadTrans))
randomFilterResult-class

Class "randomFilterResult"

Description

Container to store the result of applying a filter on a flowFrame object

Slots

- subSet: Object of class "numeric"
- frameId: Object of class "character" referencing the flowFrame object filtered. Used for sanity checking.
- filterDetails: Object of class "list" describing the filter applied
- filterId: Object of class "character" referencing the filter applied

Extends

Class "filterResult", directly. Class "filter", by class "filterResult", distance 2.

Author(s)

B. Ellis

See Also

filter

d ratio-class

Class "ratio"

Description

ratio transform calculates the ratio of two parameters defined by the function

\[
f(parameter_1, parameter_2) = \frac{parameter_1}{parameter_2}\]

Objects from the Class

Objects can be created by calls to the constructor \(\text{ratio}(\text{parameter}_1, \text{parameter}_2, \text{transformationId})\).

Slots

- .Data: Object of class "function" ~
- numerator: Object of class "transformation" -flow parameter to be transformed
- denominator: Object of class "transformation" -flow parameter to be transformed
- transformationId: Object of class "character" unique ID to reference the transformation
Class "transformation", by class "transform", distance 2. Class "characterOrTransformation", by class "transform", distance 3.

Methods

No methods defined with class "ratio" in the signature.

Note

The ratio transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as matrix with one column. (See example below)

Author(s)

Gopalakrishnan N, F.Hahne

References

Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry V 1.5

See Also

d1polynomial, quadratic, squareroot

Examples

dat <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))
rat1 <- ratio("FSC-H", "SSC-H", transformationId="rat1")
transOut <- eval(rat1)(exprs(dat))

ratiotGml2-class

Class "ratiotGml2"

Description

Ratio transformation as parameterized in Gating-ML 2.0 is defined by the following function:

\[
\text{bound}(f, \text{boundMin}, \text{boundMax}) = \max(\min(f, \text{boundMax}), \text{boundMin})
\]

where

\[
f(p1, p2, A, B, C) = A \times (p1 - B)/(p2 - C)
\]

If a boundary is defined by the boundMin and/or boundMax parameters, then the result of this transformation is restricted to the [boundMin,boundMax] interval. Specifically, should the result of the f function be less than boundMin, then let the result of this transformation be boundMin. Analogically, should the result of the f function be more than boundMax, then let the result of this transformation be boundMax. The boundMin parameter shall not be greater than the boundMax parameter.
Objects from the Class

Objects can be created by calls to the constructor

\begin{verbatim}
ratiotGml2(p1, p2, A, B, C, transformationId, boundMin, boundMax)
\end{verbatim}

Slots

- **.Data**: Object of class `function``
- **numerator**: Object of class "transformation"-flow parameter to be used as numerator in the transformation function
- **denominator**: Object of class "transformation"-flow parameter to be used as denominator in the transformation function
- **pA**: Object of class numeric constant A
- **pB**: Object of class numeric constant B
- **pC**: Object of class numeric constant C
- **transformationId**: Object of class "character"-unique ID to reference the transformation
- **boundMin**: Object of class numeric -lower bound of the transformation, default -Inf
- **boundMax**: Object of class numeric -upper bound of the transformation, default Inf

Extends


Methods

No methods defined with class "ratiotGml2" in the signature.

Note

The ratiotGml2 transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as matrix with one column. (See example below)

Author(s)

Spidlen, J.

References

Gating-ML 2.0: International Society for Advancement of Cytometry (ISAC) standard for representing gating descriptions in flow cytometry.  
http://flowcyt.sourceforge.net/gating/20141009.pdf

See Also

ratio, transform-class, transform
Examples

```r
myDataIn <- read.FCS(system.file("extdata", "0877408774.B08", 
    package="flowCore"))
myRatioT <- ratiotGml2("FSC-H", "SSC-H", pA = 2, pB = 3, 
    pC = -10, transformationId = "myRatioT")
transOut <- eval(myRatioT)(exprs(myDataIn))
```

---

**read.FCS**

*Read an FCS file*

**Description**

Check validity and Read Data File Standard for Flow Cytometry

**Usage**

```r
isFCSfile(files)
read.FCS(filename, transformation="linearize", which.lines=NULL, 
    alter.names=FALSE, column.pattern=NULL, invert.pattern = FALSE, 
    decades=0, ncdf = FALSE, min.limit=NULL, truncate_max_range = TRUE, dataset=NULL, emptyValue=TRUE, ...)
```

**Arguments**

- **files**: A vector of filenames
- **filename**: Character of length 1: filename
- **transformation**: An character string that defines the type of transformation. Valid values are `linearize` (default), `linearize-with-PnG-scaling`, or `scale`. The `linearize` transformation applies the appropriate power transform to the data. The `linearize-with-PnG-scaling` transformation applies the appropriate power transform for parameters stored on log scale, and also a linear scaling transformation based on the 'gain' (FCS `$\text{PnG}$ keywords) for parameters stored on a linear scale. The `scale` transformation scales all columns to $[0,10^{\text{decades}}]$. defaulting to decades=0 as in the FCS4 specification. A logical can also be used: `TRUE` is equal to linearize and `FALSE` (or `NULL`) corresponds to no transformation. Also when the transformation keyword of the `FCS` header is set to "custom" or "applied", no transformation will be used.
- **which.lines**: Numeric vector to specify the indices of the lines to be read. If `NULL` all the records are read, if of length 1, a random sample of the size indicated by `which.lines` is read in.
- **alter.names**: boolean indicating whether or not we should rename the columns to valid R names using `make.names`. The default is `FALSE`.
- **column.pattern**: An optional regular expression defining parameters we should keep when loading the file. The default is `NULL`.
- **invert.pattern**: logical. By default, `FALSE`. If `TRUE`, inverts the regular expression specified in `column.pattern`. This is useful for indicating the channel names that we do not want to read. If `column.pattern` is set to `NULL`, this argument is ignored.
decades When scaling is activated, the number of decades to use for the output.
ncdf Deprecated. Please use `ncdfFlow` package for cdf based storage.
min.limit The minimum value in the data range that is allowed. Some instruments produce
extreme artifactual values. The positive data range for each parameter is completely
defined by the measurement range of the instrument and all larger values
are set to this threshold. The lower data boundary is not that well defined, since
compensation might shift some values below the original measurement range of
the instrument. The default value of -111 copies the behavior of flowJo. It can
be set to an arbitrary number or to NULL, in which case the original values are
kept.
truncate_max_range logical type. Default is TRUE, can be optionally turned off to avoid truncating
the extreme positive value to the instrument measurement range i.e.`$PnR`.
dataset The FCS file specification allows for multiple data segments in a single file.
Since the output of `read.FCS` is a single `flowFrame` we can’t automatically read
in all available sets. This parameter allows to chose one of the subsets for import.
Its value is supposed to be an integer in the range of available data sets. This
argument is ignored if there is only a single data segment in the FCS file.
emptyValue boolean indicating whether or not we allow empty value for keyword values in
TEXT segment. It affects how the double delimiters are treated. IF TRUE, the
double delimiters are parsed as a pair of start and end single delimiter for an
empty value. Otherwise, double delimiters are parsed one part of string as the
keyword value. default is TRUE.
... ignore.text.offset: whether to ignore the keyword values in TEXT segment when
they don’t agree with the HEADER. Default is FALSE, which throws the error
when such discrepancy is found. User can turn it on to ignore TEXT segment
when he is sure of the accuracy of HEADER so that the file still can be read.

Details

The function `isFCSfile` determines whether its arguments are valid FCS files.
The function `read.FCS` works with the output of the FACS machine software from a number of
vendors (FCS 2.0, FCS 3.0 and List Mode Data LMD). However, the FCS 3.0 standard includes
some options that are not yet implemented in this function. If you need extensions, please let me
know. The output of the function is an object of class `flowFrame`.

For specifications of FCS 3.0 see http://www.isac-net.org and the file ../doc/fcs3.html in
the doc directory of the package.

The `which.lines` arguments allow you to read a subset of the record as you might not want to
read the thousands of events recorded in the FCS file. It is mainly used when there is not enough
memory to read one single FCS (which probably will not happen). It will probably take more time
than reading the entire FCS (due to the multiple disk IO).

Value

`isFCSfile` returns a logical vector.
`read.FCS` returns an object of class `flowFrame` that contains the data in the `exprs` slot, the param-
eters monitored in the `parameters` slot and the keywords and value saved in the header of the FCS
file.
Author(s)
F. Hahne, N. Le Meur

See Also
read.flowSet

Examples
## a sample file
fcsFile <- system.file("extdata", "00774808774.B08", package="flowCore")

## read file and linearize values
samp <- read.FCS(fcsFile, transformation="linearize")
exprs(samp[1:3,])
description(samp)[3:6]
class(samp)

## Only read in lines 2 to 5
subset <- read.FCS(fcsFile, which.lines=2:5, transformation="linearize")
exprs(subset)

## Read in a random sample of 100 lines
subset <- read.FCS(fcsFile, which.lines=100, transformation="linearize")
nrow(subset)

---

read.FCSheader  

Read the TEXT section of a FCS file

Description
Read (part of) the TEXT section of a Data File Standard for Flow Cytometry that contains FACS keywords.

Usage
read.FCSheader(files, path=".", keyword=NULL, emptyValue = TRUE)

Arguments
- **files** Character vector of filenames.
- **path** Directory where to look for the files.
- **keyword** An optional character vector that specifies the FCS keyword to read.
- **emptyValue** see link[flowCore]{read.FCS}

Details
The function `read.FCSheader` works with the output of the FACS machine software from a number of vendors (FCS 2.0, FCS 3.0 and List Mode Data LMD). The output of the function is the TEXT section of the FCS files. The user can specify some keywords to limit the output to the information of interest.
Value

A list of character vector. Each element of the list correspond to one FCS file.

Author(s)

N.Le Meur

See Also

link[flowCore]{read.flowSet}, link[flowCore]{read.FCS}

Examples

samp <- read.FCSheader(system.file("extdata", "0877408774.B08", package="flowCore"))
samp

samp <- read.FCSheader(system.file("extdata", "0877408774.B08", package="flowCore"), keyword=c("$DATE", "$FIL"))
samp

---

read.flowSet  

Read a set of FCS files

Description

Read one or several FCS files: Data File Standard for Flow Cytometry

Usage

read.flowSet(files=NULL, path=".", pattern=NULL, phenoData, descriptions, name.keyword, alter.names=FALSE, transformation = "linearize", which.lines=NULL, column.pattern = NULL, invert.pattern = FALSE, decades=0, sep="\t", as.is=TRUE, name, ncdf=FALSE, dataset=NULL, min.limit=NULL, emptyValue=TRUE, ignore.text.offset = FALSE, ...)

Arguments

files  
Optional character vector with filenames.

path  
Directory where to look for the files.

pattern  
This argument is passed on to dir, see details.

phenoData  
An object of class AnnotatedDataFrame, character or a list of values to be extracted from the flowFrame object, see details.

descriptions  
Character vector to annotate the object of class flowSet.

name.keyword  
An optional character vector that specifies which FCS keyword to use as the sample names. If this is not set, the GUID of the FCS file is used for sample-Names, and if that is not present (or not unique), then the file names are used.
Details

There are four different ways to specify the file from which data is to be imported:

First, if the argument `phenoData` is present and is of class `AnnotatedDataFrame`, then the file names are obtained from its sample names (i.e. row names of the underlying data.frame). Also column name will be generated based on sample names if it is not there. This column is mainly used by visualization methods in flowViz. Alternatively, the argument `phenoData` can be of class `character`, in which case this function tries to read a `AnnotatedDataFrame` object from the file with that name by calling `read.AnnotatedDataFrame(file.path(path,phenoData),...)`.

In some cases the file names are not a reasonable selection criterion and the user might want to import files based on some keywords within the file. One or several keyword value pairs can be given as the `phenoData` argument in form of a named list.

Third, if the argument `phenoData` is not present and the argument `files` is not `NULL`, then `files` is expected to be a character vector with the file names.

Fourth, if neither the argument `phenoData` is present nor `files` is not `NULL`, then the file names are obtained by calling `dir(path, pattern)`.

Value

An object of class `flowSet`.

Author(s)

F. Hahne, N. Le Meur, B. Ellis
rectangleGate-class

Examples

```r
fcs.loc <- system.file("extdata", package="flowCore")
file.location <- paste(fcs.loc, dir(fcs.loc), sep="/"")
samp <- read.flowSet(file.location[1:3])
```

rectangleGate-class  Class “rectangleGate”

Description

Class and constructor for n-dimensional rectangular filter objects.

Usage

```
rectangleGate(..., .gate, filterId="defaultRectangleGate")
```

Arguments

- `filterId` An optional parameter that sets the filterId of this gate. The object can later be identified by this name.
- `.gate` A definition of the gate. This can be either a list, or a matrix, as described below.
- `...` You can also directly provide the boundaries of a rectangleGate as additional named arguments, as described below.

Details

This class describes a rectangular region in n dimensions, which is a Cartesian product of n orthogonal intervals in these dimensions. \( n=1 \) corresponds to a range gate, \( n=2 \) to a rectangle gate, \( n=3 \) corresponds to a box region and \( n>3 \) to a hyper-rectangular regions. Intervals may be open on one side, in which case the value for the boundary is supposed to be \( \text{Inf} \) or \( -\text{Inf} \), respectively. rectangleGates are inclusive, that means that events on the boundaries are considered to be in the gate.

The constructor is designed to be useful in both direct and programmatic usage. To use it programatically, you may either construct a named list or you may construct a matrix with \( n \) columns and 2 rows. The first row corresponds to the minimal value for each parameter while the second row corresponds to the maximal value for each parameter. The names of the parameters are taken from the column names or from the list names, respectively. Alternatively, the boundaries of the rectangleGate can be given as additional named arguments, where each of these arguments should be a numeric vector of length 2; the function tries to collapse these boundary values into a matrix.

Note that boundaries of rectangleGates where \( \text{min} > \text{max} \) are syntactically valid, however when evaluated they will always be empty.

rectangleGate objects can also be multiplied using the \( * \) operator, provided that both gates have orthogonal axes. This results in higher-dimensional rectangleGates. The inverse operation of subsetting by parameter name(s) is also available.

Evaluating a rectangleGate generates an object of class `logicalFilterResult`. Accordingly, rectangleGates can be used to subset and to split flow cytometry data sets.
Value

Returns a `rectangleGate` object for use in filtering `flowFrames` or other flow cytometry objects.

Extends

Class "`parameterFilter`", directly.
Class "`concreteFilter`", by class `parameterFilter`, distance 2.
Class "`filter`", by class `parameterFilter`, distance 3.

Slots

- `min, max`: Objects of class "numeric". The minimum and maximum values of the n-dimensional rectangular region.
- `parameters`: Object of class "character", indicating the parameters for which the `rectangleGate` is defined.
- `filterId`: Object of class "character", referencing the filter.

Objects from the Class

Objects can be created by calls of the form `new("rectangleGate", ...)`, by using the constructor `rectangleGate` or by combining existing `rectangleGates` using the `*` method. Using the constructor is the recommended way of object instantiation:

Methods

- `%in%` signature(x = "flowFrame", table = "rectangleGate"): The workhorse used to evaluate the filter on data. This is usually not called directly by the user, but internally by calls to the `filter` methods.
- `show` signature(object = "rectangleGate"): Print information about the filter.
- `*` signature(e1 = "rectangleGate", e2 = "rectangleGate"): combining two `rectangleGates` into one higher dimensional representation.
- `\[` signature(x = "rectangleGate", i = "character"): Subsetting of a `rectangleGate` by parameter name(s). This is essentially the inverse to `*`.

Note

See the documentation in the `flowViz` package for details on plotting of `rectangleGates`.

Author(s)

F.Hahne, B. Ellis N. Le Meur

See Also

- `flowFrame`, `polygonGate`, `ellipsoidGate`, `polytopeGate`, `filter` for evaluation of `rectangleGates` and `split` and `Subset` for splitting and subsetting of flow cytometry data sets based on that.
## Loading example data

dat <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))

# Create directly. Most likely from a command line
rectangleGate(filterId="myRectGate", "FSC-H"=c(200, 600), "SSC-H"=c(0, 400))

# To facilitate programmatic construction we also have the following
rg <- rectangleGate(filterId="myRectGate", list("FSC-H"=c(200, 600),
"SSC-H"=c(0, 400)))
mat <- matrix(c(200, 600, 0, 400), ncol=2, dimnames=list(c("min", "max"),
c("FSC-H", "SSC-H")))
rg <- rectangleGate(filterId="myRectGate", .gate=mat)

## Filtering using rectangleGates
fres <- filter(dat, rg)
fres
summary(fres)

## The result of rectangle filtering is a logical subset
Subset(dat, fres)

## We can also split, in which case we get those events in and those
## not in the gate as separate populations
split(dat, fres)

## Multiply rectangle gates
rg1 <- rectangleGate(filterId="FSC-", "FSC-H"=c(-Inf, 50))
rg2 <- rectangleGate(filterId="SSC+", "SSC-H"=c(50, Inf))
rg1 * rg2

## Subset rectangle gates
rg["FSC-H"]

## 2d rectangleGate can be coerced to polygonGate
as(rg, "polygonGate")

---

**sampleFilter-class**

### Description

This non-parameter filter selects a number of events from the primary `flowFrame`.

### Usage

```
sampleFilter(size, filterId="defaultSampleFilter")
```
Arguments

filterId An optional parameter that sets the filterId of this filter. The object can later be identified by this name.

size The number of events to select.

Details

Selects a number of events without replacement from a flowFrame.

Value

Returns a sampleFilter object for use in filtering flowFrames or other flow cytometry objects.

Extends

Class "concreteFilter", directly.

Class "filter", by class concreteFilter, distance 2.

Slots

size: Object of class "numeric". Then number of events that are to be selected.

filterId: A character vector that identifies this filter.

Objects from the Class

Objects can be created by calls of the form new("sampleFilter", ...) or using the constructor sampleFilter. The latter is the recommended way of object instantiation:

Methods

%in% signature(x = "flowFrame", table = "sampleFilter"): The workhorse used to evaluate the gate on data. This is usually not called directly by the user, but internally by calls to the filter methods.

show signature(object = "sampleFilter"): Print information about the gate.

Author(s)

B. Ellis, F.Hahne

See Also

flowFrame, filter for evaluation of sampleFilters and split and Subset for splitting and sub-setting of flow cytometry data sets based on that.

Examples

## Loading example data
dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))

#Create the filter
sf <- sampleFilter(filterId="mySampleFilter", size=500)
## Filtering using sampleFilters
fres <- filter(dat, sf)
fres
summary(fres)

## The result of sample filtering is a logical subset
Subset(dat, fres)

## We can also split, in which case we get those events in and those
## not in the gate as separate populations
split(dat, fres)

### scaleTransform

Create the definition of a scale transformation function to be applied on a data set

**Description**

Create the definition of the scale Transformation that will be applied on some parameter via the `transform` method. The definition of this function is currently \( x = \frac{(x-a)}{(b-a)} \). The transformation would normally be used to convert to a 0-1 scale. In this case, \( b \) would be the maximum possible value and \( a \) would be the minimum possible value.

**Usage**

scaleTransform(transformationId="defaultScaleTransform", a, b)

**Arguments**

- **transformationId**
  - character string to identify the transformation
- **a**
  - double that corresponds to the value that will be transformed to 0
- **b**
  - double that corresponds to the value that will be transformed to 1

**Value**

Returns an object of class `transform`.

**Author(s)**

P. Haaland

**See Also**

`transform-class`, `transform`
Examples

samp <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))
scaleTrans <- scaleTransform(transformationId="Truncate-transformation", a=1, b=10^4)
dataTransform <- transform(samp, transformList("FSC-H", scaleTrans))

setOperationFilter-class

Class "setOperationFilter"

Description

Description goes here

Slots

<table>
<thead>
<tr>
<th>filters</th>
<th>Object of class &quot;list&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>filterId</td>
<td>Object of class &quot;character&quot; referencing the filter applied</td>
</tr>
</tbody>
</table>

Extends

Class "filter", directly.

Author(s)

B. Ellis

See Also

filter

singleParameterTransform-class

Class "singleParameterTransform"

Description

A transformation that operates on a single parameter

Objects from the Class

Objects can be created by calls of the form new("singleParameterTransform", ...).

Slots

<table>
<thead>
<tr>
<th>.Data</th>
<th>Object of class &quot;function&quot;. The transformation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>Object of class &quot;transformation&quot;. The parameter to transform. Can be a derived parameter from another transformation.</td>
</tr>
<tr>
<td>transformationId</td>
<td>Object of class &quot;character&quot;. An identifier for the object.</td>
</tr>
</tbody>
</table>
Extends


Methods

No methods defined with class "singleParameterTransform" in the signature.

Author(s)

F Hahne

Examples

showClass("singleParameterTransform")

---

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Objects from the Class

Objects can be created by calls of the form new("sinht", parameters, ...). ~~ describe objects here ~~

Slots

.Data: Object of class "function" ~~

a: Object of class "numeric" - non zero constant

b: Object of class "numeric" - non zero constant

parameters: Object of class "transformation" - flow parameters to be transformed

transformationId: Object of class "character" - unique ID to reference the transformation

Extends


Methods

No methods defined with class "sinht" in the signature.

Note

The transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)
Author(s)
Gopalakrishnan N, F.Hahne

References
Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry V 1.5

See Also
asinht

Examples
```r
dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
sinh1<-asinht(parameters="FSC-H",a=1,b=2000,transformationId="sinH1")
transOut<-eval(sinh1)(exprs(dat))
```

**spillover**

*Compute a spillover matrix from a flowSet*

Description
Spillover information for a particular experiment is often obtained by running several tubes of beads or cells stained with a single color that can then be used to determine a spillover matrix for use with compensate.

When matching stain channels in x with the compensation controls, we provide a few options. If ordered, we assume the ordering of the channels in the flowSet object is the same as the ordering of the compensation-control samples. If regexpr, we use a regular expression to match the channel names with the filenames of the compensation controls. By default, we must "guess" based on the largest statistic for the compensation control (i.e., the row).

Usage
```r
## S4 method for signature 'flowSet'
spillover(x, unstained = NULL, patt = NULL, fsc = "FSC-A", ssc = "SSC-A", method = "median", stain_match = c("intensity", "ordered", "regexpr"), useNormFilt=FALSE, pregate = FALSE, plot = FALSE, ...)
```

Arguments
- **x** A flowSet of compensation beads or cells
- **unstained** The name of index of the unstained negative control
- **patt** An optional regular expression defining which parameters should be considered.
- **fsc** The name or index of the forward scatter parameter
- **ssc** The name or index of the side scatter parameter
- **method** The statistic to use for calculation. Traditionally, this has been the median so it is the default. The mean is sometimes more stable.
- **stain_match** Determines how the stain channels are matched with the compensation controls.
  See details.
useNormFilt Logical indicating whether to apply a norm2Filter first before computing the spillover
pregate logical. Should we pregate each channel before computing the spillover matrix? By default, no.
plot logical. Plots the kernel density for each channel when pregating. Displays the gate used. If pregate is set to FALSE, this argument is ignored.
Additional arguments passed to rangeGate.

Details
The algorithm used is fairly simple. First, using the scatter parameters, we restrict ourselves to the most closely clustered population to reduce the amount of debris. The selected statistic is then calculated on all appropriate parameters and the unstained values swept out of the matrix. Every sample is then normalized to $[0,1]$ with respect to the maximum value of the sample, giving the spillover in terms of a proportion of the primary channel intensity.

Value
A matrix for each of the parameters

Author(s)
B. Ellis

References

See Also
compensate

Description
Divide a flow cytometry data set into several subset according to the results of a filtering operation. There are also methods available to split according to a factor variable.

Details
The splitting operation in the context of flowFrames and flowSets is the logical extension of sub-setting. While the latter only returns the events contained within a gate, the former splits the data into the groups of events contained within and those not contained within a particular gate. This concept is extremely useful in applications where gates describe the distinction between positivity and negativity for a particular marker.

The flow data structures in FlowCore can be split into subsets on various levels:
flowFrame: row-wise splitting of the raw data. In most cases, this will be done according to the outcome of a filtering operation, either using a filter that identifies more than one sub-population
or by a logical filter, in which case the data is split into two populations: "in the filter" and "not in the filter". In addition, the data can be split according to a factor (or a numeric or character vector that can be coerced into a factor).

`flowSet`: can be either split into subsets of `flowFrames` according to a factor or a vector that can be coerced into a factor, or each individual `flowFrame` into subpopulations based on the `filters` or `filterResults` provided as a list of equal length.

Splitting has a special meaning for filters that result in `multipleFilterResults` or `manyFilterResults`, in which case simple subsetting doesn't make much sense (there are multiple populations that are defined by the gate and it is not clear which of those should be used for the subsetting operation). Accordingly, splitting of multipleFilterResults creates multiple subsets. The argument population can be used to limit the output to only one or some of the resulting subsets. It takes as values a character vector of names of the populations of interest. See the documentation of the different filter classes on how population names can be defined and the respective default values. For splitting of `logicalFilterResults`, the population argument can be used to set the population names since there is no reasonable default other than the name of the gate. The content of the argument prefix will be prepended to the population names and '+' or '-' are finally appended allowing for more flexible naming schemes.

The default return value for any of the split methods is a list, but the optional logical argument `filterSet` can be used to return a `flowSet` instead. This only applies when splitting `flowFrames`, splitting of `flowSets` always results in lists of `flowSet` objects.

**Methods**

`flowFrame` methods:

- `x = "flowFrame", f = "ANY", drop = "ANY"` Catch all input and cast an error if there is no method for `f` to dispatch to.
- `x = "flowFrame", f = "factor", drop = "ANY"` Split a `flowFrame` by a factor variable. Length of `f` should be the same as `nrow(x)`, otherwise it will be recycled, possibly leading to undesired outcomes. The optional argument `drop` works in the usual way, in that it removes empty levels from the factor before splitting.
- `x = "flowFrame", f = "character", drop = "ANY"` Coerce `f` to a factor and split on that.
- `x = "flowFrame", f = "numeric", drop = "ANY"` Coerce `f` to a factor and split on that.
- `x = "flowFrame", f = "filter", drop = "ANY"` First applies the `filter` to the `flowFrame` and then splits on the resulting `filterResult` object.
- `x = "flowFrame", f = "filterSet", drop = "ANY"` First applies the `filterSet` to the `flowFrame` and then splits on the resulting final `filterResult` object.
- `x = "flowFrame", f = "logicalFilterResult", drop = "ANY"` Split into the two subpopulations (in and out of the gate). The optional argument population can be used to control the names of the results.
- `x = "flowFrame", f = "manyFilterResult", drop = "ANY"` Split into the several subpopulations identified by the filtering operation. Instead of returning a list, the additional logical argument `codeflowSet` makes the method return an object of class `flowSet`. The optional population argument takes a character vector indicating the subpopulations to use for splitting (as identified by the population name in the `filterDetails` slot).
- `x = "flowFrame", f = "multipleFilterResult", drop = "ANY"` Split into the several subpopulations identified by the filtering operation. Instead of returning a list, the additional logical argument `codeflowSet` makes the method return an object of class `flowSet`. The optional population argument takes a character vector indicating the subpopulations to use for splitting (as identified by the population name in the `filterDetails` slot). Alternatively, this can
be a list of characters, in which case the populations for each list item are collapsed into one
flowFrame.

**flowSet** methods:

- **x = "flowSet", f = "ANY", drop = "ANY"** Catch all input and cast an error if there is no method
  for f to dispatch to.
- **x = "flowSet", f = "factor", drop = "ANY"** Split a flowSet by a factor variable. Length of f
  needs to be the same as length(x). The optional argument drop works in the usual way, in
  that it removes empty levels from the factor before splitting.
- **x = "flowSet", f = "character", drop = "ANY"** Coerce f to a factor and split on that.
- **x = "flowSet", f = "numeric", drop = "ANY"** Coerce f to a factor and split on that.
- **x = "flowSet", f = "list", drop = "ANY"** Split a flowSet by a list of filterResults (as typi-
  cally returned by filtering operations on a flowSet). The length of the list has to be equal to
  the length of the flowSet and every list item needs to be a filterResult of equal class with
  the same parameters. Instead of returning a list, the additional logical argument codeflowSet
  makes the method return an object of class flowSet. The optional population argument
  takes a character vector indicating the subpopulations to use for splitting (as identified by the
  population name in the filterDetails slot). Alternatively, this can be a list of characters,
  in which case the populations for each list item are collapsed into one flowFrame. Note that
  using the population argument implies common population names for all filterResults in
  the list and there will be an error if this is not the case.

**Author(s)**

F Hahne, B. Ellis, N. Le Meur

**Examples**

data(GvHD)
qGate <- quadGate(filterId="qg", "FSC-H"=200, "SSC-H"=400)

## split a flowFrame by a filter that creates
## a multipleFilterResult
samp <- GvHD[[1]]
fres <- filter(samp, qGate)
split(samp, qGate)

## return a flowSet rather than a list
split(samp, fres, flowSet=TRUE)

## only keep one population
names(fres)
##split(samp, fres, population="FSC-Height+SSC-Height+")

## split the whole set, only keep two populations
##split(GvHD, qGate, population=c("FSC-Height+SSC-Height+",
##  "FSC-Height-SSC-Height+"))

## now split the flowSet according to a factor
split(GvHD, pData(GvHD)$Patient)
splitscale-class

Class "splitscale"

Description

The split scale transformation class defines a transformation that has a logarithmic scale at high values and a linear scale at low values. The transition points are chosen so that the slope of the transformation is continuous at the transition points.

The split scale transformation is defined by the function

\[ f(\text{parameter}, r, \text{maxValue}, \text{transitionChannel}) = a \times \text{parameter} + b \quad \text{parameter} \leq t \]

\[ \log_{10}(c \times \text{parameter}) \times \frac{r}{d} \quad \text{parameter} > t \]

where,

\[ b = \frac{\text{transitionChannel}}{2} \]

\[ d = \frac{2 \times \log_{10}(c) \times r}{\text{transitionChannel}} + \log_{10}(\text{maxValue}) \]

\[ t = 10^{\log_{10}(t)} \]

\[ a = \frac{\text{transitionChannel}}{2 \times t} \]

\[ \log_{10}ct = \frac{(a \times t + b) \times d}{r} \]

\[ c = 10^{\log_{10}ct} \]

Objects from the Class

Objects can be created by calls to the constructor `splitscale(parameters, r, maxValue, transitionChannel, transformationId)`.

Slots

.Data: Object of class "function" ~~

r: Object of class "numeric"-a positive value indicating the range of the logarithmic part of the display

maxValue: Object of class "numeric" -a positive value indicating the maximum value the transformation is applied to

transitionChannel: Object of class "numeric" -non negative value that indicates where to split the linear vs. logarithmic transformation

parameters: Object of class "transformation" - flow parameter to be transformed

transformationId: Object of class "character"-unique ID to reference the transformation

Extends

Methods

No methods defined with class "splitscale" in the signature.

Note

The transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a matrix with a single column. (See example below)

Author(s)

Gopalakrishnan N, F.Hahne

References

Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry

See Also

invsplitscale

Examples

dat <- read.FCS(system.file("extdata","0877406774.B08",package="flowCore"))
sp1 <- splitscale("FSC-H",r=768,maxValue=10000,transitionChannel=256)
transOut <- eval(sp1)(exprs(dat))

Description

The split scale transformation described by Francis L. Battye [B15] (Figure 13) consists of a logarithmic scale at high values and a linear scale at low values with a fixed transition point chosen so that the slope (first derivative) of the transform is continuous at that point. The scale extends to the negative of the transition value that is reached at the bottom of the display.

Usage

splitScaleTransform(transformationId="defaultSplitscaleTransform", maxValue=1023, transitionChannel=64, r=192)

Arguments

transformationId

A name to assign to the transformation. Used by the transform/filter integration routines.

maxValue

Maximum value the transformation is applied to, e.g., 1023

transitionChannel

Where to split the linear versus the logarithmic transformation, e.g., 64

r

Range of the logarithm part of the display, i.e., it may be expressed as the maxChannel - transitionChannel considering the maxChannel as the maximum value to be obtained after the transformation.
Value

Returns values giving the inverse of the biexponential within a certain tolerance. This function should be used with care as numerical inversion routines often have problems with the inversion process due to the large range of values that are essentially 0. Do not be surprised if you end up with population splitting about \( w \) and other odd artifacts.

Author(s)

N. LeMeur

References


See Also

transform

Examples

data(GvHD)
ssTransform <- splitScaleTransform("mySplitTransform")
after.1 <- transform(GvHD, transformList('FSC-H', ssTransform))

opar = par(mfcol=c(2, 1))
plot(density(exprs(GvHD[[1]])[, 1]), main="Original")
plot(density(exprs(after.1[[1]])[, 1]), main="Split-scale Transform")

squareroot-class

Class “squareroot”

Description

Square root transform class defines a transformation defined by the function

\[
f(\text{parameter}, a) = \sqrt{\frac{\text{parameter}}{a}}
\]

Objects from the Class

Objects can be created by calls to the constructor squareroot(parameters, a, transformationId)

Slots

.Data: Object of class "function" ~-

a: Object of class "numeric" - non zero multiplicative constant

parameters: Object of class "transformation" - flow parameter to be transformed

transformationId: Object of class "character" - unique ID to reference the transformation
**Subset**

### Extends


### Methods

No methods defined with class "squareroot" in the signature.

### Note

The squareroot transformation object can be evaluated using the eval method by passing the data frame as an argument. The transformed parameters are returned as a column vector. (See example below)

### Author(s)

Gopalakrishnan N, F.Hahne

### References

Gating-ML Candidate Recommendation for Gating Description in Flow Cytometry

### See Also

dgl1polynomial, ratio, quadratic

### Examples

```r
dat <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))
sqrt1<-squareroot(parameters="FSC-H",a=2,transformationId="sqrt1")
transOut<-eval(sqrt1)(exprs(dat))
```

---

**Description**

An equivalent of a `subset` function for `flowFrame` or a `flowSet` object. Alternatively, the regular subsetting operators can be used for most of the topics documented here.

### Usage

```r
Subset(x, subset, ...)
```

### Arguments

- `x`: The flow object, frame or set, to subset.
- `subset`: A filter object or, in the case of `flowSet` subsetting, a named list of filters.
- `...`: Like the original `subset` function, you can also select columns.
The Subset method is the recommended method for obtaining a `flowFrame` that only contains events consistent with a particular filter. It is functionally equivalent to `frame[as(filter(frame, subset), "logical")`, when used in the `flowFrame` context. Used in the `flowSet` context, it is equivalent to using `fsApply` to apply the filtering operation to each `flowFrame`.

Additionally, using Subset on a `flowSet` can also take a named list as the subset. In this case, the names of the list object should correspond to the sampleNames of the `flowSet`, allowing a different filter to be applied to each frame. If not all of the names are used or excess names are present, a warning will be generated but the valid filters will be applied for the rare instances where this is the intended operation. Note that a `filter` operation will generate a list of `filterResult` objects that can be used directly with `Subset` in this manner.

**Value**

Depending on the original context, either a `flowFrame` or a `flowSet`.

**Author(s)**

B. Ellis

**See Also**

`split`, `subset`

**Examples**

```r
sample <- read.flowSet(path=system.file("extdata", package="flowCore"),
                       pattern="0877408774")
result <- filter(sample, rectangleGate("FSC-H"=c(-Inf, 1024)))
result
Subset(sample,result)
```

---

**Description**

Class and methods to subset a `flowSet`. This is only needed for method dispatch in the `workFlow` framework.

**Usage**

```
subsetting(indices, subsettingId="defaultSubsetting")
```
Arguments

- **indices**: Character or numeric vector of sample names.
- **subsettingId**: The identifier for the subsetting object.

Details

The class mainly existst for method dispatch in the workflow tools.

Value

A subsetting object.

Objects from the Class

Objects should be created using the constructor `subsetting()`. See the Usage and Arguments sections for details.

Slots

- **subsettingId**: Object of class "character". An identifier for the object.
- **indices**: Object of class "numericOrCharacter". Indices into a flowSet.

Methods

- **add** signature(wf = "workFlow", action = "subsetting"): The constructor for the workFlow.
- **identifier<-** signature(object = "subsetting", value = "character"): Set method for the identifier slot.
- **identifier** signature(object = "subsetting"): Get method for the identifier slot.
- **show** signature(object = "subsetting"): Show details about the object.

Author(s)

F. Hahne

---

**subsettingActionItem-class**

Class "subsettingActionItem"

Description

Class and method to capture subsetting operations in a flow cytometry workflow.

Usage

```
subsettingActionItem(ID = paste("subActionRef", guid(), sep = ","),
                    name = paste("action", identifier(get(subsetting)), sep = ","),
                    parentView, subsetting, workflow)
```
subsettingActionItem-class

Arguments

workflow An object of class workFlow for which a view is to be created.
ID A unique identifier of the view, most likely created by using the internal guid function.
name A more human-readable name of the view.
parentView, subsetting References to the parent view and subsetting objects, respectively.

Details

subsettingActionItems provide a means to bind subsetting operations in a workflow. Each subsettingActionItem represents a single subsetting.

Value

A reference to the subsettingActionItem that is created inside the workFlow environment as a side effect of calling the add method.
A subsettingActionItem object for the constructor.

Objects from the Class

Objects should be created using the add method, which creates a subsettingActionItem from a normalization object and directly assigns it to a workFlow. Alternatively, one can use the subsettingActionItem constructor function for more programmatic access.

Slots

subsetting: Object of class "fcSubsettingReference". A reference to the subsetting object that is used for the operation.
ID: Object of class "character". A unique identifier for the actionItem.
name: Object of class "character". A more human-readable name
parentView: Object of class "fcViewReference". A reference to the parent view the subsettingActionItem is applied on.
env: Object of class "environment". The evaluation environment in the workFlow.

Extends

Class "actionItem", directly.

Methods

print signature(x = "subsettingActionItem"): Print details about the object.
Rm signature(symbol = "subsettingActionItem", envir = "workFlow", subSymbol = "character"): Remove a subsettingActionItem from a workFlow. This method is recursive and will also remove all dependent views and actionItems.
show signature(object = "subsettingActionItem"): Print details about the object.

Author(s)

Florian Hahne
subsettingView-class

See Also
workFlow, actionItem, gateActionItem, transformActionItem, compensateActionItem, view

Examples
showClass("view")

subsettingView-class  Class "subsettingView"

Description
Class and method to capture the result of subsetting operations in a flow cytometry workflow.

Usage
subsettingView(workflow, ID=paste("subViewRef", guid(), sep="_"),
               name="default", action, data)

Arguments
workflow  An object of class workFlow for which a view is to be created.
ID  A unique identifier of the view, most likely created by using the internal guid function.
name  A more human-readable name of the view.
data, action References to the data and actionItem objects, respectively.

Value
A reference to the subsettingView that is created inside the workFlow environment as a side effect of calling the add method.
A subsettingView object for the constructor.

Objects from the Class
Objects should be created using the add method, which creates a subsettingView from a subsetting object and directly assigns it to a workFlow. Alternatively, one can use the subsettingView constructor function for more programmatic access.

Slots
ID: Object of class "character". A unique identifier for the view.
name: Object of class "character". A more human-readable name
action: Object of class "fcActionReference". A reference to the actionItem that generated the view.
env: Object of class "environment". The evaluation environment in the workFlow.
data: Object of class "fcDataReference" A reference to the data that is associated to the view.
summarizeFilter-methods

Extends

Class "view", directly.

Methods

Rm signature(symbol = "subsettingView", envir = "workFlow", subSymbol = "character"):
Remove a subsettingView from a workFlow. This method is recursive and will also remove all dependent views and actionItems.

Author(s)

Florian Hahne

See Also

workFlow, view, gateView, transformView, compensateView, actionItem

Examples

showClass("view")

summarizeFilter-methods

Methods for function summarizeFilter

Description

Internal methods to populate the filterDetails slot of a filterResult object.

Methods

result = "filterResult", filter = "filter" summarizeFilter methods are called during the process of filtering. Their output is a list, and it can be arbitrary data that should be stored along with the results of a filtering operation.

result = "filterResult", filter = "filterReference" see above
result = "filterResult", filter = "parameterFilter" see above
result = "filterResult", filter = "subsetFilter" see above
result = "logicalFilterResult", filter = "norm2Filter" see above
result = "logicalFilterResult", filter = "parameterFilter" see above
result = "multipleFilterResult", filter = "parameterFilter" see above
timeFilter-class

Class "timeFilter"

Description

Define a filter that removes stretches of unusual data distribution within a single parameter over time. This can be used to correct for problems during data acquisition like air bubbles or clods.

Usage

timeFilter(..., bandwidth=0.75, binSize, timeParameter, filterId="defaultTimeFilter")

Arguments

... The names of the parameters on which the filter is supposed to work on. Names can either be given as individual arguments, or as a list or a character vector.
filterId An optional parameter that sets the filterId slot of this gate. The object can later be identified by this name.
bandwidth, binSize Numerics used to set the bandwidth and binSize slots of the object.
timeParameter Character used to set the timeParameter slot of the object.

Details

Clods and disturbances in the laminar flow of a FACS instrument can cause temporal aberrations in the data acquisition that lead to artifactual values. timeFilters try to identify such stretches of disturbance by computing local variance and location estimates and to remove them from the data.

Value

Returns a timeFilter object for use in filtering flowFrames or other flow cytometry objects.

Extends

Class "parameterFilter", directly.
Class "concreteFilter", by class parameterFilter, distance 2.
Class "filter", by class parameterFilter, distance 3.

Slots

bandwidth: Object of class "numeric". The sensitivity of the filter, i.e., the amount of local variance of the signal we want to allow.
binSize: Object of class "numeric". The size of the bins used for the local variance and location estimation. If NULL, a reasonable default is used when evaluating the filter.
timeParameter: Object of class "character", used to define the time domain parameter. If NULL, the filter tries to guess the time domain from the flowFrame.
parameters: Object of class "character", describing the parameters used to filter the flowFrame.

filterId: Object of class "character", referencing the filter.

Objects from the Class

Objects can be created by calls of the form new("timeFilter", ...) or using the constructor timeFilter. Using the constructor is the recommended way of object instantiation:

Methods

%in% signature(x = "flowFrame", table = "timeFilter"): The workhorse used to evaluate the filter on data. This is usually not called directly by the user.

show signature(object = "timeFilter"): Print information about the filter.

Note

See the documentation of timeLinePlot in the flowViz package for details on visualizing temporal problems in flow cytometry data.

Author(s)

Florian Hahne

See Also

flowFrame, filter for evaluation of timeFilters and split and Subset for splitting and subsetting of flow cytometry data sets based on that.

Examples

## Loading example data
data(GvHD)
dat <- GvHD[1:10]

## create the filter
tf <- timeFilter("SSC-H", bandwidth=1, filterId="myTimeFilter")
tf

## Visualize problems
## Not run:
library(flowViz)
timelinePlot(dat, "SSC-H")

## End(Not run)

## Filtering using timeFilters
fres <- filter(dat, tf)
fres[[1]]
summary(fres[[1]])
summary(fres[[7]])

## The result of rectangle filtering is a logical subset
cleanDat <- Subset(dat, fres)
transform-class

'transform': a class for transforming flow-cytometry data by applying scale factors.

Description

Transform objects are simply functions that have been extended to allow for specialized dispatch. All of the ‘...Transform’ constructors return functions of this type for use in one of the transformation modalities.

Slots

.Data: Object of class "function"

transformationId: A name for the transformation object

Methods

summary Return the parameters

Author(s)

N LeMeur

See Also

linearTransform, lnTransform, logicleTransform, biexponentialTransform, arcsinhTransform, quadraticTransform, logTransform

Examples

```
cosTransform <- function(transformId, a=1, b=1){
  t = new("transform", .Data = function(x) cos(a*x+b))
  t@transformationId = transformId
  t
}
```
cosT <- cosTransform(transformId="CosT",a=2,b=1)
summary(cosT)

transformActionItem-class

Class "transformActionItem"

Description

Class and method to capture transformation operations in a flow cytometry workflow.

Usage

transformActionItem(ID = paste("transActionRef", guid(), sep = "_"),
    name=paste("action", identifier(get(transform)), sep = "_"),
    parentView, transform, workflow)

Arguments

workflow  An object of class workFlow for which a view is to be created.

ID  A unique identifier of the view, most likely created by using the internal guid
    function.

name  A more human-readable name of the view.

parentView, transform
    References to the parent view and transform objects, respectively.

Details

transformActionItems provide a means to bind transformation operations in a workflow. Each
transformActionItem represents a single transform.

Value

A reference to the transformActionItem that is created inside the workFlow environment as a side
effect of calling the add method.

A transformActionItem object for the constructor.

Objects from the Class

Objects should be created using the add method, which creates a transformActionItem from a
transform object and directly assigns it to a workFlow. Alternatively, one can use the transformActionItem
constructor function for more programmatic access.
transformation-class

Slots

transform: Object of class "fcTransformReference". A reference to the transform object that is used for the transformation operation.
ID: Object of class "character". A unique identifier for the actionItem.
name: Object of class "character". A more human-readable name
parentView: Object of class "fcViewReference". A reference to the parent view the transformActionItem is applied on.
env: Object of class "environment". The evaluation environment in the workFlow.

Extends

Class "actionItem", directly.

Methods

print signature(x = "transformActionItem"): Print details about the object.
Rm signature(symbol = "transformActionItem", envir = "workFlow", subSymbol = "character"): Remove a transformActionItem from a workFlow. This method is recursive and will also remove all dependent views and actionItems.
show signature(object = "transformActionItem"): Print details about the object.

Author(s)

Florian Hahne

See Also

workFlow, actionItem, gateActionItem, compensateActionItem, view

Examples

showClass("view")

d---
transformation-class Class "transformation"

d---

Description

A virtual class to abstract transformations.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "characterOrTransformation", directly.

Methods

No methods defined with class "transformation" in the signature.
transformFilter-class

A class for encapsulating a filter to be performed on transformed parameters

Description

The transformFilter class is a mechanism for including one or more variable transformations into the filtering process. Using a special case of transform we can introduce transformations inline with the filtering process eliminating the need to process flowFrame objects before applying a filter.

Objects from the Class

Objects of this type are not generally created “by hand”. They are a side effect of the use of the %on% method with a filter object on the left hand side and a transformList on the right hand side.

Slots

- transforms: A list of transforms to perform on the target flowFrame
- filter: The filter to be applied to the transformed frame
- filterId: The name of the filter (chosen automatically)

Extends

Class "filter", directly.

Author(s)

B. Ellis

See Also

"filter", "transform", transform

Examples

samp <- read.FCS(system.file("extdata", "0877408774.B08", package="flowCore"))

## Gate this object after log transforming the forward and side scatter variables
filter(samp, norm2Filter("FSC-H", "SSC-H", scale.factor=2)
  %on% transform("FSC-H"=log,"SSC-H"=log))
transformList-class  

Class "transformList"

Description

Class "transformList"

Usage

transformList(from, tfun, to=from, transformationId =
  "defaultTransformation")

Arguments

from, to  
Characters giving the names of the measurement parameter on which to trans- 
form on and into which the result is supposed to be stored. If both are equal, the 
existing parameters will be overwritten.

tfun  
A list if functions or a character vector of the names of the functions used to 
transform the data. R's recycling rules apply, so a single function can be given 
to be used on all parameters.

transformationId  
The identifier for the object.

Objects from the Class

Objects can be created by calls of the form new("transformList", ...), by calling the 
transform method with key-value pair arguments of the form key equals character and value 
equals function, or by using the constructor transformList. See below for details

Slots

transforms: Object of class "list", where each list item is of class transformMap. 
transformationId: Object of class "character", the identifier for the object.

Methods

colnames signature(x = "transformList"): This returns the names of the parameters that are 
to be transformed.

c signature(x = "transformList"): Concatenate transformLists or regular lists and transformLists.

%on% signature(e1 = "transformList", e2 = "flowFrame"): Perform a transforma-
tion using the transformList on a flowFrame or flowSet.

Author(s)

B. Ellis, F. Hahne

See Also

transform, transformMap
Examples

tl <- transformList(c("FSC-H", "SSC-H"), list(log, asinh))
colnames(tl)
c(tl, transformList("FL1-H", "linearTransform"))
data(GvHD)
transform(GvHD[[1]], tl)

transformMap-class  A class for mapping transforms between parameters

Description

This class provides a mapping between parameters and transformed parameters via a function.

Objects from the Class

Objects of this type are not usually created by the user, except perhaps in special circumstances. They are generally automatically created by the inline transform process during the creation of a transformFilter, or by a call to the transformList constructor.

Slots

  output: Name of the transformed parameter
  input: Name of the parameter to transform
  f: Function used to accomplish the transform

Methods

  show signature(object = "transformList"): Print details about the object.

Author(s)

  B. Ellis, F. Hahne

See Also

  transform, transformList

Examples

  new("transformMap", input="FSC-H", output="FSC-H", f=log)
transformReference-class

Class "transformReference"

Description

Class allowing to reference transforms, for instance as parameters.

Objects from the Class

Objects will be created internally whenever necessary and this should not be of any concern to the user.

Slots

.Data: The list of references
searchEnv: The environment into which the reference points.
transformationId: The name of the transformation

Extends


Methods

No methods defined with class "transformReference" in the signature.

Author(s)

N. Gopalakrishnan

transformView-class

Class "transformView"

Description

Class and method to capture the result of transformation operations in a flow cytometry workflow.

Usage

transformView(workflow, ID=paste("transViewRef", guid(), sep="_"), name="default", action, data)
transformView-class

Arguments

- **workflow**
  - An object of class `workFlow` for which a view is to be created.
- **ID**
  - A unique identifier of the view, most likely created by using the internal `guid` function.
- **name**
  - A more human-readable name of the view.
- **data, action**
  - References to the data and `actionItem` objects, respectively.

Value

- A reference to the `transformView` that is created inside the `workFlow` environment as a side effect of calling the add method.
- A `transformView` object for the constructor.

Objects from the Class

Objects should be created using the add method, which creates a `transformView` from a `transform` object and directly assigns it to a `workFlow`. Alternatively, one can use the `transformView` constructor function for more programmatic access.

Slots

- **ID**: Object of class "character". A unique identifier for the view.
- **name**: Object of class "character". A more human-readable name
- **action**: Object of class "fcActionReference". A reference to the `actionItem` that generated the view.
- **env**: Object of class "environment". The evaluation environment in the `workFlow`.
- **data**: Object of class "fcDataReference". A reference to the data that is associated to the view.

Extends

- Class "view", directly.

Methods

- **Rm** signature(symbol = "transformView", envir = "workFlow", subSymbol = "character")
  - Remove a `transformView` from a `workFlow`. This method is recursive and will also remove all dependent views and `actionItems`.

Author(s)

- Florian Hahne

See Also

- `workFlow`, `view`, `gateView`, `compensateView`, `actionItem`

Examples

- `showClass("view")`
truncatetransform

Create the definition of a truncate transformation function to be applied on a data set

Description
Create the definition of the truncate Transformation that will be applied on some parameter via the transform method. The definition of this function is currently x[x<a] <- a. Hence, all values less than a are replaced by a. The typical use would be to replace all values less than 1 by 1.

Usage
truncatetransform(transformationId="defaultTruncateTransform", a=1)

Arguments
transformationId
character string to identify the transformation
a
double that corresponds to the value at which to truncate

Value
Returns an object of class transform.

Author(s)
P. Haaland

See Also
transform-class, transform

Examples
samp <- read.FCS(system.file("extdata", 
"0877408774.B08", package="flowCore"))
truncatetrans <- truncatetransform(transformationId="Truncate-transformation", a=5)
dataTransform <- transform(samp, transformList('FSC-H', truncatetrans))

unitytransform-class

Class "unitytransform"

Description
Unity transform class transforms parameters names provided as characters into unity transform objects which can be evaluated to retrieve the corresponding columns from the data frame

Objects from the Class
Objects can be created by calls to the constructor unitytransform(parameters, transformationId).
validFilters

Slots

.Data: Object of class "function" ~~
parameters: Object of class "character" -flow parameters to be transformed
transformationId: Object of class "character"-unique ID to reference the transformation

Extends


Methods

No methods defined with class "unitytransform" in the signature.

Author(s)

Gopalakrishnan N, F.Hahne

See Also
dg1polynomial, ratio

Examples

dat <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
un1<-unitytransform(c("FSC-H","SSC-H"),transformationId="un1")
transOut<-eval(un1)(exprs(dat))

validFilters(flist)  
Check if all filters in a filters matches same parameters

Description

Check if all filters in a filters matches same parameters

Usage

validFilters(flist)

Arguments

flist a filters object

Value

TRUE or FALSE
**Description**

Class and method to capture the results of standard operations (called "views" here) in a flow cytometry workflow.

**Usage**

```r
view(workflow, ID=paste("viewRef", guid(), sep="_"),
     name="default", data, action)
pARENT(object)
DATA(object)
action(object)
alias(object, ...)
```

**Arguments**

- `workflow`: An object of class `workFlow` for which a view is to be created.
- `object`: An object of class `view` or one of its subclasses.
- `ID`: A unique identifier of the view, most likely created by using the internal `guid` function.
- `name`: A more human-readable name of the view.
- `data, action`: References to the data and `actionItem` objects, respectively.
- `...`: Further arguments that get passed to the generic.

**Details**

Views provide a means to bind the results of standard operations on flow cytometry data in a workflow. Each view can be considered the outcome of one operation. There are more specific subclasses for the three possible types of operation: `gateView` for gating operations, `transformView` for transformations, and `compensateView` for compensation operations. See their documentation for details.

**Value**

A reference to the view that is created inside the `workFlow` environment as a side effect of calling the constructor.

The parent view (i.e., the view based on which the current view was created) for the parent method.

**Objects from the Class**

Objects should be created using the constructor `view`, which also assigns the view to a `workFlow` object.
Slots

ID: Object of class "character". A unique identifier for the view.
name: Object of class "character". A more human-readable name
action: Object of class "fcActionReference". A reference to the actionItem that generated the view.
env: Object of class "environment". The evaluation environment in the workflow.
alias: Object of class "fcAliasReference". A reference to the alias table.
data: Object of class "fcDataReference" A reference to the data that is associated to the view.

See gateView for details on copying and subsetting of the raw data in the context of gating.

Methods

action signature(object = "view"): Accessor for the action slot. Note that this returns the actual actionItem object, i.e., the reference gets resolved.

Data signature(object = "view"): Accessor for the data slot. Note that this returns the actual data object, i.e., the reference gets resolved.

names signature(x = "view"): Accessor to the name slot.
alias signature(object = "view"): Get the alias table from a view.

parent signature(object = "view"): The parent view, i.e., the view based on which the current view was created.

print signature(x = "view"): Print details about the object.

Rm signature(symbol = "view", envir = "workflow", subSymbol = "character"): Remove a view from a workflow. This method is recursive and will also remove all dependent views and actionItems.

show signature(object = "view"): Print details about the object.

xyplot signature(x = "formula", data = "view"): Plot the data underlying the view.

xyplot signature(x = "view", data = "missing"): Plot the data underlying the view.

Author(s)

Florian Hahne

See Also

workFlow, gateView, transformView, compensateView, actionItem

Examples

showClass("view")
Class “workFlow” (deprecated)

Description

Class and methods to organize standard flow cytometry data analysis operation in a concise workflow.

Usage

```
workFlow(data, name = "default", env = new.env(parent = emptyenv()))
undo(wf, n=1)
```

Arguments

- **data**: An object of class `flowFrame` or `flowSet` for which a basic view is created.
- **name**: A more human-readable name of the view.
- **env**: Object of class `environment`. The evaluation environment used for the `workFlow`.
- **wf**: Object of class `workFlow`.
- **n**: The number of operations to undo.

Details

`workFlow` objects organize standard flow data analysis operations like gating, compensation and transformation in one single object. The user can interact with a `workFlow` object (e.g. adding operations, removing them, summarizing the results) without having to keep track of intermediate objects and names.

The integral part of a `workFlow` is an evaluation environment which holds all objects that are created during the analysis. The structure of the whole workflow is a tree, where nodes represent `link(view)`s (or results of an operation) and edges represent `actionItems` (or the operations themselves).

Value

A `workFlow` object for the constructor

Both `applyParentFilter` and `undo` are called for their side-effects.

Objects from the Class

Objects should be created using the constructor `workFlow`, which takes a `flowFrame` or `flowSet` as only mandatory input and creates a basic view for that.
Slots

name: Object of class "character". The name of the workFlow object.

tree: Object of class "fcTreeReference". A reference to the graphNEL objects representing the view structure of the workflow.

alias: Object of class "fcAliasReference". A reference to the alias table.

journal: Object of class "fcJournalReference". A reference to the journal.

env: Object of class "environment". The evaluation environment for the workflow in which all objects will be stored.

Methods

add signature(wf = "workFlow", action = "concreteFilter"): Create a new gateActionItem and gateView from a filter and assign those to the workflow.

add signature(wf = "workFlow", action = "filterList"): Create a new gateActionItem and gateView from a filterList and assign those to the workflow.

add signature(wf = "workFlow", action = "transformList"): Create a new transformActionItem and transformView from a transform and assign those to the workflow.

add signature(wf = "workFlow", action = "compensation"): Create a new compensateActionItem and compensateView from a compensation and assign those to the workflow.

assign signature(x = "ANY", value = "ANY", pos = "missing", envir = "workFlow", inherits = "missing", immediate = "missing"): Assign an object to the environment in the workFlow object and return a fcReference to it. The symbol for the object is created as a unique identifier.

assign signature(x = "missing", value = "ANY", pos = "workFlow", envir = "missing", inherits = "missing", immediate = "missing"): same as above, but provide custom symbol for the assignment.

assign signature(x = "missing", value = "ANY", pos = "missing", envir = "workFlow", inherits = "missing", immediate = "missing"): same as above, but assign object using an existing fcReference. Note that assigning NULL essentially removes the original object.

[ signature(x = "workFlow", i = "ANY"): Cast a useful error message.

[[ signature(x = "workFlow", i = "ANY"): Treat the workFlow object as a regular environment. Essentially, this is equivalent to get(x, i).

$ signature(x = "workFlow", name = "character"): Allow for list-like access. Note that completion is only available for views since all other objects in the environment are considered to be internal.

get signature(x = "character", pos = "workFlow", envir = "missing", mode = "missing", inherits = "missing"): Get an object identified by symbol x from the environment in the workFlow.

get signature(x = "character", pos = "missing", envir = "workFlow", mode = "missing", inherits = "missing"): same as above, but assign object using an existing fcReference.

ls signature(name = "workFlow", pos = "missing", envir = "missing", all.names = "missing", pattern = "missing"): List the content of the environment in the workFlow.

ls signature(name = "workFlow", pos = "missing", envir = "missing", all.names = "missing", pattern = "missing"): same as above
workFlow-class

mget signature(x = "character", envir = "workFlow", mode = "missing", ifnotfound = "missing", inherits = "missing"): Get multiple objects identified by the symbols in x from the environment in the workflow.

names signature(x = "workFlow"): List the identifiers for all views and actionItems in the workflow.

plot signature(x = "workFlow", y = "missing"): Plot the structure of the workflow tree.

Rm signature(symbol = "character", envir = "workFlow", subSymbol = "character"): Remove the object identified by the symbol symbol from the workflow.

undo signature(wf = "workFlow", n = "numeric"): Undo the last n operations on the workflow.

display signature(object = "workFlow"): Print details about the object.

summary signature(object = "workFlow"): Summarize a view in the workflow.

nodes signature(object = "workFlow"): Return a named vector of node ids where the names are the human readable names stored in the alias table.

actions signature(x = "workFlow"): List the names of the actionItems in the workflow.

views signature(x = "workFlow"): List the names of only the views in the workflow.

alias signature(object = "workFlow"): Return the alias table for the workflow.

alias signature(object = "environment"): Return the alias table from a generic environment.

The method tries to find 'fcAliasRef' among the object symbols in the environment.

journal signature(object = "workFlow"): Return the journal for the workflow.

journal signature(object = "environment"): Return the journal from a generic environment.

The method tries to find 'fcJournalRef' among the object symbols in the environment.

tree signature(object = "workFlow"): Return the tree of the workflow.

journal signature(object = "environment"): Return the tree from a generic environment.

The method tries to find 'fcTreeRef' among the object symbols in the environment.

Author(s)

Florian Hahne

See Also

"view", "actionItem"

Examples

showClass("view")
write.FCS

Write FCS file

Description
Write FCS file from a flowFrame

Usage
write.FCS(x, filename, what="numeric", delimiter = "\")

Arguments
x
A flowFrame.
filename
A character scalar giving the output file name.
what
A character scalar defining the output data type. One in integer, numeric, double. Note that forcing the data type to integer may result in considerable loss of precision if the data has been transformed. We recommend using the default data type unless disc space is an issue.
delimiter
a single character to separate the FCS keyword/value pairs. Default is back-slash: "\"

Details
The function write.FCS creates FCS 3.0 standard file from an object of class flowFrame.
For specifications of FCS 3.0 see http://www.isac-net.org and the file ../doc/fcs3.html in the doc directory of the package.

Value
A character of the file name.

Author(s)
F. Hahne

See Also
link[flowCore]{write.flowSet}

Examples
## a sample file
inFile <- system.file("extdata", "0877408774.B08", package="flowCore")
foo <- read.FCS(inFile, transform=FALSE)
outFile <- file.path(tempdir(), "foo.fcs")

## now write out into a file
write.FCS(foo, outFile)
bar <- read.FCS(outFile, transform=FALSE)
all(exprs(foo) == exprs(bar))
write.flowSet

Write an FCS file

Description

Write FCS file for each flowFrame in a flowSet

Usage

write.flowSet(x, outdir=identifier(x), filename, ...)

Arguments

x A flowSet.
filename A character scalar or vector giving the output file names. By default, the function will use the identifiers of the individual flowFrames as the file name, potentially adding the .fcs suffix unless a file extension is already present. Alternatively, one can supply either a character scalar, in which case the prefix i_ is appended (i being an integer in seq_len(length(x)), or a character vector of the same length as the flowSet x.
outdir A character scalar giving the output directory. As the default, the output of identifier(x) is used.
... Further arguments that are passed on to write.FCS.

Details

The function write.flowSet creates FCS 3.0 standard file for all flowFrames in an object of class flowSet. In addition, it will write the content of the phenoData slot in the ASCII file "annotation.txt". This file can subsequently be used to reconstruct the whole flowSet using the read.flowSet function, e.g.:

read.flowSet(path=outdir, phenoData="annotation.txt"

The function uses write.FCS for the actual writing of the FCS files.

Value

A character of the output directory.

Author(s)

F. Hahne

See Also

link[flowCore]{write.FCS}
Examples

```r
## sample data
data(GvHD)
foo <- GvHD[1:5]
outDir <- file.path(tempdir(), "foo")

## now write out into files
write.flowSet(foo, outDir)
dir(outDir)

## and read back in
bar <- read.flowSet(path=outDir, phenoData="annotation.txt")
```

Methods for Function `%on%` in Package ‘flowCore’

Description

This operator is used to construct a `transformFilter` that first applies a `transformList` to the data before applying the filter operation. You may also apply the operator to a `flowFrame` or `flowSet` to obtain transformed values specified in the list.

Author(s)

B. Ellis

Examples

```r
samp <- read.FCS(system.file("extdata","0877408774.B08", package="flowCore"))
plot(transform("FSC-H"=log, "SSC-H"=log) %on% samp)

rectangle <- rectangleGate(filterId="rectangleGateI","FSC-H"=c(4.5, 5.5))
sampFiltered <- filter(samp, rectangle %on% transform("FSC-H"=log, "SSC-H"=log))
res <- Subset(samp, sampFiltered)
plot(transform("FSC-H"=log, "SSC-H"=log) %on% res)
```
Index

!, filter-method (filter-class), 37

*Topic IO
read.FCS, 110
read.FCSHeader, 112
read.flowSet, 113
write.FCS, 152
write.flowSet, 153

*Topic classes
actionItem-class, 4
asinht-class, 7
asinhtGml2-class, 8
boundaryFilter-class, 11
compensateActionItem-class, 14
compensatedParameter-class, 16
compensateView-class, 17
compensation-class, 18
concreteFilter-class, 21
dg1polynomial-class, 22
EHtrans-class, 24
exponential-class, 27
expressionFilter-class, 29
fcReference-class, 31
filter-class, 37
filterList-class, 39
filterReference-class, 40
filterResult-class, 41
filterResultList-class, 42
filters-class, 43
filterSet-class, 44
filterSummary-class, 45
filterSummaryList-class, 47
flowFrame-class, 48
flowSet-class, 54
gateActionItem-class, 60
gateView-class, 62
hyperlog-class, 66
hyperlogGml2-class, 67
invsplitscale-class, 71
kmeansFilter-class, 74
lintGml2-class, 77
logarithm-class, 80
logicalFilterResult-class, 81
logicletGml2-class, 82
logtGml2-class, 86
manyFilterResult-class, 88
multipleFilterResult-class, 90
norm2Filter-class, 91
normalization-class, 93
normalizeActionItem-class, 94
normalizeView-class, 96
parameterFilter-class, 97
parameters-class, 99
parameterTransform-class, 99
quadGate-class, 103
quadratic-class, 105
randomFilterResult-class, 107
ratio-class, 107
ratiotGml2-class, 108
rectangleGate-class, 115
sampleFilter-class, 117
setOperationFilter-class, 120
singleParameterTransform-class, 120
sinht-class, 121
splitscale-class, 126
squareroot-class, 128
subsetting-class, 130
subsettingActionItem-class, 131
subsettingView-class, 133
timeFilter-class, 135
transform-class, 137
transformActionItem-class, 138
transformation-class, 139
transformFilter-class, 140
transformList-class, 141
transformMap-class, 142
transformReference-class, 143
transformView-class, 143
unitytransform-class, 145
view-class, 147
workFlow-class, 149

*Topic datasets
GvHD, 65

*Topic iteration
<table>
<thead>
<tr>
<th>Topic</th>
<th>Subtopic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsApply</td>
<td></td>
<td>59</td>
</tr>
<tr>
<td>*Topic manip</td>
<td></td>
<td>129</td>
</tr>
<tr>
<td></td>
<td>Subset, 129</td>
<td></td>
</tr>
<tr>
<td>*Topic methods</td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>%onX</td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>arcsinhTransform</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>biexponentialTransform</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>boundaryFilter-class</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>coerce</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>compensation-class</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>each_col</td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>ellipsoidGate-class</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>expressionFilter-class</td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>FCSTransTransform</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>filter</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>filter-and-methods</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>filter-in-methods</td>
<td></td>
<td>38</td>
</tr>
<tr>
<td>filterDetails-methods</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>getIndexSort</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>identifier</td>
<td></td>
<td>69</td>
</tr>
<tr>
<td>inverseLogicleTransform</td>
<td></td>
<td>70</td>
</tr>
<tr>
<td>keyword-methods</td>
<td></td>
<td>73</td>
</tr>
<tr>
<td>kmeansFilter-class</td>
<td></td>
<td>74</td>
</tr>
<tr>
<td>linearTransform</td>
<td></td>
<td>76</td>
</tr>
<tr>
<td>lnTransform</td>
<td></td>
<td>79</td>
</tr>
<tr>
<td>logicleTransform</td>
<td></td>
<td>84</td>
</tr>
<tr>
<td>logTransform</td>
<td></td>
<td>87</td>
</tr>
<tr>
<td>norm2Filter-class</td>
<td></td>
<td>91</td>
</tr>
<tr>
<td>normalization-class</td>
<td></td>
<td>93</td>
</tr>
<tr>
<td>parameters</td>
<td></td>
<td>98</td>
</tr>
<tr>
<td>polygonGate-class</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>polytopeGate-class</td>
<td></td>
<td>102</td>
</tr>
<tr>
<td>quadGate-class</td>
<td></td>
<td>103</td>
</tr>
<tr>
<td>quadraticTransform</td>
<td></td>
<td>106</td>
</tr>
<tr>
<td>rectangleGate-class</td>
<td></td>
<td>115</td>
</tr>
<tr>
<td>sampleFilter-class</td>
<td></td>
<td>117</td>
</tr>
<tr>
<td>scaleTransform</td>
<td></td>
<td>119</td>
</tr>
<tr>
<td>spillover</td>
<td></td>
<td>122</td>
</tr>
<tr>
<td>split-methods</td>
<td></td>
<td>123</td>
</tr>
<tr>
<td>splitScaleTransform</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>subsetting-class</td>
<td></td>
<td>130</td>
</tr>
<tr>
<td>summarizeFilter-methods</td>
<td></td>
<td>134</td>
</tr>
<tr>
<td>timeFilter-class</td>
<td></td>
<td>135</td>
</tr>
<tr>
<td>truncateTransform</td>
<td></td>
<td>145</td>
</tr>
<tr>
<td>*Topic package</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flowCore-package</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>*,rectangleGate,rectangleGate-method (rectangleGate-class)</td>
<td></td>
<td>115</td>
</tr>
<tr>
<td>&lt;,flowFrame,ANY-method (flowFrame-class)</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>&lt;=,flowFrame,ANY-method (flowFrame-class)</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>==,filterResult,flowFrame-method (filterResult-class)</td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>==,flowFrame,filterResult-method (flowFrame-class)</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>==,flowFrame,flowFrame-method (flowFrame-class)</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>&gt;,flowFrame,ANY-method (flowFrame-class)</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>&gt;=,flowFrame,ANY-method (flowFrame-class)</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>[,filterResultList,ANY-method (filterResultList-class)</td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>[,filterSet,character-method (filterSet-class)</td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>[,flowFrame,ANY-method (flowFrame-class)</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>[,flowFrame,filter-method (flowFrame-class)</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>[,flowFrame,filterResult-method (flowFrame-class)</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>[,flowSet,ANY-method (flowSet-class)</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>[,flowSet-method (flowSet-class)</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>[,multipleFilterResult,ANY-method (multipleFilterResult-class)</td>
<td></td>
<td>90</td>
</tr>
<tr>
<td>[,rectangleGate,ANY-method (rectangleGate-class)</td>
<td></td>
<td>115</td>
</tr>
<tr>
<td>[,rectangleGate,character-method (rectangleGate-class)</td>
<td></td>
<td>115</td>
</tr>
<tr>
<td>[,workFlow,ANY-method (workFlow-class)</td>
<td></td>
<td>149</td>
</tr>
<tr>
<td>[[,filterResult,ANY-method (filterResult-class)</td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>[[,filterResultList,ANY-method (filterResultList-class)</td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>[[,filterSet,character-method (filterSet-class)</td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>[[,filterSummary,character-method (filterSummary-class)</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>[[,filterSummary,numeric-method (filterSummary-class)</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>[[,flowSet,ANY-method (flowSet-class)</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>[[,flowSet-method (flowSet-class)</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>[[,logicalFilterResult,ANY-method (logicalFilterResult-class)</td>
<td></td>
<td>81</td>
</tr>
<tr>
<td>[[,manyFilterResult,ANY-method (manyFilterResult-class)</td>
<td></td>
<td>88</td>
</tr>
<tr>
<td>[[,manyFilterResult-method (manyFilterResult-class)</td>
<td></td>
<td>88</td>
</tr>
<tr>
<td>[[,multipleFilterResult,ANY-method (multipleFilterResult-class)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(filter-and-methods), 37
&filter, list-method
(filter-and-methods), 37
&list,filter-method
(filter-and-methods), 37
%on%, 51, 140, 154
action (view-class), 147
action,view-method (view-class), 147
actionItem, 15–18, 32, 61–63, 95–97, 132–134, 139, 144, 147, 148, 151
actionItem-class, 4
actionItems, 15, 95, 132, 139, 144, 149, 151
actions (workFlow-class), 149
actions,workFlow-method (workFlow-class), 149
add (workFlow-class), 149
add,workFlow,character-method (subsetting-class), 130
add,workFlow,compensation-method (workFlow-class), 149
add,workFlow,concreteFilter-method (workFlow-class), 149
add,workFlow,filterList-method (workFlow-class), 149
add,workFlow,logical-method (subsetting-class), 130
add,workFlow,normalization-method (normalization-class), 93
add,workFlow,numeric-method (subsetting-class), 130
add,workFlow,subsetting-method (subsetting-class), 130
add,workFlow,transformList-method (workFlow-class), 149
alias (view-class), 147
alias,actionItem-method (actionItem-class), 4
alias,environment-method (workFlow-class), 149
alias,view-method (view-class), 147
alias,workFlow-method (workFlow-class), 149
AnnotatedDataFrame, 49, 50, 54, 55, 98, 114
AnnotatedDataFrames, 50
apply, 23, 60
arcsinhTransform, 6, 137
as.data.frame.manyFilterResult
(manyFilterResult-class), 88
asinht, 9
asinht (asinht-class), 7
asinht-class, 7
asinhtGml2 (asinhtGml2-class), 8
asinhtGml2-class, 8
assign (fcReference-class), 31
assign,ANY,ANY,missing,workflow,missing,missing-method (workFlow-class), 149
assign,character,ANY,workFlow,missing,missing,missing-method (workFlow-class), 149
assign,fcReference,ANY,workFlow,missing,missing,missing-method (workFlow-class), 149
assign,missing,ANY,workFlow,missing,missing,missing-method (workFlow-class), 149
assign,missing,ANY,workFlow,missing,missing,missing-method (workFlow-class), 149
biexponentialTransform, 9, 84, 137
booleanGate,filter-class
(filter-class), 37
boundaryFilter (boundaryFilter-class), 11
boundaryFilter-class, 11
c,transformList-method (transformList-class), 141
call,filter-method (filter), 35
cbind2,flowFrame,matrix-method (flowFrame-class), 48
cbind2,flowFrame,numeric-method (flowFrame-class), 48
char2ExpressionFilter
(expressionFilter-class), 29
character,filter-method (filter), 35
characterOrTransformation, 7, 9, 16, 22, 24, 28, 66, 68, 72, 78, 80, 83, 86, 105, 108, 109, 121, 126, 129, 139, 143, 146
characterOrTransformation-class, 13
checkOffset, 13
cleanup (read.FCS), 110
coerce, 14
coerce,call,filter-method (coerce), 14
coerce,character,filter-method (coerce), 14
coerce,complementFilter,call-method (coerce), 14
coerce,complementFilter,logical-method (coerce), 14
coerce,ellipsoidGate,polygonGate-method (coerce), 14
coerce,environment,flowSet-method (coerce), 14
coerce,factor,filterResult-method (coerce), 14
coerce,filter,call-method (coerce), 14
INDEX

coerce, filter, logical-method (coerce), 14
coerce, filterReference, call-method
(coerce), 14
coerce, filterReference, concreteFilter-method
(coerce), 14
coerce, filterResult, logical-method
(coerce), 14
coerce, filterResultList, list-method
(coerce), 14
coerce, filterSet, list-method (coerce), 14
coerce, filterSummary, data.frame-method
(filterSummary-class), 45
coerce, flowFrame, filterSet-method
(coerce), 14
coerce, flowFrame, flowSet-method
(coerce), 14
coerce, flowSet, flowFrame-method
(coerce), 14
coerce, flowSet, list-method (coerce), 14
coerce, formula, filter-method
(coerce), 14
colnames, flowFrame-method
(flowFrame-class), 48
colnames, flowSet-method
(flowSet-class), 54
colnames<-, transformList-method
(transformList-class), 141
colnames<-, flowFrame-method
(flowFrame-class), 48
colnames<-, flowSet-method
(flowSet-class), 54
compensate, 122, 123
compensate (compensation-class), 18
compensate, flowFrame, compensation-method
(flowFrame-class), 48
compensate, flowFrame, data.frame-method
(flowFrame-class), 48
compensate, flowFrame, matrix-method
(flowFrame-class), 48
compensate, flowSet, ANY-method
(flowSet-class), 54
compensate, flowSet, data.frame-method
(flowSet-class), 54
compensateActionItem, 5, 62, 96, 133, 139,
150
compensateActionItem
(compensateActionItem-class), 14
compensateActionItem-class, 14
compensatedParameter
(compensatedParameter-class), 16
compensatedParameter-class, 16
compensateView, 63, 97, 134, 144, 147, 148, 150
compensateView (compensateView-class), 17
compensateView-class, 17
compensation, 15, 18, 32, 52, 150
compensation (compensation-class), 18
compensation-class, 18
complementFilter-class
(setOperationFilter-class), 120
concreteFilter, 11, 25, 29, 42, 75, 91, 97, 100, 103, 116, 118, 135
concreteFilter-class, 21
cov.rob, 92
CovMcd, 92
Data (view-class), 147
Data, view-method (view-class), 147
data.frames, 49
description, 73, 74
description (flowFrame-class), 48
description, flowFrame-method (flowFrame-class), 48
description<-, flowFrame, ANY-method (flowFrame-class), 48
description<-, flowFrame, list-method (flowFrame-class), 48
dg1polynomial (dg1polynomial-class), 22
dg1polynomial-class, 22
dim (flowFrame-class), 48
dim, flowFrame-method (flowFrame-class), 48
dir, 113
do.call, 93
each_col, 23, 51
each_col, flowFrame-method (each_col), 23
each_col-methods (each_col), 23
each_row (each_col), 23
each_row, flowFrame-method (each_col), 23
eah_col-methods (each_col), 23
EHtrans (EHtrans-class), 24
EHtrans-class, 24
ellipsoidGate, 25, 101, 116
ellipsoidGate (ellipsoidGate-class), 25
ellipsoidGate, filter-class (filter-class), 37
ellipsoidGate-class, 25
environment, 55, 57
estimateLogicle, 25, 85
estimateLogicle (logicleTransform), 84
estimateMedianLogicle, 27
eval, asinh, missing-method (asinh-class), 7
eval, asinhGml2, missing-method (asinhGml2-class), 8
eval, compensatedParameter, missing-method (compensatedParameter-class), 16
eval, dg1polynomial, missing-method (dg1polynomial-class), 22
eval, EHtrans, missing-method (EHtrans-class, 24
eval, exponential, missing-method (exponential-class), 27
eval, filterReference, missing-method (filterReference-class), 40
eval, hyperlog, missing-method (hyperlog-class), 66
eval, hyperlogtGml2, missing-method (hyperlogtGml2-class), 67
eval, invsplitscale, missing-method (invsplitscale-class), 71
eval, lintGml2, missing-method (lintGml2-class), 77
eval, logarithm, missing-method (logarithm-class), 80
eval, logicletGml2, missing-method (logicletGml2-class), 82
eval, logGml2, missing-method (logGml2-class), 86
val, quadratic, missing-method (quadratic-class), 105
eval, ratio, missing-method (ratio-class), 107
eval, ratiotGml2, missing-method (ratiotGml2-class), 108
eval, sinh, missing-method (sinh-class), 121
eval, sqrt, missing-method (sqrt-class), 126
val, squareroot, missing-method (squareroot-class), 128
eval, transformReference, missing-method (transformReference-class), 143
eval, unitytransform, missing-method (unitytransform-class), 145
exponential (exponential-class), 27
exponential-class, 27
expressionFilter, 29
expressionFilter (expressionFilter-class), 29
expressionFilter-class, 29
exprs(flowFrame-class), 48
exprs,flowFrame-method (flowFrame-class), 48
exprs<-,flowFrame,ANY-method (flowFrame-class), 48
exprs<-,flowFrame,matrix-method (flowFrame-class), 48
fcActionReference,33
fcActionReference (fcReference-class), 31
fcAliasReference,33
fcAliasReference (fcReference-class), 31
fcAliasReference (fcReference-class), 31
fcCompensateReference,33
fcCompensateReference (fcReference-class), 31
fcCompensateReference-class (fcReference-class), 31
fcDataReference,33
fcDataReference (fcReference-class), 31
fcDataReference-class (fcReference-class), 31
fcDataReference (fcReference-class), 31
fcFilterReference,33
fcFilterReference (fcReference-class), 31
fcFilterReference-class (fcReference-class), 31
fcFilterReference (fcReference-class), 31
fcFilterReference-class (fcReference-class), 31
fcFilterResultReference,33
fcFilterResultReference (fcReference-class), 31
fcFilterResultReference-class (fcReference-class), 31
fcFilterResultReference (fcReference-class), 31
fcFilterResultReference-class (fcReference-class), 31
fcNormalizationReference,33
fcNormalizationReference (fcReference-class), 31
fcNormalizationReference-class (fcReference-class), 31
fcNullReference,33
fcNullReference (fcReference-class), 31
fcNullReference-class (fcReference-class), 31
fcReference,32,33,150
fcReference (fcReference-class), 31
fcReference-class, 31
FCSTransTransform,34
fcStructureReference,32,33
fcStructureReference-class (fcReference-class), 31
fcSubsettingReference (fcReference-class), 31
fcSubsettingReference-class (fcReference-class), 31
fcTransformReference,33
fcTransformReference (fcReference-class), 31
fcTransformReference-class (fcReference-class), 31
fcTreeReference,33
fcTreeReference (fcReference-class), 31
fcTreeReference-class (fcReference-class), 31
fcViewReference,33
fcViewReference (fcReference-class), 31
fcViewReference-class (fcReference-class), 31
featureNames (flowFrame-class), 48
featureNames,flowFrame-method (flowFrame-class), 48
filerList,43
filter,11,12,21,25,26,29,30,32,35,
36–44,49,51,56,57,61,63,69,70,
75,76,81,88,90–92,97,98,
100–104,107,115,116,118,120,
124,130,135,136,140,150
filter,filter-method (filter-class), 37
filter,flowFrame,filter-method (filter), 35
filter,flowFrame,filterSet-method (filter), 35
filter,flowFrame,norm2Filter (filter), 35
filter,flowFrame,polygonGate (filter), 35
filter,flowFrame,rectangleGate (filter), 35
filter,flowFrame-method (filter), 35
filter,flowSet,filter-method (filter), 35
filter,flowSet,filterList-method (filter), 35
filter,flowSet,filterSet-method (filter), 35
filter,flowSet,filterSet-method (filter), 35
filter,flowSet,filterSet-method (filter), 35
filter-and-methods,37
filter-and-methods (%on%), 154
filterDetails(filterDetails-methods), 39
filterDetails,filterResult,ANY-method
hyperlog-class, 66
hyperlogGml2 (hyperlogGml2-class), 67
hyperlogGml2-class, 67

identifier, 51, 69
identifier, actionItem-method (actionItem-class), 4
identifier, compensation-method (compensation-class), 18
identifier, fcReference-method (fcReference-class), 31
identifier, filter-method (identifier), 69
identifier, filterList-method (filterList-class), 39
identifier, filterReference-method (identifier), 69
identifier, filterResult-method (identifier), 69
identifier, filterSet, character-method (filterSet-class), 44
identifier, filterSet-method (filterSet-class), 44
identifier, flowFrame-method (identifier), 69
identifier, flowSet-method (flowSet-class), 54
identifier, normalization-method (normalization-class), 93
identifier, NULL-method (view-class), 147
identifier, subsetting-method (subsetting-class), 130
identifier, transform-method (identifier), 69
identifier, transformList-method (transformList-class), 141
identifier, view-method (view-class), 147
identifier<-, methods (identifier), 69
identifier<-, compensation, character-method (compensation-class), 18
identifier<-, filter, character-method (filter), 35
identifier<-, filterList, character-method (filterList-class), 39
identifier<-, flowFrame, ANY-method (identifier), 69
identifier<-, flowFrame-method (identifier), 69
identifier<-, flowSet, ANY-method (flowSet-class), 54
identifier<-, normalization, character-method (normalization-class), 93

identifier<-, subsetting, character-method (subsetting-class), 130
identifier<-, transformList, character-method (transformList-class), 140
initialize, dg1polynomial-method (dg1polynomial-class), 141
initialize, flowFrame-method (flowFrame-class), 22
initialize, flowFrame-method (flowFrame-class), 48
initialize, parameterFilter-method (parameterFilter-class), 97
initialize, ratio-method (ratio-class), 107
initialize, ratiotGml2-method (ratiotGml2-class), 108
initialize, singleParameterTransform-method (singleParameterTransform-class), 120
intersectFilter-class (setOperationFilter-class), 120
intersectFilter-method (filter-and-methods), 37
inverseLogicTransform, 35, 70, 85
invsplitscale (invsplitscale-class), 71
invsplitscale-class, 71
isFCSfile (read.FCS), 110
isNull (fcReference-class), 31
isNull, fcReference-method (fcReference-class), 31
journal (workFlow-class), 149
journal, environment-method (workFlow-class), 149
journal, workFlow-method (workFlow-class), 149

keyword, 50, 56
keyword (keyword-methods), 73
keyword, flowFrame, character-method (keyword-methods), 73
keyword, flowFrame, function-method (keyword-methods), 73
keyword, flowFrame, list-method (keyword-methods), 73
keyword, flowFrame, missing-method (keyword-methods), 73
keyword, flowSet, ANY-method (keyword-methods), 73
keyword, flowSet, list-method (keyword-methods), 73
keyword-methods, 73
keyword<-, (keyword-methods), 73
keyword<-, flowFrame, ANY-method (keyword-methods), 73
ncol,flowFrame-method (flowFrame-class), 48
nodes,workFlow-method (workFlow-class), 149
norm2Filter, 37, 38, 91, 123
norm2Filter (norm2Filter-class), 91
norm2Filter,filter-class (filter-class), 37
norm2Filter-class, 91
normalization, 95, 96, 132
normalization (normalization-class), 93
normalize (normalization-class), 93
normalize,flowSet,normalization-method (normalization-class), 93
normalizeActionItem (normalizeActionItem-class), 94
normalizeView (normalizeView-class), 96
nrow,flowFrame-method (flowFrame-class), 48
parameterFilter, 11, 21, 25, 37, 75, 91, 98, 100, 103, 116, 135
parameterFilter-class, 97
parameters, 50, 75, 98
parameters,compensation-method (compensation-class), 18
parameters,filter-method (parameters), 98
parameters,filterReference-method (parameters), 98
parameters,filterResult-method (parameters), 98
parameters,filterResultList-method (filterResultList-class), 42
parameters,flowFrame,missing-method (parameters), 98
parameters,flowFrame-method (parameters), 98
parameters,manyFilterResult-method (manyFilterResult-class), 88
parameters,normalization-method (normalization-class), 93
parameters,nullParameter-method (parameters), 98
parameters,parameterFilter-method (parameters), 98
parameters,parameterTransform-method (parameters), 98
parameters,ratio-method (parameters), 98
parameters,ratiotGml12-method (ratiotGml12-class), 108
parameters,setParameterOperationFilter-method (parameters), 98
parameters,setParameterOperationTransform-method (setParameterOperationTransform-class), 120
parameters,transform-method (parameters), 98
parameters,transformReference-method (transformReference-class), 143
parameters-class, 99
parameters<- (parameters), 98
parameters<-,dg1polynomial,character-method (dg1polynomial-class), 22
parameters<-,dg1polynomial,parameters-method (dg1polynomial-class), 22
parameters<-,dg1polynomial,transform-method (parameters), 98
parameters<-,flowFrame,AnnotatedDataFrame-method (parameters), 98
parameters<-,parameterFilter,character-method (parameters), 98
parameters<-,parameterFilter,list-method (parameters), 98
parameters<-,parameterFilter,transform-method (parameters), 98
parameters<-,singleParameterTransform,character-method (parameters), 98
parameters<-,singleParameterTransform,transform-method (parameters), 98
parameterTransform-class, 99
parent (view-class), 147
parent,actionItem-method (actionItem-class), 4
parent,NULL-method (view-class), 147
parent,view-method (view-class), 147
pData,flowSet-method (flowSet-class), 54
pData<-,flowSet,data.frame-method (flowSet-class), 54
phenoData,flowSet-method (flowSet-class), 54
phenoData<-,flowSet,ANY-method (flowSet-class), 54
phenoData<-,flowSet,phenoData-method (flowSet-class), 54
plot,flowFrame,ANY-method (flowFrame-class), 48
plot,flowFrame-method (flowFrame-class), 48
plot,flowSet,ANY-method (flowSet-class), 54
INDEX

plot, flowSet-method (flowSet-class), 54
plot, workflow, missing-method (workflow-class), 149
polygonGate, 26, 100, 102, 116
polygonGate (polygonGate-class), 100
polygonGate, filter-class (filter-class), 37
polygonGate-class, 100
polytopeGate, 26, 101, 116
polytopeGate (polytopeGate-class), 102
polytopeGate-class, 102
print, compensateActionItem-method (compensateActionItem-class), 14
print, filterSummary-method (filterSummary-class), 45
print, gateActionItem-method (gateActionItem-class), 60
print, normalizeActionItem-method (normalizeActionItem-class), 94
print, subsettingActionItem-method (subsettingActionItem-class), 131
print, transformActionItem-method (transformActionItem-class), 138
print, view-method (view-class), 147
quadGate (quadGate-class), 103
quadGate-class, 103
quadratic (quadratic-class), 105
quadratic-class, 105
quadGate-class, 103
randomFilterResult, 41, 42
randomFilterResult-class, 107
range (flowFrame-class), 48
range, flowFrame-method (flowFrame-class), 48
rangeGate, 123
ratio, 109
ratio (ratio-class), 107
ratio-class, 107
ratioGml2 (ratioGml2-class), 108
ratioGml2-class, 108
rbind2, flowFrame, flowSet-method (flowSet-class), 54
rbind2, flowSet, flowFrame-method (flowSet-class), 54
rbind2, flowSet, flowSet-method (flowSet-class), 54
rbind2, flowSet, missing-method (flowSet-class), 54
rbind2, flowSet, flowSet-method (flowSet-class), 54
rbind2, flowSet, missing-method (flowSet-class), 54
read.AnnotatedDataFrame, 114
read.FCS, 49, 52, 110, 114
read.FCSHeader, 112
read.flowSet, 55, 57, 112, 113, 153
rectangleGate, 26, 101, 102, 116
rectangleGate (rectangleGate-class), 115
rectangleGate(), 37
rectangleGate, filter-class (filter-class), 37
rectangleGate-class, 115
Rm (fcReference-class), 31
Rm, actionItem, workflow, character-method (actionItem-class), 4
Rm, character, workflow, character-method (workflow-class), 149
Rm, compensateActionItem, workflow, character-method (compensateActionItem-class), 14
Rm, compensateView, workflow, character-method (compensateView-class), 17
Rm, fcNullReference, missing, character-method (fcReference-class), 31
Rm, fcReference, missing, character-method (fcReference-class), 31
Rm, fcReference, workflow, character-method (fcReference-class), 31
Rm, gateActionItem, workflow, character-method (gateActionItem-class), 60
Rm, gateView, workflow, character-method (gateView-class), 62
Rm, normalizeActionItem, workflow, character-method (normalizeActionItem-class), 94
Rm, normalizeView, workflow, character-method (normalizeView-class), 96
Rm, subsettingActionItem, workflow, character-method (subsettingActionItem-class), 131
Rm, subsettingView, workflow, character-method (subsettingView-class), 133
Rm, transformActionItem, workflow, character-method (transformActionItem-class), 138
Rm, transformView, workflow, character-method (transformView-class), 143
Rm, view, workflow, character-method (view-class), 147
sampleFilter (sampleFilter-class), 117
sampleFilter-class, 117
INDEX

sampleNames, flowSet-method (flowSet-class), 54
sampleNames<-, flowSet, ANY-method (flowSet-class), 54
sapply, 56, 60
scaleTransform, 119
setOperationFilter-class, 120
show, boundaryFilter-method (boundaryFilter-class), 11
show, compensateActionItem-method (compensateActionItem-class), 14
show, compensation-method (compensation-class), 18
show, complementFilter-method (setOperationFilter-class), 120
show, ellipsoidGate-method (ellipsoidGate-class), 25
show, expressionFilter-method (expressionFilter-class), 29
show, fcNullReference-method (fcReference-class), 31
show, fcReference-method (fcReference-class), 31
show, filter-method (filter), 35
show, filterList-method (filterList-class), 39
show, filterReference-method (filterReference-class), 40
show, filterResult-method (filterResult-class), 41
show, filterResultList-method (filterResultList-class), 42
show, filters-method (filters-class), 43
show, filterSet-method (filterSet-class), 44
show, filtersList-method (filters-class), 43
show, filterSummary-method (filterSummary-class), 45
show, flowFrame-method (flowFrame-class), 48
show, flowSet-method (flowSet-class), 54
show, gateActionItem-method (gateActionItem-class), 60
show, intersectFilter-method (setOperationFilter-class), 120
show, kmeansFilter-method (kmeansFilter-class), 74
show, manyFilterResult-method (manyFilterResult-class), 88
show, multipleFilterResult-method (multipleFilterResult-class), 90
show, norm2Filter-method (norm2Filter-class), 91
show, normalizeActionItem-method (normalizeActionItem-class), 94
show, polygonGate-method (polygonGate-class), 100
show, polytopeGate-method (polytopeGate-class), 102
show, quadGate-method (quadGate-class), 103
show, rectangleGate-method (rectangleGate-class), 115
show, sampleFilter-method (sampleFilter-class), 117
show, subsetFilter-method (setOperationFilter-class), 120
show, subsetting-method (subsetting-class), 130
show, subsettingActionItem-method (subsettingActionItem-class), 131
show, timeFilter-method (timeFilter-class), 135
show, transform-method (transform-class), 137
show, transformActionItem-method (transformActionItem-class), 138
show, transformFilter-method (transformFilter-class), 140
show, transformMap-method (transformMap-class), 142
show, unionFilter-method (setOperationFilter-class), 120
show, unitytransform-method (unitytransform-class), 145
show, view-method (view-class), 147
show, workFlow-method (workFlow-class), 149
singleParameterTransform, 7, 9, 24, 28, 66, 68, 72, 78, 80, 83, 86, 105, 121, 126, 129
singleParameterTransform-class, 120
sinht (sinht-class), 121
sinht-class, 121
smoothScatter, 50
sort, filterSet-method (filterSet-class), 44
spillover, 19, 20, 122
spillover, flowFrame-method
INDEX

(flowFrame-class), 48
spillover, flowSet-method (spillover), 122
split, 26, 30, 42, 51, 56, 75, 76, 92, 101, 104, 116, 118, 130, 136
split (split-methods), 123
split, flowFrame, ANY-method
(split-methods), 123
split, flowFrame, character-method
(split-methods), 123
split, flowFrame, factor-method
(split-methods), 123
split, flowFrame, filter-method
(split-methods), 123
split, flowFrame, filterSet-method
(split-methods), 123
split, flowFrame, logicalFilterResult-method
(split-methods), 123
split, flowFrame, manyFilterResult-method
(split-methods), 123
split, flowFrame, multipleFilterResult-method
(split-methods), 123
split, flowFrame, numeric-method
(split-methods), 123
split, flowSet, ANY-method
(split-methods), 123
split, flowSet, character-method
(split-methods), 123
split, flowSet, factor-method
(split-methods), 123
split, flowSet, filter-method
(split-methods), 123
split, flowSet, filterResult-method
(split-methods), 123
split, flowSet, filterResultList-method
(filterResultList-class), 42
split, flowSet, list-method
(split-methods), 123
split, flowSet, numeric-method
(split-methods), 123
split-methods, 123
splitscale (splitscale-class), 126
splitscale-class, 126
splitScaleTransform, 127
splom, 50
squareroot (squareroot-class), 128
squareroot-class, 128
Subset, 12, 26, 30, 36, 92, 101, 116, 118, 129, 136
subset, 129, 130
Subset, flowFrame, filter-method
(Subset), 129
Subset, flowFrame, logical-method
(Subset), 129
Subset, flowFrame-method (Subset), 129
Subset, flowSet, ANY (Subset), 129
Subset, flowSet, ANY-method (Subset), 129
Subset, flowSet, filterResultList-method
(Subset), 129
Subset, flowSet, list-method (Subset), 129
subsetFilter-class
(setOperationFilter-class), 120
subsetFilter-method
(filter-and-methods), 37
subsetting, 132, 133
subsetting (subsetting-class), 130
subsetting-class, 130
subsettingActionItem
(subsettingActionItem-class), 131
subsettingActionItem-class, 131
subsettingView (subsettingView-class), 133
subsettingView-class, 133
summarizeFilter
(summarizeFilter-methods), 134
summarizeFilter, filterResult, filter-method
(summarizeFilter-methods), 134
summarizeFilter, filterResult, filterReference-method
(summarizeFilter-methods), 134
summarizeFilter, filterResult, parameterFilter-method
(summarizeFilter-methods), 134
summarizeFilter, filterResult, subsetFilter-method
(summarizeFilter-methods), 134
summarizeFilter, logicalFilterResult, norm2Filter-method
(summarizeFilter-methods), 134
summarizeFilter, logicalFilterResult, parameterFilter-method
(summarizeFilter-methods), 134
summarizeFilter, multipleFilterResult, parameterFilter-method
(summarizeFilter-methods), 134
summarizeFilter-methods, 134
summary, 37
summary, filter-method (filter), 35
summary, filterReference-method
(filterReference-class), 40
summary, filterResult-method
(filterSummary-class), 45
summary, filterResultList-method
(filterResultList-class), 42
summary, flowFrame-method
(flowFrame-class), 48
summary, flowSet-method (flowSet-class), 54
summary, gateActionItem-method
INDEX

(gateActionItem-class), 60
summary, gateView-method
(gateView-class), 62
summary, logicalFilterResult-method
(logicalFilterResult-class), 81
summary, manyFilterResult-method
(manyFilterResult-class), 88
summary, multipleFilterResult-method
(multipleFilterResult-class), 90
summary, rectangleGate-method
(rectangleGate-class), 115
summary, subsetFilter-method
(setOperationFilter-class), 120
summary, transform-method
(transform-class), 137
summary, workFlow-method
(workFlow-class), 149
tail, flowFrame-method
(flowFrame-class), 48
timeFilter, 135
timeFilter (timeFilter-class), 135
timeFilter-class, 135
timeLinePlot, 136
toTable (filterSummary-class), 45
toTable, filterSummary-method
(filterSummary-class), 45
toTable, filterSummaryList-method
(filterSummaryList-class), 47
transform, 6, 7, 9, 10, 16, 19, 20, 22, 24, 28,
32, 37, 38, 51, 66, 68–70, 72, 75,
77–80, 83, 84, 86–88, 99, 105, 106,
108, 109, 119, 121, 126, 128, 129,
138–146, 150
transform (transform-class), 137
transform, flowFrame-method
(flowFrame-class), 48
transform, flowSet-method
(flowSet-class), 54
transform, missing-method
(transform-class), 137
transform-class, 137
transformActionItem, 5, 16, 62, 96, 133, 150
transformActionItem
(transformActionItem-class), 138
transformActionItem-class, 138
transformation, 7, 9, 16, 22, 24, 28, 66, 68,
72, 75, 78, 80, 83, 86, 105, 108, 109,
121, 126, 129, 143, 146
transformation-class, 139
transformFilter, 142
transformFilter
(transformFilter-class), 140
transformFilter-class, 140
transformList, 38, 140, 142
transformList (transformList-class), 141
transformList-class, 141
transformMap, 141
transformMap (transformMap-class), 142
transformMap-class, 142
transformReference
(transformReference-class), 143
transformReference-class, 143
transformReferences, 20
transforms, 20
transformView, 18, 63, 97, 134, 147, 148, 150
transformView (transformView-class), 143
transformView-class, 143
tree (workFlow-class), 149
tree, environment-method
(workFlow-class), 149
tree, workFlow-method (workFlow-class), 149
truncateTransform, 145
undo (workFlow-class), 149
unionFilter-class
(setOperationFilter-class), 120
uniroot, 10
unityTransform (unityTransform-class), 145
unityTransform-class, 145
validFilters, 146
varLabels, flowSet-method
(flowSet-class), 54
varLabels<-, flowSet, ANY-method
(flowSet-class), 54
varLabels<-, flowSet, ANY-method
(flowSet-class), 54
varMetadata, flowSet-method
(flowSet-class), 54
varMetadata<-, flowSet, ANY-method
(flowSet-class), 54
vector, 99
view, 5, 15, 16, 18, 32, 61–63, 95–97,
132–134, 138, 139, 144, 149, 151
view (view-class), 147
view-class, 147
views, 5, 150, 151
views, workFlow-method (workFlow-class), 149
views, workFlow-method (workFlow-class), 149
workFlow (workFlow-class), 149
workFlow-class, 149
write.FCS, 152, 153
write.flowSet, 153

xyplot, 43
xyplot, formula, gateView-method
  (gateView-class), 62
xyplot, formula, view-method
  (view-class), 147
xyplot, view, missing-method
  (view-class), 147